EE 4330 Final Project Report

Nurudeen Agbonoga, Dulce Jazmin Almanza Mosqueda

Electrical Engineering Department, University of Texas at Arlington
710 South Nedderman, Drive Arlington, Texas, United States of America

[1]nurudeen.agbonoga@mavs.uta.edu

[2]dulce.almanzamosqueda@mavs.uta.edu

***Abstract*— The purpose of this project is to demonstrate some of the key principles of a digital communication system. This will involve implementing the following digital compression techniques to an input video sequence file using MATLAB software: Pulse code modulation, differential pulse code modulation and discrete cosine transformation.**

I. Introduction

Throughout the past 10 years, there has been a rapid expansion of digital communication technologies. As a result, digital communication is quickly replacing outdated analog communication system.

This rapid change is due to the many advantages that digital communication has. Digital messages are transmitted electronically as waveforms through versatile, powerful, inexpensive high-speed microprocessors. The use of a microcontroller provides the opportunity to quickly adapt messages, while also providing an enhanced immunity of digital signals to noise and interference. This immunity is due to digital messages being limited to a finite number of symbols. Therefore, the receiver has a limited amount of options when selecting between multiple possible pulse received. As a result, the receiver's decision can be made with reasonable certainty even through distortion and noise. Analog signals on the other hand contain an infinite number of symbols, making them receptive to noise and distortion.

Digital systems can also be transmitted at much longer distances. This is due to the viability of regenerative repeaters and network nodes. Repeaters are placed along the communication path of a digital system at distances short enough to reduce distortion, therefore pulse detection can be done at a high accuracy. Using multiple repeaters along the communication path prevents the accumulation of noise and distortion because a new "clean" pulse is being transmitted to the next repeater station.

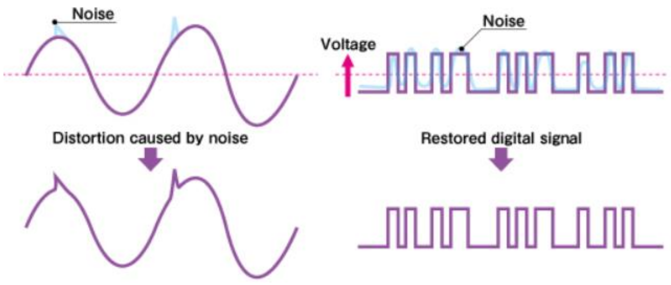


Figure 1. Analog vs Digital signals

Although there are many differences between analog and digital signals, a meeting ground exists between them. Analog signals can be converted to digital signal through A/D conversion.

Analog-to-Digital converters allow digital communication system to convey analog source signals such as video and audio. This is a two-step process which consists of *sampling* and *quantizing* analog signals.

The **sampling theorem** states that in order for a signal to be reconstructed, it must be sampled at no less than twice the highest frequency in the signal spectrum. Sampling is the process of reducing an analog continuous signal to a digital signal.

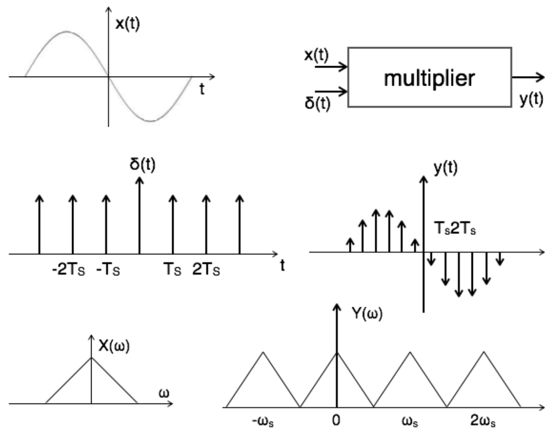


Figure 2. Sampling Theorem Demonstrated

Through **quantization**, each sample is approximated to the nearest quantized level. This is done without compromising the signal quality. The number of levels, L are quantized partitions of the signal range. The larger the number of levels, the more information that will be captured.

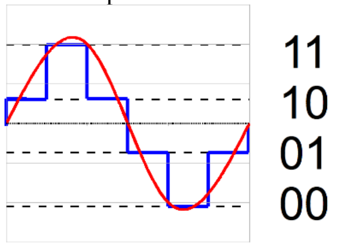


Figure 3. Quantization with 4 levels

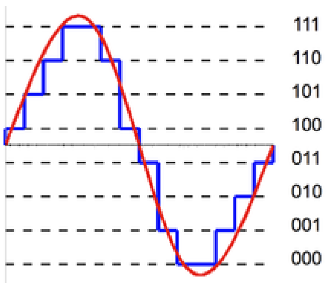


Figure 4. Quantization with 8 levels.

II. Methods of Video Compression

A. Quantization and Pulse Code Modulation

Pulse Code Modulation is the most widely used form of pulse modulation. This involves converting an *L-ary* signal into a binary signal by using pulsing coding. It consists of three steps: sampling and quantization (as previously mentioned), and additional step known as encoding. Consider an analog signal *m(t)* that lies in the range $(-m_p, m_p)$ and is partitioned into L sublevels, each having a magnitude of $\Delta v = 2m_p/L$.

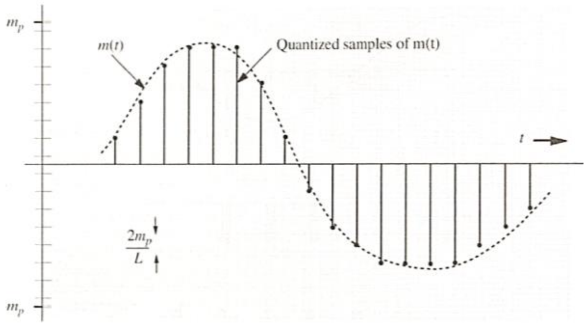


Figure 5. PCM at L-Levels

Each of the L levels that is to be transmitted is assigned an N-digit binary code (N bits). The value of N is derived using the following,

$$2^N = L$$

therefore,

$$N = \log_2 L$$

To transmit binary data, a distinct pulse shape needs to be assigned to each bit. This can be done in several ways. One possible method of doing this is to assign a negative pulse to a binary 0 and a positive pulse to a binary 1 as shown below:

$$\begin{cases} 0, & -p(t) \\ 1, & p(t) \end{cases}$$

This will result in a binary signal. Since the message signal *m(t)* is band-limited to B Hz the minimum required bandwidth for transmission is *2B* samples per second. Therefore, a total of 2*NB* bits/s is required. Unit bandwidth of 1 Hz can transmit two pieces of information per second, there the minimum channel of transmission bandwidth is given by,

$$B_T = NB\ Hz$$

Assuming that the error is equally likely to lie anywhere in the range ($-\Delta v/2, \Delta v/2$) the mean square quantizing error is given by,

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

The signal-to-noise ratio , SNR can be found using the following,

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

where P_m is the power of the message signal *m(t)*, and m_p is the peak amplitude. SNR is an indication of the quality of the received signal. Ideally, SNR would be constant for all values of the message signal power. However, it varies from speaker to speaker due to the nonconstant values of signal power.

This is a result of having a uniform value of quantizing steps, $\Delta v = 2m_p/L$ and the quantizing error being directly proportional to the square of the step size.

To resolve this problem, nonuniform quantization is introduced. This consists of using smaller steps for smaller amplitudes.

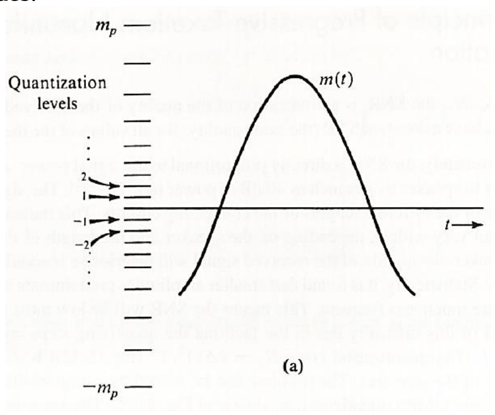


Figure 6a. Non-uniform quantization

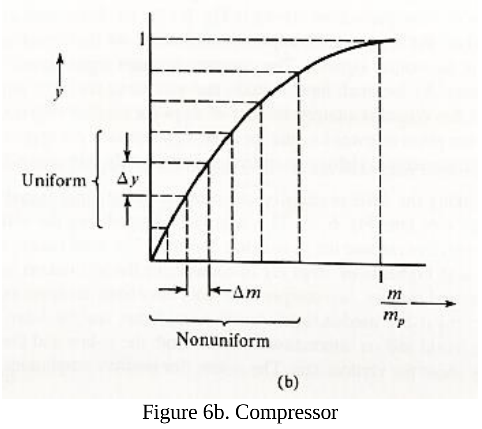


Figure 6b. Compressor

As shown in Figure 6b. the horizontal axis is normalized message signal and vertical axis is the output signal. The quantization noise is now nearly proportional the signal power, since Δm contains a larger number of steps when m is small.

Therefore, the SNR is practically independent of the input signal power.

There are two compression laws that have been accepted as the desired standards: the μ-law and the A-law. The parameters μ and A determine the degree of compression. Only the μ-law is used in North America. It is given by,

$$y = \frac{1}{\ln(1+\mu)} \cdot \ln({}^{\mu m}/_{m_p})$$

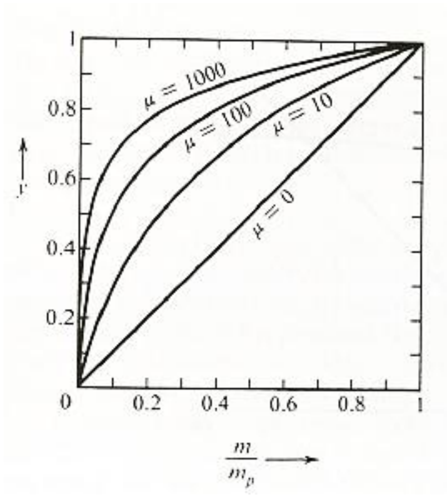


Figure 7. μ-law characteristic

To obtain a nearly constant SNR value μ should be greater than 100 for an input signal power of 40 dB.

The A-Law is given by

$$y = \begin{cases} \dfrac{A}{1+\ln A} \cdot ({}^{m}/_{m_p}), & 0 \le \dfrac{m}{m_p} \le \dfrac{1}{A} \\ \dfrac{A}{1+\ln A}(1+\ln({}^{Am}/_{m_p}), & x \ge 0 \end{cases}$$

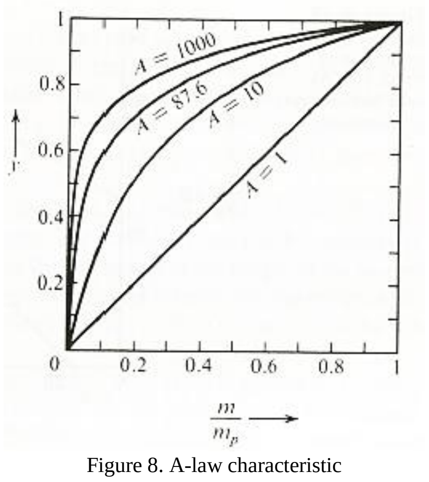


Figure 8. A-law characteristic

To restore samples after being compressed, an expander must be used by the receiver. The compressor and the expander together are called the compandor. The compandor is utilizes a semiconductor diode. The V-I characteristic of diodes provide the desired form in the first quadrant,

$$V = \frac{KT}{q}\ln(1 + {}^{I}/_{I_s})$$

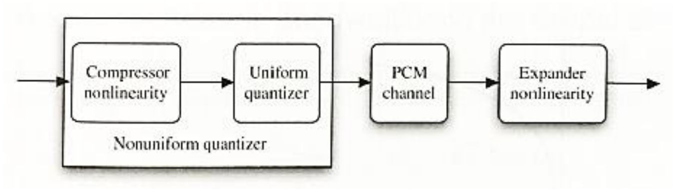


Figure 9. Utilization of compressor and expander for nonuniform quantization

B. Differential Pulse Code Modulation

PCM has the drawback of generating too many bits and requiring too much bandwidth to transmit. When working with analog messages, a good guess can be made about a sample value based of the information gathered from the previous values. Therefore, the values of samples depend on each other and are not independent. There is a great deal of redundancy in Nyquist samples. Exploitation of this redundancy leads to encoding a signal with fewer bits.

The approach of DPCM is to transmit the difference between the successive samples rather than the sample values themselves.

For example, if $m[k]$ is the kth sample,

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

would be transmitted rather than $m[k]$. Using $d[k]$ and several samples of $m[k-1]$, $m[k]$ can be reconstructed at the receiver. It is important to note that $d[k]$ is much smaller than the sample values. For this reason, the peak amplitude of the message signal is reduced considerably. Recall that the size of quantization levels is defined by $\Delta v = 2m_p/L$. Therefore, when the peak amplitude is reduced, the quantization level size is also reduced as well as the quantization noise. This means that for a given transmission bandwidth SNR can be increased, likewise for a given SNR the transmission bandwidth can be reduced.

This can be improved by estimating the value of the kth sample from a knowledge of previous sample values. This will provide the opportunity for the difference to be predicted as well. Therefore,

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

can be transmitted instead, where $d_p\,[k]$ is the predicted difference and $\hat{m}[k]$ is the predicted value for the kth sample. If the prediction was done well, the value of $\hat{m}[k]$ will be close to the value of $m[k]$. Therefore, the difference should be very small. To verify, at the receiver end if the predicted value is added to the predicted difference, the result would be $m[k]$ as shown below,

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

The difficulty when working with this method is that rather than receiving past sample values of $m[k-2]$, $m[k-1]$,…, as well as $d[k]$, we receive their quantized versions $m_q[k-2]$, $m_q[k-1]$,…, and d_p $[k]$ instead. Hence $\widehat{m}$ $[k]$ is not determined but instead $\widehat{m}_q[k]$ is.

Therefore, it is better to focus on determining $\widehat{m}_q[k]$ and estimate $m_q[k]$. In this case the quantized difference,

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

will now be transmitted via PCM. At the receiver, $\widehat{m}_q[k]$ will be generated and $m_q[k]$ can be reconstructed from the received $d_p[k]$.

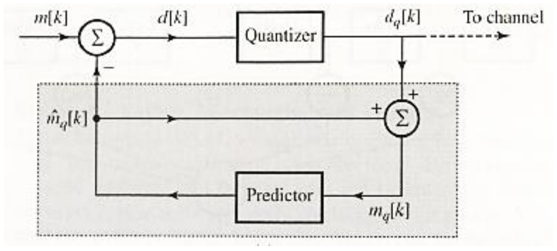


Figure 10. DPCM transmitter

It is shown that the predictor input is $m_q[k]$ and the output is $\widehat{m}_q[k]$ and the difference is

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

This is quantized to yield

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

where $q[k]$ is the quantization error. The predictor is fed back to its input so that the predictor input $m_q[k]$ is

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

An interesting thing to not is that that feedback for the transmitter is identical to the block diagram for the receiver

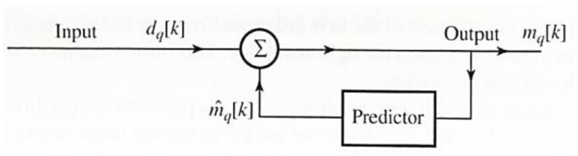


Figure 11. DPCM receiver

This is because the inputs in both cases are also the same, $d_q[k]$. Therefore, the predictor output must be $\widehat{m}_q[k]$ and the receiver output. The $m[k]$ can be received plus the quantization noise associated with the difference signal. The received signal $m_q[k]$ can then be decoded and passed through a low-pass filter.
The signal to noise ratios is calculated as,

SNR = average power of signal/average power of noise

= 20*$\log_{10}$ (magnitude of original video/magnitude of original video – DPCM output)

In order to compare the improvement of DPCM from PCM the signal power ration can be found,

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

where G_p is the SNR improvement, P_d is the power of *d(t)* and P_m is the power of *m(t)*.

C. *Discrete Cosine Transformation*

Video and television face a tremendous challenge. Due to their high video bandwidth, applying sampling and quantization directly lead to an uncompressed digital video signal. The challenge is finding a way to reduce the bandwidth required for video transmission. MPEG, the Moving Picture Experts Group has developed new compression techniques. It is noted that a relatively small number of pixels change frame to frame. Therefore, by only transmitting the changes, the transmission bandwidth can be reduced significantly.

There are two types of MPEG compression which eliminate redundancies in audiovisual signals:

1. Temporal or *interframe* compression which predicts the interframe motion and removes redundancy
2. Spatial or *intraframe* compression, which forms a block identifier for a group of pixels that have the same characteristics for each frame. Only the block identifier is transmitted.

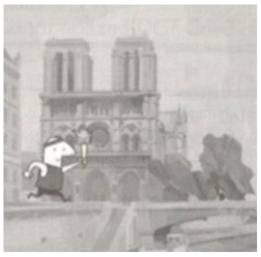


Figure 12. Frame 1

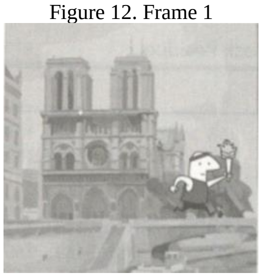


Figure 13. Frame 2

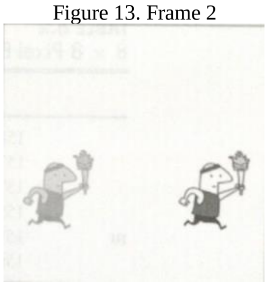


Figure 14. Information transferred between frames 1 and 2

Video compression begins by using an encoder to convert analog video signals to a digital format on a pixel-to-pixel basis. Each video frame is divided into 8 x 8 pixel blocks which are analyzed through the encoder to determine which should be transmitted. Only blocks containing significant changes from frame to frame are transmitted. This is done in a two- step process.

First, a motion estimator identifies areas of groups of blocks from a preceding frame and sends a vector displacement to a predictor in the decoder. The frame difference is known as the residual

Secondly, the residual is then transformed into a more compact form by means of a discrete cosine transform, DCT. DCT uses a numerical value to represent each pixel and normalizes that value for a more efficient transmission. It is in the form,

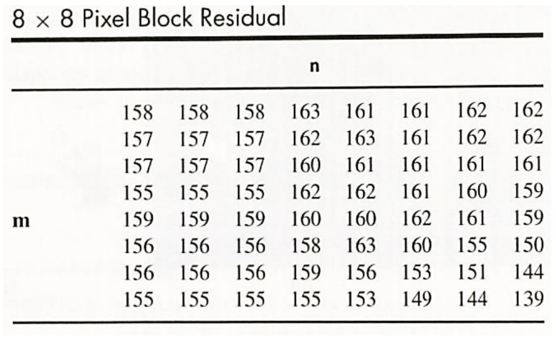
$$F(i,j) = \sum_{n=0}^{N-1}\sum_{m=0}^{N-1} f(n,m)\cos[{}^{(2n+1)*j\pi}/_{2N}]\cos[{}^{(2n+1)*k\pi}/_{2N}]$$

The inverse transform is given by,

$$f(n,m) = \frac{1}{N^2}\sum_{n=0}^{N-1}\sum_{m=0}^{N-1} F(j,k)\cos[{}^{(2n+1)*j\pi}/_{2N}]\cos[{}^{(2n+1)*k\pi}/_{2N}]$$

The DCT is multiplied, for an 8 x 8 block by the expression *C(j)C(k)/4*, where

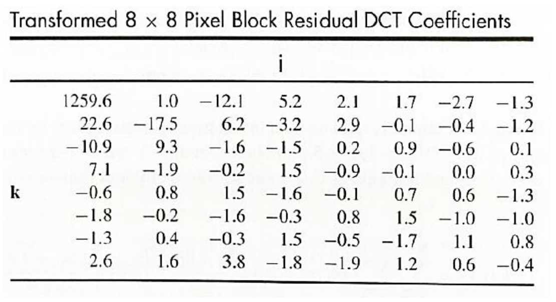
$$C(x) = \begin{cases} \frac{1}{\sqrt{2}}, & x = 0 \\ 1, & O.W. \end{cases}$$



8 × 8 Pixel Block Residual

	n							
	158	158	158	163	161	161	162	162
	157	157	157	162	163	161	162	162
	157	157	157	160	161	161	161	161
	155	155	155	162	162	161	160	159
m	159	159	159	160	160	162	161	159
	156	156	156	158	163	160	155	150
	156	156	156	159	156	153	151	144
	155	155	155	155	153	149	144	139

Figure 15. Pixel Block Residual



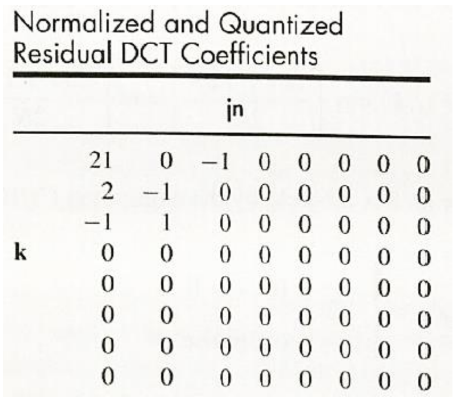
Transformed 8 × 8 Pixel Block Residual DCT Coefficients

	i							
	1259.6	1.0	−12.1	5.2	2.1	1.7	−2.7	−1.3
	22.6	−17.5	6.2	−3.2	2.9	−0.1	−0.4	−1.2
	−10.9	9.3	−1.6	−1.5	0.2	0.9	−0.6	0.1
	7.1	−1.9	−0.2	1.5	−0.9	−0.1	0.0	0.3
k	−0.6	0.8	1.5	−1.6	−0.1	0.7	0.6	−1.3
	−1.8	−0.2	−1.6	−0.3	0.8	1.5	−1.0	−1.0
	−1.3	0.4	−0.3	1.5	−0.5	−1.7	1.1	0.8
	2.6	1.6	3.8	−1.8	−1.9	1.2	0.6	−0.4

Figure 16. Pixel Block Residual DCT coefficients

Figures 15 and 16 depict the pixel block values before and after DCT. It is evident from Figure 16 that are relatively less meaningful elements. These are significant values relative t the values centered about the origin. Therefore,
most of the matrix may be assumed to be 0 as shown in Figure 17. Upon inverse transformation, the original values are quite accurately reproduced. This process reduces the amount of data that must be transmitted, therefore making video compression a lot more achievable because this process reduces the amount of data that is transmitted by perhaps a factor of 8 to 10 on average. The size of the transmitted residual may be that of an individual block or even that of the entire picture.

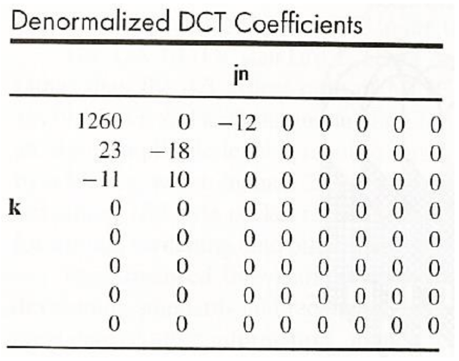
The transformed matrix values in Figure 17 are normalized and quantized so that most of the values in the block matrix are less than 1.



Normalized and Quantized Residual DCT Coefficients

	jn							
	21	0	−1	0	0	0	0	0
	2	−1	0	0	0	0	0	0
	−1	1	0	0	0	0	0	0
k	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0

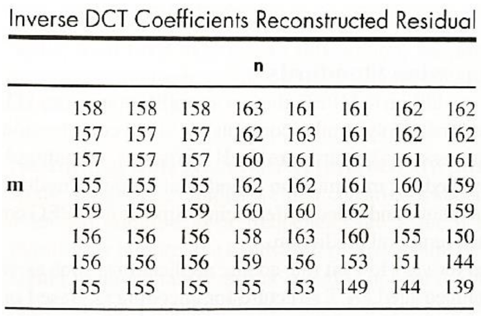
Figure 17. Normalized and Quantized Residual DCT Coefficients



Denormalized DCT Coefficients

	jn							
	1260	0	−12	0	0	0	0	0
	23	−18	0	0	0	0	0	0
	−11	10	0	0	0	0	0	0
k	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0

Figure 18. Denormalized DCT Coefficients



Inverse DCT Coefficients Reconstructed Residual

	n							
	158	158	158	163	161	161	162	162
	157	157	157	162	163	161	162	162
	157	157	157	160	161	161	161	161
m	155	155	155	162	162	161	160	159
	159	159	159	160	160	162	161	159
	156	156	156	158	163	160	155	150
	156	156	156	159	156	153	151	144
	155	155	155	155	153	149	144	139

Figure 19. Inverse DCT Coefficient Reconstructed

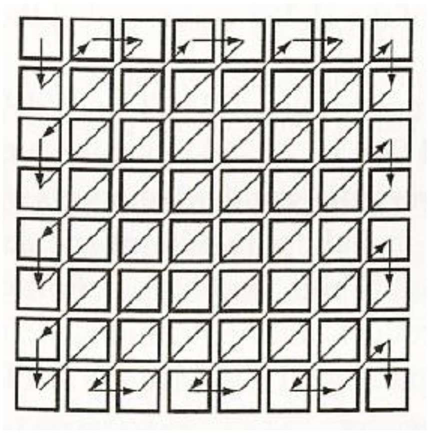


Figure 20. Zigzag DCT

III. Procedure

Begin by reading a video input signal into MATLAB. To read a video file the user can use the built in VideoReader(filename) function. This function will simply allow MATLAB to accept the file as an input. The user must still have a routine that defines the frames of the video signal. For this purpose, the following function was used:

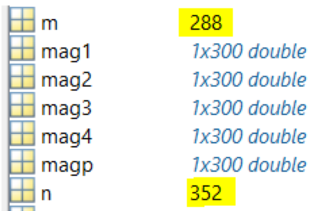
```
function output = readData(filename)
videoObject = VideoReader(filename);
output = zeros([videoObject.Height
videoObject.Width
videoObject.Duration*videoObject.FrameRate],
'uint8');
i = 1;
while hasFrame(videoObject)
    frameYUV = readFrame(videoObject);
    output(:,:,i) = frameYUV(:,:,1);
    i = i + 1;
end
end
```

i. PCM – 16 Level Quantization

Recall that the first step for PCM is to sample the signal. In order to do this, the user must obtain the size of the data array being used. After the video file has been read the following global variable can be used to obtain the size of the data array:

```
[m,n] = size(p(:,:,1));
```

As seen in the Workspace, the values of m and n are 288 and 352, respectively.



m	288
mag1	1x300 double
mag2	1x300 double
mag3	1x300 double
mag4	1x300 double
magp	1x300 double
n	352

Figure 21. m x n data array values

This video contains a total of 300 frames and a 288 x 352 data array. To apply 16 level quantization, both m and n will be divided into 16 equal sections. Note that since this is not a square matrix, therefore some information will be lost on either end. After the array is divided into 16 equal parts on either end, the individual ranges can be examined. The value in the middle of each range will be used for testing. For this purpose, we will only consider the values from 0 to 256 on either end. 256 divided into 16 equal parts will result in each section having 16 data points.

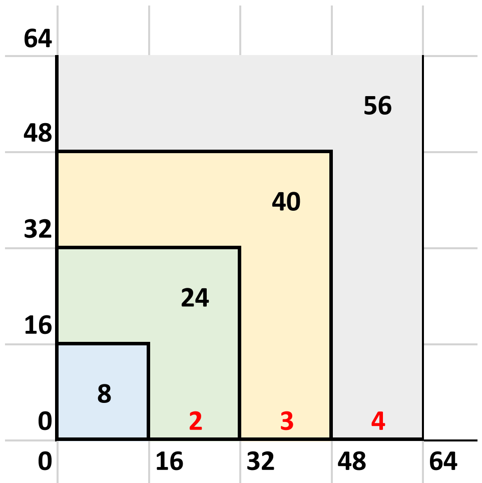


Figure 22. Values for first 4 Levels of Quantization

Figure 22 shows the values for the first four levels of quantization. The horizontal values are the column values for the data array. The vertical values are the row values. The first, second, third, and fourth levels are shaded in blue, green, orange, and grey respectively. The values inside which level in black ink represent the middle value in each range that is to be used for testing. The values in red ink represent the level value.

For example, the first range which is shaded in blue is to be examined is from 0 to 16. This will create a 16 x 16 matrix within the data array. The value in the middle of the range will be tested. In this level the middle value would be 8.

The second level which is shaded in green, is be examined from 17 to 32, creating a 32 x 32 matrix, that excludes the values from the first level. In this level the middle value would be 24, this will be the value that will be tested.

The third level which is shaded in orange, is be examined from 33 to 47, creating a 48 x 48 matrix that will exclude both the values from the first level and the second level. In this level the middle value would be 40, this will be the value that will be tested.

The fourth level which is shaded in grey, is to be examined from 49 to 64, creating a 64 x 64 matrix that will exclude all the values from the previous levels. In this level the middle value would be 56, this will be the value that will be tested.

To implement this in MATLAB, a global variable must be used to count the value of the picture frame, the rows, and the columns of the data array. In this case count1 was assigned to track the value of the frame, count2 was used for the columns and count3 was used for the rows. When working with arrays, for loops are often used to manipulate the values within the array. This can be initialized using the following,

```
for count1 = 1:300
    for count2 = 1:352
        for count3 = 1:288
```

To assign the test values that are to be used for each level the following can be done,

```
if q(count3,count2,count1)>=0 &&
q(count3,count2,count1)<= 15
                q(count3,count2,count1) = 8;

elseif q(count3,count2,count1)>= 17 &&
q(count3,count2,count1)<= 32
                q(count3,count2,count1) = 24;

elseif q(count3,count2,count1)>= 33 &&
q(count3,count2,count1)<= 47
                q(count3,count2,count1) = 40;

elseif q(count3,count2,count1)>= 49 &&
q(count3,count2,count1)<= 64
                q(count3,count2,count1) = 56;
```

This can be continued until the 16 level is reached. Each range will be examining the next 16 values and using the middle value for testing. Figure 23 shows the values for all 16 levels. Note that the extrema values of 288 and 352 are not reached. The complete code including the remaining levels can be found in the appendix.

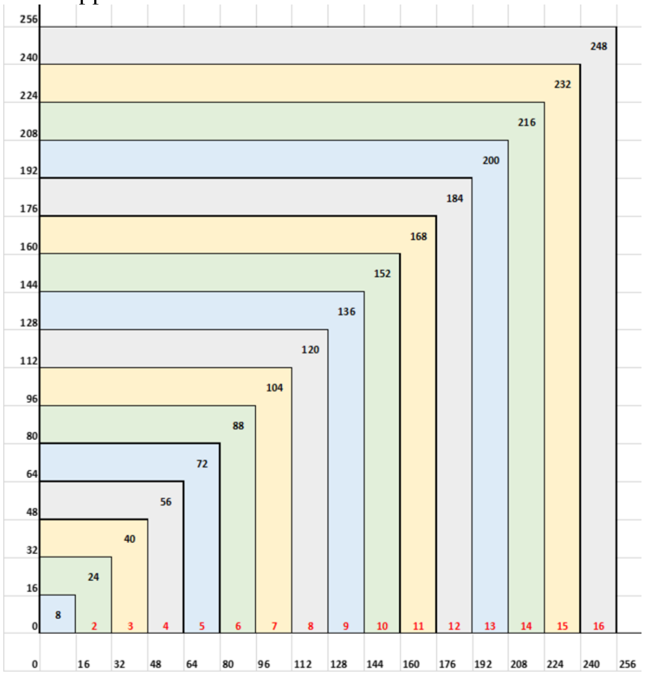


Figure 23. Values for all 16 Levels of Quantization

For the sake of comparison, quantization can be done at different levels. For example, observe the result when the number of levels is reduced by one-half.

In this case, there will only be 8 levels. 256 divided into 8 equal parts will result in each section will have a total 32 data points.

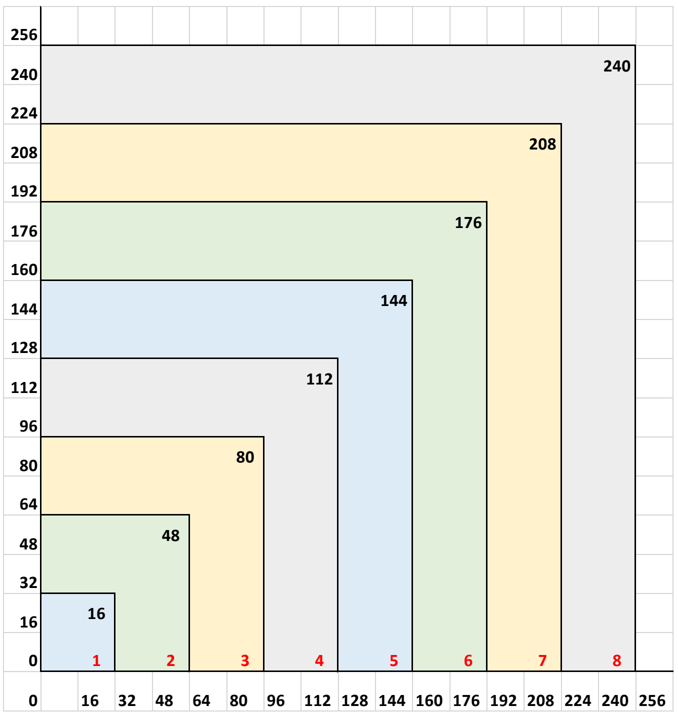


Figure 24. Values for 8 levels

In this case, the first level is to be examined is from 0 to 32, and the middle value is 16. The second level is to be examined from 33 to 64 excluding the first level, with a middle value of 48 and so forth.

Consider dividing the number of levels by two once more. In this case there will only be 4 levels. 256 divided into 4 equal parts will result in each section having a total of 64 data points. Furthermore, dividing the number of levels by two once more will result in only 2 levels where 256 would be split down the middle. The same method used for 16 levels can be applied for each in MATLAB, see appendix for this code.

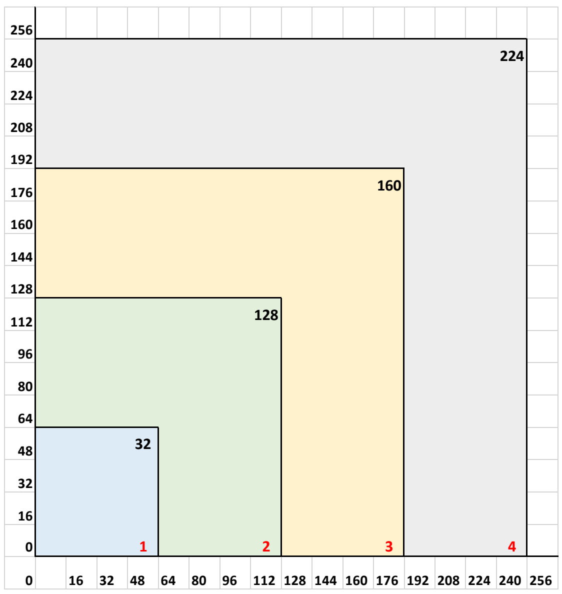


Figure 25. Values for 4 Levels of Quantization

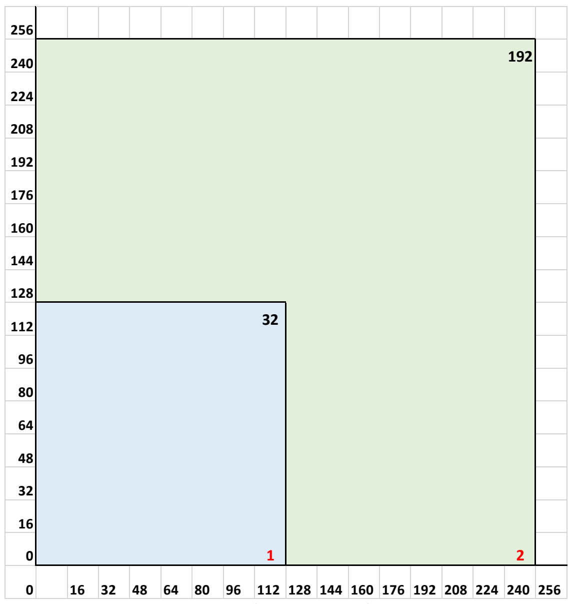


Figure 26. Values for 2 levels of quantization

The SNR can be calculated in order to compare the quality resulting for each level of quantization. This can be done using the following formula:

$$SNR = 20\log[{}^{mag(X)}/_{norm(X)}]$$

where X is the array resulted from each level of quantization. The results can be found in the Results section.

ii. DPCM

a. Derivation of DPCM Predictor

Recall that DPCM entails predicting samples based on previous pixels. First $\hat{x}$ must be derived, which takes the previous pixel into account. The range of the error $= X - Y$ is greater than the range of the individual random variables. The range of error can be reduced by minimizing the variance:

First, to find the smallest a and be such that

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

where $\hat{X}$ is the predicted value of the current pixel, $\hat{x}$ must equal Y

$$\hat{X} = Y$$

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

From calculus it is know that to minimize a, the first derivative is taken with respect to a and set equal to,

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

which results in,

$$\begin{aligned} 0 &= 2E[(X - aY + b)(-Y)] \\ &= -\text{E[XY]} + a\text{E[Y2]} + b\text{E[Y]} \\ &= a\text{E[Y2]} + b\text{E[Y]} = \text{E[XY]} \qquad (5) \end{aligned}$$

To minimize b,

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

which results in,

$$\begin{aligned} 0 &= 2\text{E}[(\text{X} - a\text{Y} - b)(-1)] \\ &= -\text{E[X]} + a\text{ E[Y]} + b \\ &= a\text{ E[Y]} + b = \text{E[X]} \end{aligned}$$

therefore,

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

Substituting equation (6) in (5) the following is obtained,

$$\begin{aligned} &a\text{ E}[\text{Y}^2] + [\text{ E[X]} - a\text{ E[Y]}]\text{ E[Y]} = \text{E[XY]} \\ &a\text{ }[\text{E}[\text{Y}^2] + \text{E}^2[\text{Y}]] = \text{E[XY] - E[X] E[Y]} \\ &\text{E[XY] - E[X] E[Y]} = \text{Cov(X, Y)} \\ &a = (\text{E[XY] - E[X] E[Y]}) / ([\text{E}[\text{Y}^2] + \text{E}^2[\text{Y}]]) \\ &= \text{Cov(X, Y)} / (\sigma_\text{Y})^2 \end{aligned}$$

therefore,

$$a = \frac{\text{Cov}(X,Y)}{(\sigma_y)^2}$$

Plugging in a results in,

$$b = E[X] - aE[Y]$$

$$b = E[X] - \frac{\text{Cov}(X,Y)}{(\sigma_y)^2} \cdot E[Y]$$

recall,

$$\rho = \frac{\text{Cov}(X,Y)}{\sigma_X \sigma_Y}$$

therefore,

$$\text{Cov}(X,Y) = \rho\sigma_X\sigma_Y$$

Plugging this into a,

$$a = \frac{\rho\sigma_X\sigma_Y}{(\sigma_y)^2} = \frac{\rho\sigma_X}{\sigma_Y}$$

Plugging this into b,

$$\begin{aligned} b &= E[X] - \frac{\text{Cov}(X,Y)}{(\sigma_y)^2} \cdot E[Y] \\ &= E[X] - \frac{\rho\sigma_X}{\sigma_Y} \cdot E[Y] \end{aligned}$$

The values of both a and b can now be plugged in to find $\hat{X}$

$$\begin{aligned} \hat{X} &= aY + b \\ &= \frac{\rho\sigma_X}{\sigma_Y}Y + E[X] - \frac{\rho\sigma_X}{\sigma_Y} \cdot E[Y] \\ &= (Y - E[Y]) \cdot \frac{\rho\sigma_X}{\sigma_Y} + E[X] \end{aligned}$$

b. *Derivation of DPCM 3rd order Predictor*

The range of the error = X – Y is greater than the range of the individual random variables. But we can reduce the range of the error by minimizing the variance; e^2.
Min $E(e^2)$ i.e. find the smallest a and b such that

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

where Y_1 is previous pixel to the current pixel X on the same, Y_2 is the pixel Diagonally to the left on the previous line and, Y_3 is the pixel in equivalent position on previous frame

$$\text{Min } E[(X - \hat{X})^2] = \text{Min } E[(X - aY_1 - bY_2 - cY_3)^2]$$

Minimizing each of the following results in ,

$d/da\ [E[(X - aY_1 - bY_2 - cY_3)^2] = 0$

$d/db\ [E[(X - aY_1 - bY_2 - cY_3)^2] = 0$

$d/dc\ [E[(X - aY_1 - bY_2 - cY_3)^2] = 0$

$2E[(X - aY1 - bY2 - cY3)(-Y1)] = 0$
$2E[(X - aY1 - bY2 - cY3)(-Y2)] = 0$
$2E[(X - aY1 - bY2 - cY3)(-Y3)] = 0$

$aE[Y1^2] + bE[Y1Y2] + cE[Y1Y3] = E[XY1]$
$aE[Y1Y2] + bE[Y2^2] + cE[Y2Y3] = E[XY2]$
$aE[Y1Y3] + bE[Y2Y3] + cE[Y3^2] = E[XY3]$

Next, using MATLAB and Cramer's rule to solve for a, b and c:

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;

a =

(EXY1*EY2Y3^2 - EXY1*EY22*EY32 + EXY3*EY22*EY1Y3 + EXY2*EY32*EY1Y2 - EXY2*EY1Y3*EY2Y3 - EXY3*EY1Y2*EY2Y3)/(EY32*EY1Y2^2 - 2*EY1Y2*EY1Y3*EY2Y3 + EY22*EY1Y3^2 + EY12*EY2Y3^2 - EY12*EY22*EY32)

b =

(EXY2*EY1Y3^2 - EXY2*EY12*EY32 + EXY3*EY12*EY2Y3 + EXY1*EY32*EY1Y2 - EXY3*EY1Y2*EY1Y3 - EXY1*EY1Y3*EY2Y3)/(EY32*EY1Y2^2 - 2*EY1Y2*EY1Y3*EY2Y3 + EY22*EY1Y3^2 + EY12*EY2Y3^2 - EY12*EY22*EY32)

c =

(EXY3*EY1Y2^2 - EXY3*EY12*EY22 + EXY1*EY22*EY1Y3 + EXY2*EY12*EY2Y3 - EXY2*EY1Y2*EY1Y3 - EXY1*EY1Y2*EY2Y3)/(EY32*EY1Y2^2 - 2*EY1Y2*EY1Y3*EY2Y3 + EY22*EY1Y3^2 + EY12*EY2Y3^2 - EY12*EY22*EY32)
$\hat{X}$ = ((EXY1*EY2Y3^2 - EXY1*EY22*EY32 + EXY3*EY22*EY1Y3 + EXY2*EY32*EY1Y2 - EXY2*EY1Y3*EY2Y3 - EXY3*EY1Y2*EY2Y3)/(EY32*EY1Y2^2 - 2*EY1Y2*EY1Y3*EY2Y3 + EY22*EY1Y3^2 + EY12*EY2Y3^2 - EY12*EY22*EY32)$Y_{1)}$ + ((EXY2*EY1Y3^2 - EXY2*EY12*EY32 + EXY3*EY12*EY2Y3 + EXY1*EY32*EY1Y2 - EXY3*EY1Y2*EY1Y3 - EXY1*EY1Y3*EY2Y3)/(EY32*EY1Y2^2 - 2*EY1Y2*EY1Y3*EY2Y3 + EY22*EY1Y3^2 + EY12*EY2Y3^2 - EY12*EY22*EY32))Y_2 + ((EXY3*EY1Y2^2 - EXY3*EY12*EY22 + EXY1*EY22*EY1Y3 + EXY2*EY12*EY2Y3 - EXY2*EY1Y2*EY1Y3 - EXY1*EY1Y2*EY2Y3)/(EY32*EY1Y2^2 - 2*EY1Y2*EY1Y3*EY2Y3 + EY22*EY1Y3^2 + EY12*EY2Y3^2 - EY12*EY22*EY32))Y_3.

c. *Compression in MATLAB*

Recall that DPCM involves comparing sampled values to the preceding values. For this reason, it is convenient to row with single rows. After determining the size of the video input signal, the data array must be altered to have all pixel in one row. This can be done using the following for intra frame compression,

```
for count = 1:300
 X(:,:,count) =
reshape(p2(:,:,count),[1,(m*n)]);
 end
```

To find the pixel values in the next sample, X must be shifted to the right by 1. This will form the array Y:

```
count = 1;
for count1 = 1:300
for count = 1:((m*n)-1)
    Y(1,count,count1) = X(1,count+1,count1);
end
end
```

In order to assure that no information is lost the following can be implemented,

```
for count1 = 1:300
  Y(1,101376,count1) = X(1,1,count1);
end
```

DPCM is to be done for 3, 4 and 5 bits. They are all done in the same way with the exception of the quantization part. This example will be used to demonstrate 3-bit.

Calculating the error signal for each frame can be found by multiplying the X and Y arrays,

```
XY = X.*Y;
```

To find the mean and standard deviation of X and Y, the following can be implemented

```
for count = 1:300
 Ex(1,1,count) = mean(X(:,:,count));
 Ey(1,1,count) = mean(Y(:,:,count));
 Exy(1,1,count) = mean(XY(:,:,count));
 sigx(1,1,count) = std(X(:,:,count));
 sigy(1,1,count) = std(Y(:,:,count));
 end
 for count1 = 1:300
  rho(1,1,count1) = (Exy(1,1,count1) -
(Ex(1,1,count1) *
Ey(1,1,count1)))/(sigx(1,1,count1) *
sigy(1,1,count1));
end
for count = 1:300
    Xhat(1,:,count) =
((rho(1,1,count)*(Y(1,:,count)-
Ey(1,1,count))*sigx(1,1,count))/sigy(1,1,count
)) + Ex(1,1,count);
end
```

Recall that the error can be found by subtracting X from the predicated value $\hat{X}$ as follows

```
ers2 = X - Xhat;
```

Quantization can now be done for 3 bits in the same way that was demonstrated in the PCM section.

The SNR can then be found by using the following,

```
Xq = double(Xq);

  for count = 1:300
      magI(1,count) = norm(X(:,:,count));
      magerr(1,count) = norm(X(:,:,count)-
Xq(:,:,count));
  end

    SNR3 = 20*log10 (norm(magI)/norm(magerr));
```

The signal must be reshaped to its original size in order to plot the histogram. See histogram in Results section.

The process for interframe compression is almost identical to that of intraframe with one exception. This difference lies in the way in the way the data array is initialized. For interframe, implement the following,

```
count = 1;
for count1 = 1:300-1
for count = 1:((m*n))
    Y(:,count,count1) = X(:,count,count1+1);
end
end
for count1 = 1:((m*n))
  Y(1,count,300) = X(1,count,1);
end
```

The remaining parts are identical to DPCM for intraframe and can be preformed for 3, 4, and 5 bits to be used for comparison.

iii. DCT based compression

Recall that DCT entails dividing each frame into 8 x 8 pixels blocks. To implement this the following can be done,

```
I = readData('output.avi');
I = double(I);
[m,n] = size(I(:,:,1));
DCT3 = zeros([288,352,300]);
I3 = zeros([288,352,300]);

for count = 1:300
T = dctmtx(8); %returns an 8 x 8 DCT matrix
dct = @(block_struct) T * block_struct.data *
T';
DCT3(:,:,count) = blockproc(I(:,:,count),[8
8],dct);
end
```

This will return an 8 x 8 DCT matrix. As previously mentioned, the next step for DCT is to multiply the 8 x 8 matrix by *C(j)C(k)/4*. This can be implemented in MATLAB by doing the following:

```
for count1 = 1:8
    for count2 = 1:8
 tt(count1,count2) = C(count1)*C(count2)/4;
    end
end
```

The next step is to normalize the matrix:

```
for count1 = 1:7:281
    for count2 = 1:7:345
        norm (count1:count1+7,count2:count2+7)
= tt;
    end
end
norm(:,352) = norm(:,351);

for count = 1:300
DCTnorm(:,:,count) = DCT3(:,:,count).*norm;
End
```

For quantization, a function is used:

```
%function [q_out,Delta,SQNR] =
PCMfun(sig_in,L)
function [q_out] = PCMfun(sig_in,L)
%L - Quantization levels
%sig_in - Input signals
%q_out - quantized outputs
%delta - quantization intervals
%SQNR - sognal to noise ratio
sig_pmax = max(max(sig_in));
sig_nmax = min(min(sig_in));
Delta = (sig_pmax - sig_nmax)/L;
q_level = sig_nmax + Delta/2:Delta:sig_pmax -
Delta/2;
L_sig = length(sig_in);
sigp = (sig_in - sig_nmax)/Delta+1/2;
qindex = round(sigp);
qindex = min(qindex,L);
q_out = q_level(qindex);
SQNR = 20*log10(norm(sig_in)/norm(sig_in -
q_out));

end
```

The function is called using a for loop:

```
for count = 1:300
     DCTnormq(:,:,count) =
PCMfun(DCTnorm(:,:,count),8);
end
```

DCT is now complete. To reverse this, the following can be done:

```
%%%%denormalization%%%
for count1 = 1:8
    for count2 = 1:8
 ttt(count1,count2) = 4/(C(count1)*C(count2));
    end
end

for count1 = 1:7:281
    for count2 = 1:7:345
        denorm
(count1:count1+7,count2:count2+7) = ttt;
    end
end
denorm(:,352) = denorm(:,351);

for count = 1:300
DCTdenorm(:,:,count) =
DCTnorm(:,:,count).*denorm;
end
%%%%inverse DCT%%%%%
count2 = 1
for count = 1:300
invdct = @(block_struct) T' *
block_struct.data * T;
I3(:,:,count) =
blockproc(DCTdenorm(:,:,count),[8 8],invdct);
end
```

iv. Bonus: Non-uniform quantizer

We will implement the two compression laws that have been accepted as desirable standards by the ITU-IT, the µ-law used in North America and Japan, and the A –law used in Europe and the rest of the world and on international routes [2]. We will compress according to the equations for the laws and calculate the signal to noise ratios.

For this purpose, it is also convenient to contain all the pixel values in one row:

```
for count = 1:300
 X(:,:,count) =
reshape(p2(:,:,count),[1,(m*n)]);
 end
 mp  = max(abs(X));
```

To apply the A-law, the following can be done:

```
A = 87.6;
B = 1 + log(A);
y1 = zeros([1,m*n,300]);
  for count = 1:300
      for count2 = 1:m*n
    if X(:,count2,count)/mp(:,:,count) >=0 &&
X(:,count2,count)/mp(:,:,count) <= 1/A
       y1(:,count2,count) =
(A/B)*(X(:,count2,count)/mp(:,:,count));
    elseif X(:,count2,count)/mp(:,:,count)
>=1/A && X(:,count2,count)/mp(:,:,count) <= 1
      y1(:,count2,count) =
      1/B*(1+log(A*X(:,count2,count)/mp(:,:,co
      unt)));
    end
      end
  end
```

To apply the µ-law the following can be done:

```
u =255;
C = 1/(log(1+u));
y2 = zeros([1,m*n,300]);
   for count = 1:300
   for count2 = 1:m*n
       y2(:,count2,count)  =  C*log(1  +
((u*X(:,count2,count))/mp(:,:,count)));
   end
   end
```

IV. Results

A. PCM

Using knowledge from pulse code modulation the original video was represented with differing quantization levels. The signal to noise ratio or each level performed is shown in table 1 below.

TABLE 1 PCM SIGNAL TO NOISE RATIO

PCM Quantization Levels	**Signal to Noise Ratio**
16	44.0990
8	34.9554
4	24.1286
2	20.2587

As expected, the signal to noise ratio decreases as the video is represented with fewer quantization levels. This is also evident in the quality of the video. Figure 8 and 9 shows the outputs of the video represented by 16 quantization levels and by 2 quantization levels.

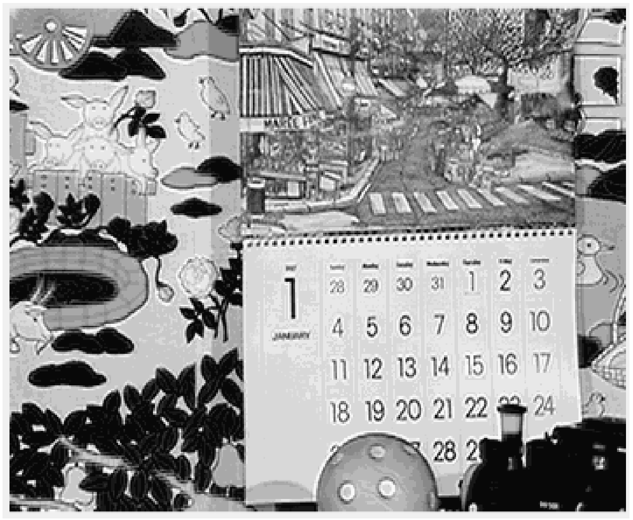


Figure 27. 4 bit PCM output video

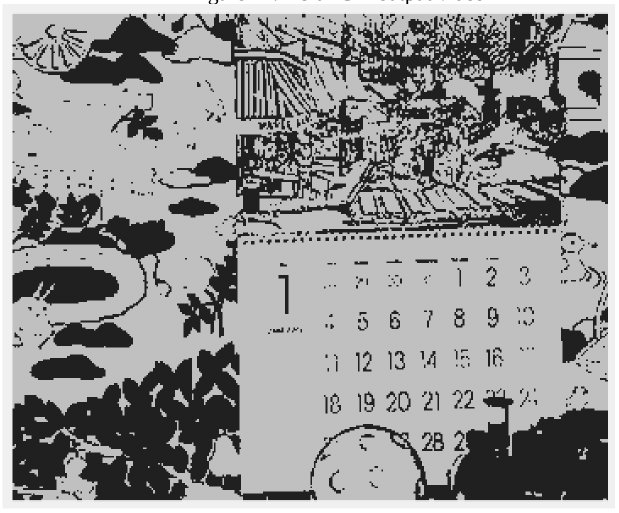


Figure 28. 1 bit PCM output video

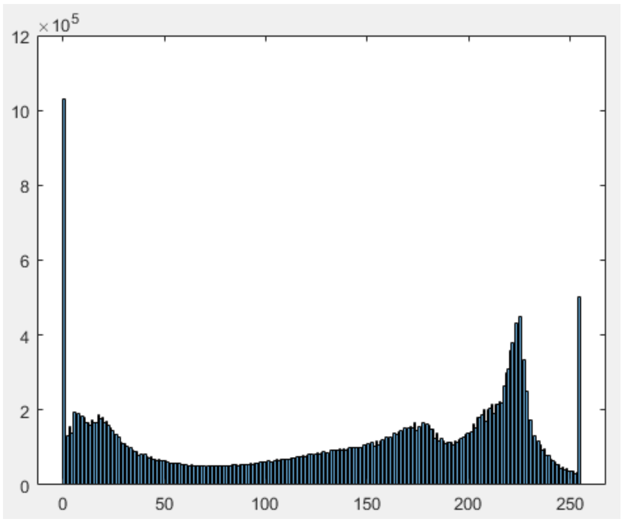


Figure 29. histogram of original video

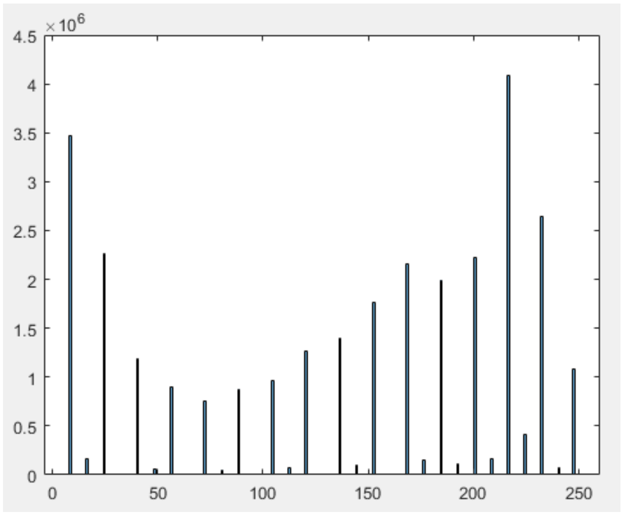


Figure 30. Histogram of 16 bit quantized video

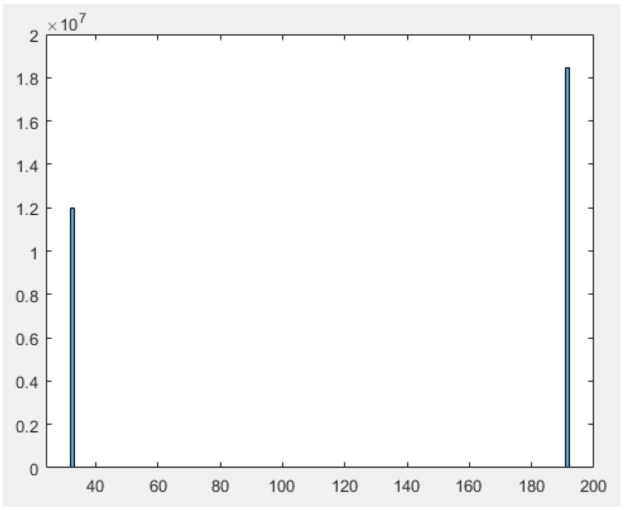


Figure 31. Histogram of 1 bit quantized video

From the histogram and screenshots of the video PCM restricts the pixel values to discrete values based on the number of quantization levels. In the 1 bit quantization, the pixel values for each frame is only allowed to be one of two values and as a result of this a lot information from the video is lost. So by itself, Pulse code modulation is not a good way to compress video because of the heavy loss of information.

B. DPCM

The signal to noise ratios were calculated for each version of DPCM performed as;
SNR = Average power of signal/Average power of noise
= 20*$\log_{10}$ (magnitude of original video/magnitude of original video – DPCM output)

TABLE 2 DPCM SIGNAL TO NOISE RATIOS USING LINEAR PREDICTORS

DPCM variation	Signal to Noise Ratio	Compression Ratio
Intra frame DPCM with 3-bit quantization	27.7157	8 - 3
Intra frame DPCM with 4-bit quantization	35.0593	8 - 4
Intra frame DPCM with 5-bit quantization	40.4939	8 - 5
Inter frame DPCM with 3-bit quantization	26.6360	8 – 3
Inter frame DPCM with 4-bit quantization	34.4122	8 – 4
Inter frame DPCM with 5-bit quantization	40.9841	8 - 5

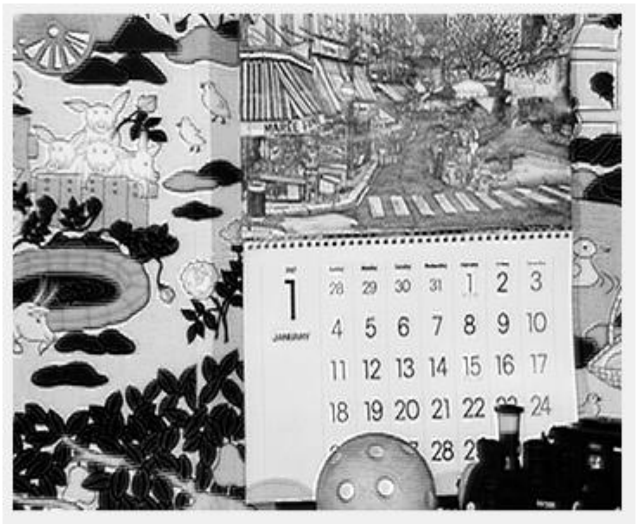


Figure 32. Video after Intra frame DPCM with 3-bit quantization

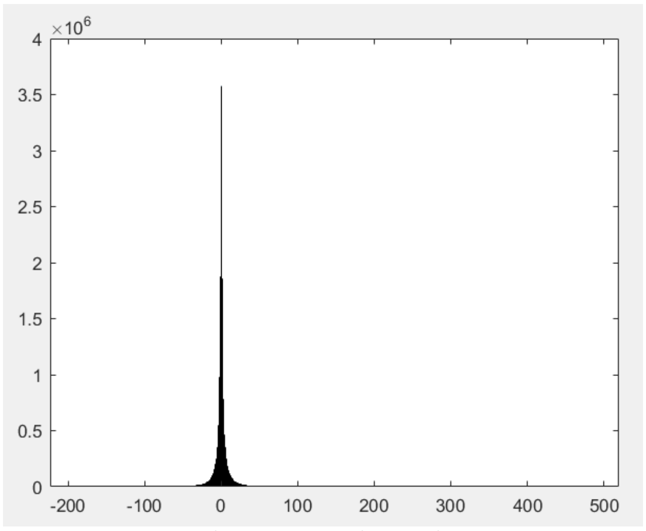


Figure 33: Histogram of the error signal for Intra frame DPCM with 3-bit quantization

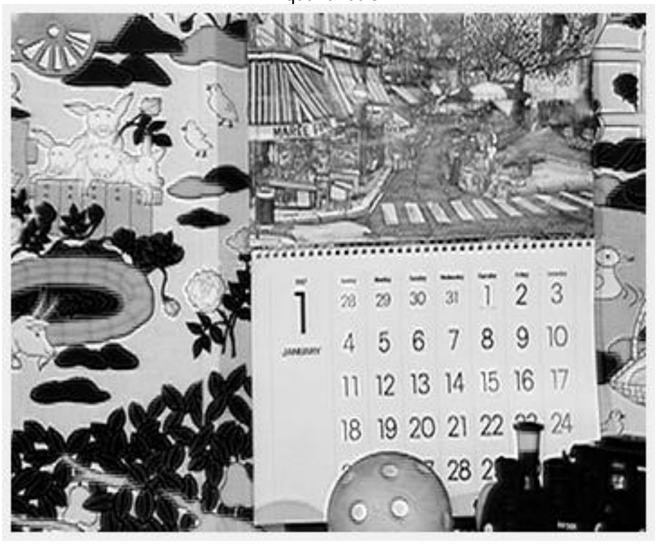


Figure 34. Video after Intra frame DPCM with 4-bit quantization

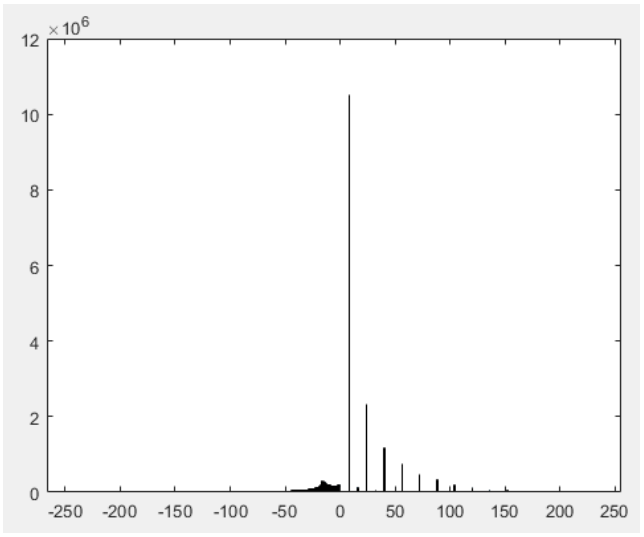


Figure 35: Histogram of the error signal for Intra frame DPCM with 4-bit quantization

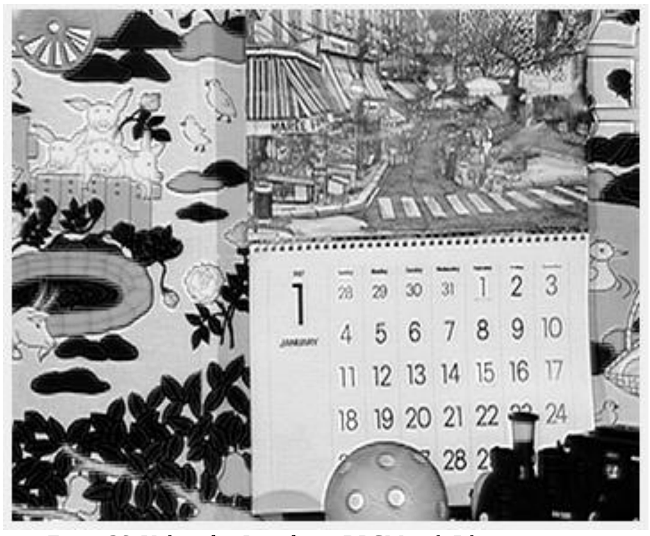


Figure 36. Video after Intra frame DPCM with 5-bit quantization

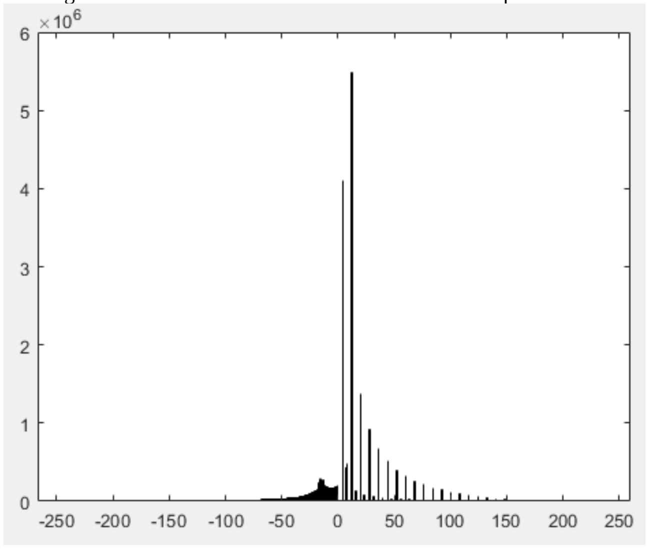


Figure 37: Histogram of the error signal for Intra frame DPCM with 5-bit quantization

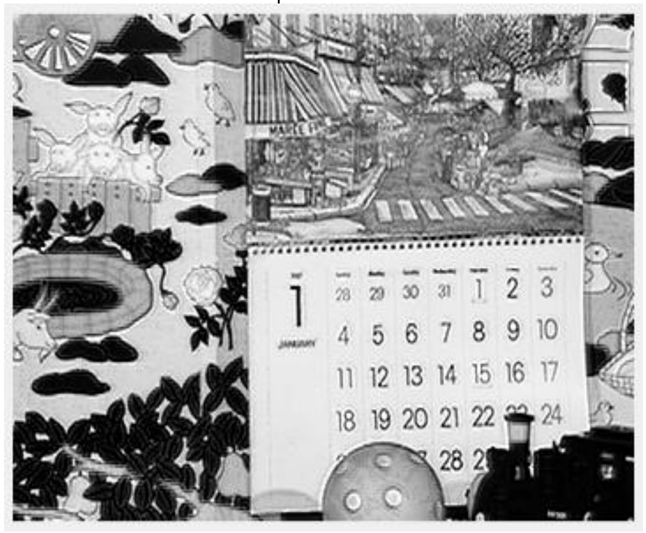


Figure 38. Video after Inter frame DPCM with 3-bit quantization

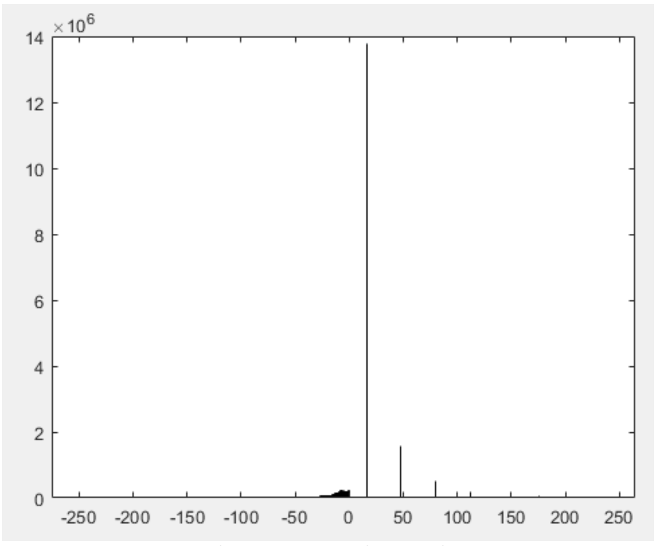


Figure 39: Histogram of the error signal for Inter frame DPCM with 3-bit quantization

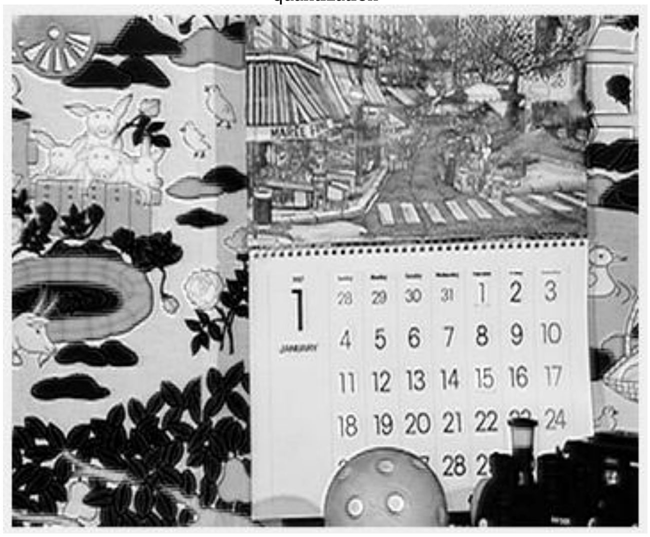


Figure 40. Video after Inter frame DPCM with 4-bit quantization

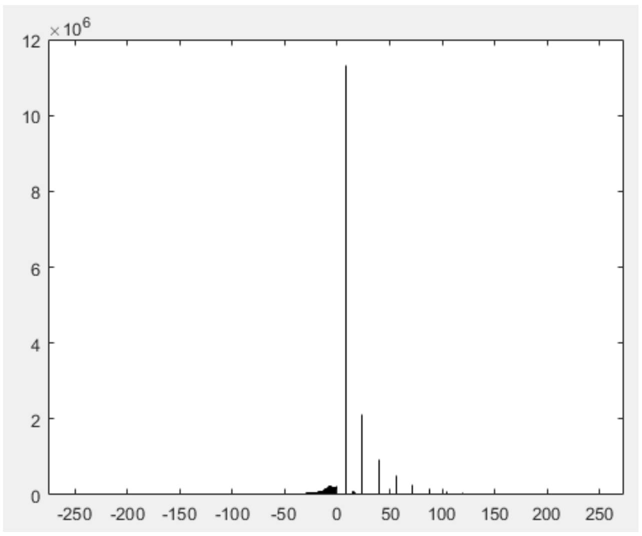


Figure 41: Histogram of the error signal for Inter frame DPCM with 4-bit quantization

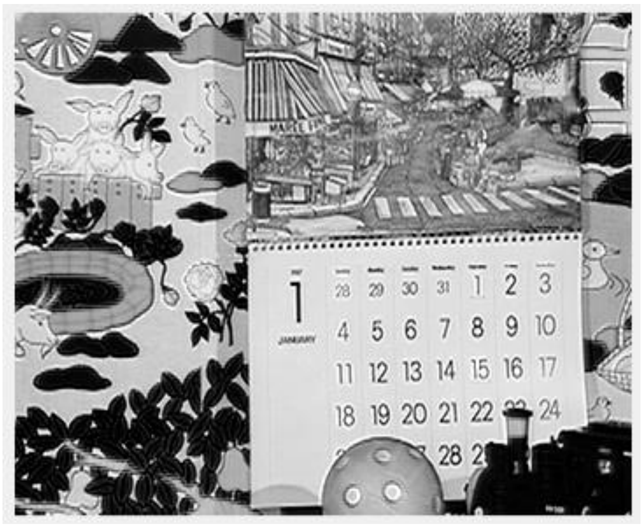


Figure 42: Video after Inter frame DPCM with 5-bit quantization

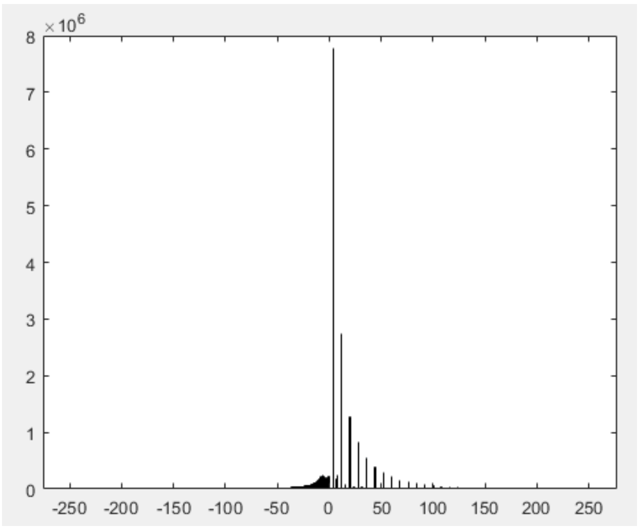


Figure 43: Histogram of the error signal for Inter frame DPCM with 5-bit quantization

The histograms for the error signals from the DPCM compressions are appear similar as the goal of the compression is to minimize the variance of this compression. DPCM using a 3rd order predictor is used to lower the variance even more.

C. DPCM with a 3rd order Predictor

TABLE 3 SIGNAL TO NOISE RATIOS OF DPCM WITH 3RD ORDER PREDICTOR

Quantization	Signal to noise ratio	Compression Ratio
3-Bit	26.5869	8 – 3
4-Bit	33.5967	8 – 4
5-Bit	40.4321	8 - 5

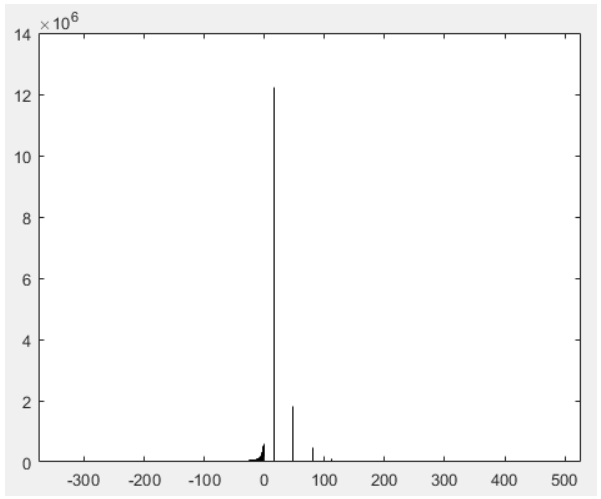


Figure 44. Histogram of the error signal for DPCM with a 3rd order predictor and 3-bit quantization

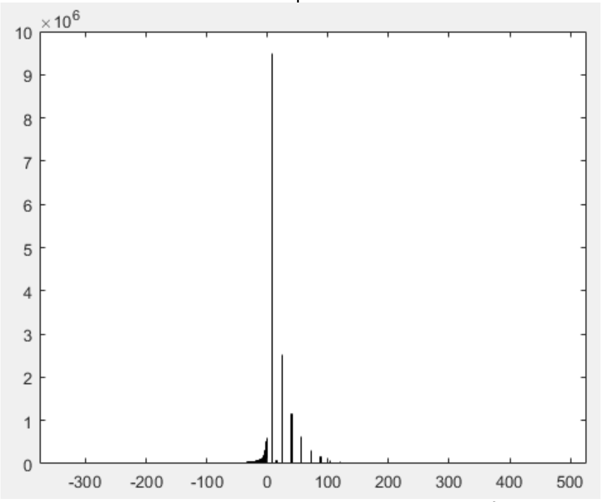


Figure 45. Histogram of the error signal for DPCM with a 3rd order predictor and 4-bit quantization

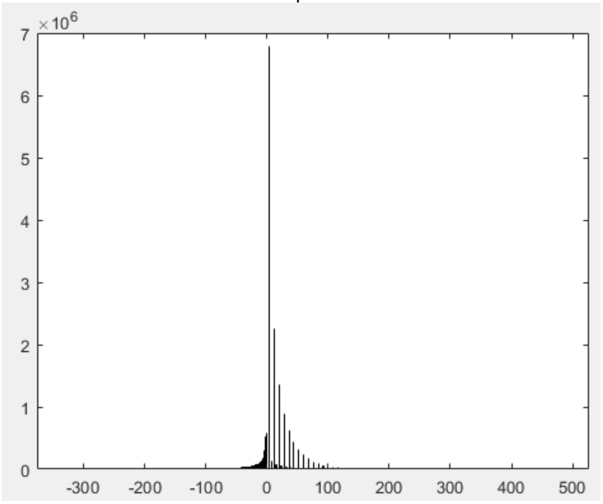


Figure 46. Histogram of the error signal for DPCM with a 3rd order predictor and 5-bit quantization

The goal of having a 3rd order predictor is to reduce the variance of the error signal. From our results we observe that the difference in the signal to noise ratio with this predictor and the inter frame linear predictor is not very large.

D. Discrete Cosine Transform

We implemented the discrete cosine transform on each 8 x 8 piece using the DCT function in MATLAB. After applying the transform and normalization, the original video was obtained by denormalization and inverse DCT. DCT is a good compression method because it requires the transmission of fewer number of bits.

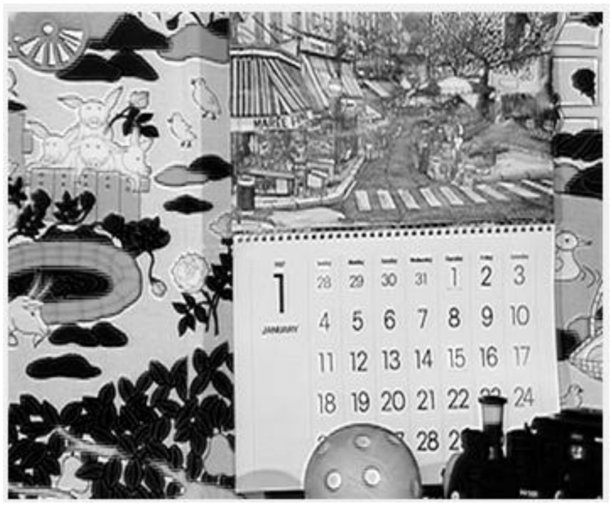


Figure 47. Video after DCT

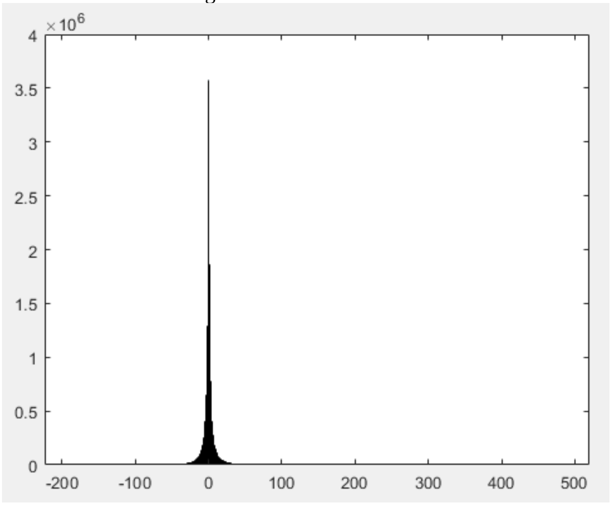


Figure 48. Histogram of video after undergoing Discrete Cosine Transform and normalization

As a side note: we noticed an interesting effect when we performed the inverse DCT on the normalized signal without de-normalizing.

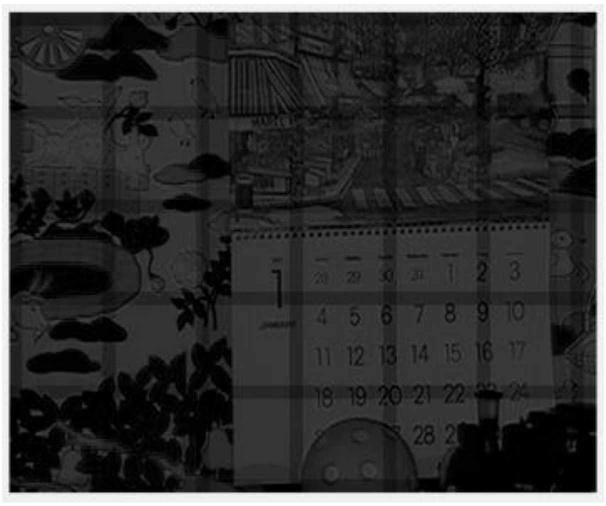


Figure 49. DCT compression, normalization and decompression

To provide an additional 2 – 1 compression we tried applying DPCM on the DCT matrix. So instead of transmitting the 8 x 8 blocks as they were, we will transmit the difference between adjacent 8 x 8 blocks. And by going through the process and creating a linear predictor, we were able to generate the error signal. The histogram is presented in the figure below.

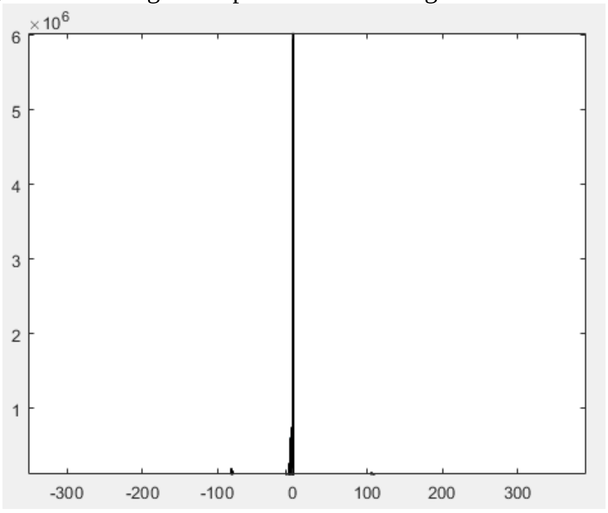


Figure 50. Histogram of error signal of DPCM on a DCT normalized matrix

E. Non-Uniform Quantization

The purpose of non-uniform quantization is to use smaller steps for smaller amplitudes and larger for larger amplitudes. It is intended to be used in speech where larger amplitudes are less frequent than smaller amplitudes. The table below shows the result of calculating the signal to noise ratio's of the compressed videos. The signal is a compressed version of the original video and it is expected to have very low SNR.
The result from the non-uniform quantization is more evident in the histograms of the signal.

TABLE 4 SIGNAL TO NOISE RATIO AFTER NON-UNIFORM QUANTIZATION USING A-LAW AND μ-LAW WITH A = 87.9, μ = 255

Non-Uniform Quantization (Compression)	Signal to Noise Ratio
A-Law	0.0436
μ- Law	0.0437

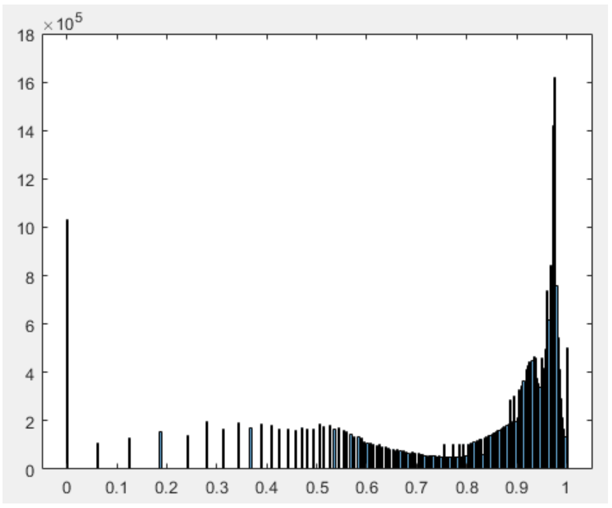


Figure 51. Non-Uniform compressed signal using the A-law histogram

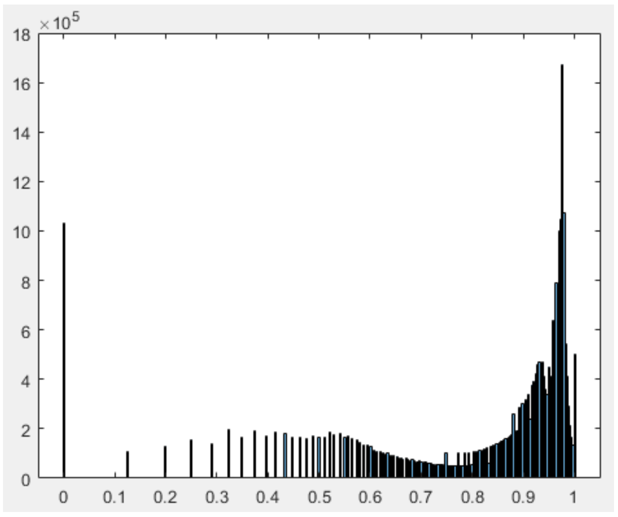


Figure 52. Non-Uniform compressed signal using the μ-law histogram

V. Discussion

Bandwidth requirements are perhaps the biggest that was faced as video and television began being transmitted digitally. Transmitting video digitally has many advantages over analog transmission;

- Digital signals can withstand noise and distortion much better than analog signals
- Digital signals can be transmitted over long distances with high fidelity with the use of regenerative repeaters employed set locations along the path the signal travels. The repeaters detect pulses and then transmits new clean distortion free pulses to the next station, until it reaches its destination
- Digital hardware implementation is flexible

To convert an analog signal to a digital signal, the signal will have to be sampled, then quantized. Sampling is the process of reducing an analog continuous signal to a digital signal and quantizing is restricting the amplitudes of the sampled signal to set discrete levels within a range. Because if the high bandwidth requirements, use of direct sampling and quantization leads to signals that require lots of bandwidth

The purpose of this project was to implement video compression schemes and analyze the results from the quality of the video and signal to noise ratios. The schemes that we implemented were pulse code modulation, Differential Pulse code modulation, Discrete Cosine transform, and combination of these techniques along with a study of non-uniform quantization.

In pulse code modulation we quantized a given video signal with various quantization levels to calculate and compare signal to noise ratios and video quality. And as expected, the more quantization levels, the better the quality of the video and the higher the signal to noise ratios.

In the differential pulse code modulation (DPCM) section, we implemented DPCM with three different predictors, a linear predictor for inter frame, an intra frame prediction and a 3rd order predictor. For Intra frame prediction, we calculated and minimized the variance of the difference between a pixel and the pixel to the left of it. For Inter frame we minimized the difference between a pixel and the pixel in the same position in the previous frame. For the 3rd order predictor we minimized the difference between frames as listed previously and the difference between a pixel and the pixel right below it.

Signal to Noise Ratio and DPCM

The purpose of DPCM compression is to reduce the amount of information to be transmitted for high fidelity image transfer from a sender to a receiver. From table 2 and 3, the signal to noise ratio does not change a lot with the DPCM technique, this is because the signal to noise ratio is just a function of the quantization. If the DPCM error signals were quantized using non-uniform quantizers, we expect to see a larger variance in the signal to noise ratios. The compression reveals itself in the histogram, as we see there is a smaller range of values for the DPCM implemented with a 3rd order predictor compared to the first order predictors.

The Best Compression

Discrete cosine transform provides the best compression, and when combined with DPCM can provide additional compression. After normalization and quantization of a DCT matrix, we find that most of the values become 0. In practice these do not have to be sent at all, and can be assumed to be 0 at the receiver's end. Discrete cosine transform based compression can produce up to 32 – 1 compression.

Comparison of Compression Methods

Method of Compression	Rating: 1 lowest – 4 highest	Reasoning
PCM	1	Lot of information is lost at low levels of quantization
Non-uniform Quantization	2	Better than uniform-quantization because it reduces SNR
DPCM	3	Lowers error by predicting sampled values, quantization levels are not significant
DCT	4	Reduces a significant amount of redundancy while maintaining high quality

References

[1] S. M. Metev and V. P. Veiko, *Laser Assisted Microtechnology*, 2nd ed., R. M. Osgood, Jr., Ed. Berlin, Germany: Springer-Verlag, 1998.

[2] B.P. Lathi and Z. Ding, Modern Digital and Analog Communication Systems. Oxford University Press, 2010.

APPENDIX

PCM Code:

```
p = readData('output.avi');
q = readData('output.avi');
r = readData('output.avi');
s = readData('output.avi');
t = readData('output.avi');
[m,n] = size(p(:,:,1));
%16 quantization levels
for count1 = 1:300
    for count2 = 1:352
        for count3 = 1:288
            if q(count3,count2,count1)>= 0 && q(count3,count2,count1)<= 15
                q(count3,count2,count1) = 8;
                            elseif q(count3,count2,count1)>= 17 && q(count3,count2,count1)<= 32
                q(count3,count2,count1) = 24;
            elseif q(count3,count2,count1)>= 33 && q(count3,count2,count1)<= 47
                q(count3,count2,count1) = 40;
                            elseif q(count3,count2,count1)>= 49 && q(count3,count2,count1)<= 64
                q(count3,count2,count1) = 56;
            elseif q(count3,count2,count1)>= 65 && q(count3,count2,count1)<= 79
                q(count3,count2,count1) = 72;
                            elseif q(count3,count2,count1)>= 81 && q(count3,count2,count1)<= 96
                q(count3,count2,count1) = 88;
            elseif q(count3,count2,count1)>= 97 && q(count3,count2,count1)<= 111
                q(count3,count2,count1) = 104;
                            elseif q(count3,count2,count1)>= 113 && q(count3,count2,count1)<= 128
                q(count3,count2,count1) = 120;
            elseif q(count3,count2,count1)>= 129 && q(count3,count2,count1)<= 143
                q(count3,count2,count1) = 136;
                            elseif q(count3,count2,count1)>= 145 && q(count3,count2,count1)<= 160
                q(count3,count2,count1) = 152;
            elseif q(count3,count2,count1)>= 161 && q(count3,count2,count1)<= 175
                q(count3,count2,count1) = 168;
                         elseif q(count3,count2,count1)>= 177 && q(count3,count2,count1)<= 191
                q(count3,count2,count1) = 184;
                            elseif q(count3,count2,count1)>= 193 && q(count3,count2,count1)<= 207
                q(count3,count2,count1) = 200;
            elseif q(count3,count2,count1)>= 209 && q(count3,count2,count1)<= 223
                q(count3,count2,count1) = 216;
                            elseif q(count3,count2,count1)>= 225 && q(count3,count2,count1)<= 239
                q(count3,count2,count1) = 232;
            elseif q(count3,count2,count1)>= 241 && q(count3,count2,count1)<= 255
                q(count3,count2,count1) = 248;

            end
        end
    end
end

%8 quantization levels
for count1 = 1:300
    for count2 = 1:352
        for count3 = 1:288
            if r(count3,count2,count1)>= 0 && r(count3,count2,count1)<= 31
                r(count3,count2,count1) = 16;
                 elseif r(count3,count2,count1)>= 32 && r(count3,count2,count1)<= 63
                r(count3,count2,count1) = 48;
                elseif r(count3,count2,count1)>= 64 && r(count3,count2,count1)<= 95
                r(count3,count2,count1) = 80;
                elseif r(count3,count2,count1)>= 96 && r(count3,count2,count1)<= 127
                r(count3,count2,count1) = 112;
                elseif r(count3,count2,count1)>= 128 && r(count3,count2,count1)<= 159
                r(count3,count2,count1) = 144;
                elseif r(count3,count2,count1)>= 160 && r(count3,count2,count1)<= 191
                r(count3,count2,count1) = 176;
                elseif r(count3,count2,count1)>= 192 && r(count3,count2,count1)<= 223
                r(count3,count2,count1) = 208;
                elseif r(count3,count2,count1)>= 224 && r(count3,count2,count1)<= 255
                r(count3,count2,count1) = 240;

            end
        end
    end
end

%4 quantization levels
for count1 = 1:300
    for count2 = 1:352
        for count3 = 1:288
            if s(count3,count2,count1)>= 0 && s(count3,count2,count1)<= 63
```

```
                s(count3,count2,count1) = 32;
                 elseif s(count3,count2,count1)>= 64 && s(count3,count2,count1)<= 127
                s(count3,count2,count1) = 128;
                 elseif s(count3,count2,count1)>= 128 && s(count3,count2,count1)<= 191
                s(count3,count2,count1) = 160;
                 elseif s(count3,count2,count1)>= 192 && s(count3,count2,count1)<= 255
                s(count3,count2,count1) = 224;
            end
        end
    end
end


%2 quantization levels
for count1 = 1:300
    for count2 = 1:352
        for count3 = 1:288
            if t(count3,count2,count1)>= 0 && t(count3,count2,count1)<= 127
                t(count3,count2,count1) = 32;
                 elseif t(count3,count2,count1)>= 128 && t(count3,count2,count1)<= 255
                t(count3,count2,count1) = 192;
            end
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%SNR%%%%%%%%%%%%%%
p = double(p);
q = double(q);
r = double(r);
s = double(s);
t = double(t);


for count = 1:300
    magp(1,count) = norm(p(:,:,count));
    mag4(1,count) = norm(p(:,:,count)-q(:,:,count));
    mag3(1,count) = norm(p(:,:,count)-r(:,:,count));
    mag2(1,count) = norm(p(:,:,count)-s(:,:,count));
    mag1(1,count) = norm(p(:,:,count)-t(:,:,count));
end

    SNRpcm4 = 20*log10 (norm(magp)/norm(mag4));
    SNRpcm3 = 20*log10 (norm(magp)/norm(mag3));
    SNRpcm2 = 20*log10 (norm(magp)/norm(mag2));
    SNRpcm1 = 20*log10 (norm(magp)/norm(mag1));

q = uint8(q);
t = uint8(t);
```

DPCM 3-bit Code:

```
p2 = readData('output.avi');  %load the video and converts to double
 p2 = double(p2);
 [m,n] = size(p2(:,:,1));
%  % I want to have all the pixel values in one row
 for count = 1:300
 X(:,:,count) = reshape(p2(:,:,count),[1,(m*n)]);
 end
%The for loop is to store the next values of X in Y
count = 1;
for count1 = 1:300
for count = 1:((m*n)-1)
    Y(1,count,count1) = X(1,count+1,count1);
end
end
for count1 = 1:300
  Y(1,101376,count1) = X(1,1,count1);    %I did this so i dont lose the information in X(1)
end

%calculating the error signal for each frame
 XY = X.*Y;     %Multiplying the X and Y arrays together

 for count = 1:300
 Ex(1,1,count) = mean(X(:,:,count));
 Ey(1,1,count) = mean(Y(:,:,count));
 Exy(1,1,count) = mean(XY(:,:,count));
 sigx(1,1,count) = std(X(:,:,count)); %calculating the standard deviations
 sigy(1,1,count) = std(Y(:,:,count));
 end
 for count1 = 1:300
  rho(1,1,count1) = (Exy(1,1,count1) - (Ex(1,1,count1) * Ey(1,1,count1)))/(sigx(1,1,count1) * sigy(1,1,count1));
end
for count = 1:300
    Xhat(1,:,count) = ((rho(1,1,count)*(Y(1,:,count)-Ey(1,1,count))*sigx(1,1,count))/sigy(1,1,count)) + Ex(1,1,count);
end

 ers2 = X - Xhat; %error signal
    %%%Quantization%%%
 count3 = 1;
 for count1 = 1:300
    for count2 = 1:101376
            if ers2(count3,count2,count1)>= 0 && ers2(count3,count2,count1)<= 31
                ers2(count3,count2,count1) = 16;
                 elseif ers2(count3,count2,count1)>= 32 && ers2(count3,count2,count1)<= 63
                ers2(count3,count2,count1) = 48;
                elseif ers2(count3,count2,count1)>= 64 && ers2(count3,count2,count1)<= 95
                ers2(count3,count2,count1) = 80;
                elseif ers2(count3,count2,count1)>= 96 && ers2(count3,count2,count1)<= 127
                ers2(count3,count2,count1) = 112;
                elseif ers2(count3,count2,count1)>= 128 && ers2(count3,count2,count1)<= 159
                ers2(count3,count2,count1) = 144;
                elseif ers2(count3,count2,count1)>= 160 && ers2(count3,count2,count1)<= 191
                ers2(count3,count2,count1) = 176;
                elseif ers2(count3,count2,count1)>= 192 && ers2(count3,count2,count1)<= 223
                ers2(count3,count2,count1) = 208;
                elseif ers2(count3,count2,count1)>= 224 && ers2(count3,count2,count1)<= 255
                ers2(count3,count2,count1) = 240;
            elseif ers2(count3,count2,count1)>= 0 && ers2(count3,count2,count1)<= -31
                ers2(count3,count2,count1) = -16;
                 elseif ers2(count3,count2,count1)>= -32 && ers2(count3,count2,count1)<= -63
                ers2(count3,count2,count1) = -48;
                elseif ers2(count3,count2,count1)>= -64 && ers2(count3,count2,count1)<= -95
                ers2(count3,count2,count1) = -80;
                elseif ers2(count3,count2,count1)>= -96 && ers2(count3,count2,count1)<= -127
                ers2(count3,count2,count1) = -112;
                elseif ers2(count3,count2,count1)>= -128 && ers2(count3,count2,count1)<= -159
                ers2(count3,count2,count1) = -144;
                elseif ers2(count3,count2,count1)>= -160 && ers2(count3,count2,count1)<= -191
                ers2(count3,count2,count1) = -176;
                elseif ers2(count3,count2,count1)>= -192 && ers2(count3,count2,count1)<= -223
                ers2(count3,count2,count1) = -208;
                elseif ers2(count3,count2,count1)>= -224 && ers2(count3,count2,count1)<= -255
                ers2(count3,count2,count1) = -240;
            end
    end
 end
  %reciever side
 Xq = ers2 + Xhat;
 %%%%%%%%%%%%%%%%%%%%%%SNR%%%%%%%%%%%%%%

Xq = double(Xq);
```

```
  for count = 1:300
      magI(1,count) = norm(X(:,:,count));
      magerr(1,count) = norm(X(:,:,count)- Xq(:,:,count));
  end

    SNR3 = 20*log10 (norm(magI)/norm(magerr));

  for count = 1:300
 X2play3(:,:,count) = reshape(Xq(:,:,count),[m,n]);
  end
  X2play3 = uint8(X2play3);
 %implay(X2play3)
%%Plotting error signal histogram
histogram(ers2)
p2 = uint8(p2);
title('Error histogram: DPCM - 3 bits')
%implay(p2)
```

DPCM – 4-bit Code:

```
p2 = readData('output.avi');  %load the video and converts to double
p2 = double(p2);
 [m,n] = size(p2(:,:,1));
%  % I want to have all the pixel values in one row
 for count = 1:300
 X(:,:,count) = reshape(p2(:,:,count),[1,(m*n)]);
 end
%The for loop is to store the next values of X in Y
count = 1;
for count1 = 1:300
for count = 1:((m*n)-1)
    Y(1,count,count1) = X(1,count+1,count1);
end
end
for count1 = 1:300
  Y(1,101376,count1) = X(1,1,count1);    %I did this so i dont lose the information in X(1)
end

%calculating the error signal for each frame
 XY = X.*Y;     %Multiplying the X and Y arrays together

 for count = 1:300
 Ex(1,1,count) = mean(X(:,:,count));
 Ey(1,1,count) = mean(Y(:,:,count));
 Exy(1,1,count) = mean(XY(:,:,count));
 sigx(1,1,count) = std(X(:,:,count)); %calculating the standard deviations
 sigy(1,1,count) = std(Y(:,:,count));
 end
 for count1 = 1:300
  rho(1,1,count1) = (Exy(1,1,count1) - (Ex(1,1,count1) * Ey(1,1,count1)))/(sigx(1,1,count1) * sigy(1,1,count1));
 end
Xhat = zeros([1,m*n,300]);
for count = 1:300
    Xhat(1,:,count) = ((rho(1,1,count)*(Y(1,:,count)-Ey(1,1,count))*sigx(1,1,count))/sigy(1,1,count)) + Ex(1,1,count);
end
% for count = 1:300
%     Xhat(1,:,count) = ((rho(1,1,count)*sigx(1,1,count)/sigy(1,1,count))*Y(1,:,count)) + Ex(1,1,count) -
((rho(1,1,count)*sigx(1,1,count))/sigy(1,1,count));
% end
 ers2 = X - Xhat; %error signal
 %%%%%%%%%%Quantization%%%%%%%%%%
 count3 = 1;
 for count1 = 1:300
    for count2 = 1:101376
            if ers2(count3,count2,count1)>= 0 && ers2(count3,count2,count1)<= 15
                ers2(count3,count2,count1) = 8;
                        elseif ers2(count3,count2,count1)>= 17 && ers2(count3,count2,count1)<= 32
                ers2(count3,count2,count1) = 24;
            elseif ers2(count3,count2,count1)>= 33 && ers2(count3,count2,count1)<= 47
                ers2(count3,count2,count1) = 40;
                        elseif ers2(count3,count2,count1)>= 49 && ers2(count3,count2,count1)<= 64
                ers2(count3,count2,count1) = 56;
            elseif ers2(count3,count2,count1)>= 65 && ers2(count3,count2,count1)<= 79
                ers2(count3,count2,count1) = 72;
                        elseif ers2(count3,count2,count1)>= 81 && ers2(count3,count2,count1)<= 96
                ers2(count3,count2,count1) = 88;
            elseif ers2(count3,count2,count1)>= 97 && ers2(count3,count2,count1)<= 111
                ers2(count3,count2,count1) = 104;
                        elseif ers2(count3,count2,count1)>= 113 && ers2(count3,count2,count1)<= 128
                ers2(count3,count2,count1) = 120;
            elseif ers2(count3,count2,count1)>= 129 && ers2(count3,count2,count1)<= 143
                ers2(count3,count2,count1) = 136;
                        elseif ers2(count3,count2,count1)>= 145 && ers2(count3,count2,count1)<= 160
                ers2(count3,count2,count1) = 152;
            elseif ers2(count3,count2,count1)>= 161 && ers2(count3,count2,count1)<= 175
                ers2(count3,count2,count1) = 168;
                     elseif ers2(count3,count2,count1)>= 177 && ers2(count3,count2,count1)<= 191
                ers2(count3,count2,count1) = 184;
                        elseif ers2(count3,count2,count1)>= 193 && ers2(count3,count2,count1)<= 207
                ers2(count3,count2,count1) = 200;
            elseif ers2(count3,count2,count1)>= 209 && ers2(count3,count2,count1)<= 223
                ers2(count3,count2,count1) = 216;
                        elseif ers2(count3,count2,count1)>= 225 && ers2(count3,count2,count1)<= 239
                ers2(count3,count2,count1) = 232;
            elseif ers2(count3,count2,count1)>= 241 && ers2(count3,count2,count1)<= 255
                ers2(count3,count2,count1) = 248;
            elseif ers2(count3,count2,count1)>= 0 && ers2(count3,count2,count1)<= -15
                ers2(count3,count2,count1) = -8;
                        elseif ers2(count3,count2,count1)>= -17 && ers2(count3,count2,count1)<= -32
                ers2(count3,count2,count1) = -24;
            elseif ers2(count3,count2,count1)>= -33 && ers2(count3,count2,count1)<= -47
```

```
                ers2(count3,count2,count1) = -40;
                            elseif ers2(count3,count2,count1)>= -49 && ers2(count3,count2,count1)<= -64
                ers2(count3,count2,count1) = -56;
            elseif ers2(count3,count2,count1)>= -65 && ers2(count3,count2,count1)<= -79
                ers2(count3,count2,count1) = -72;
                            elseif ers2(count3,count2,count1)>= -81 && ers2(count3,count2,count1)<= -96
                ers2(count3,count2,count1) = -88;
            elseif ers2(count3,count2,count1)>= -97 && ers2(count3,count2,count1)<= -111
                ers2(count3,count2,count1) = -104;
                            elseif ers2(count3,count2,count1)>= -113 && ers2(count3,count2,count1)<= -128
                ers2(count3,count2,count1) = -120;
            elseif ers2(count3,count2,count1)>= -129 && ers2(count3,count2,count1)<= -143
                ers2(count3,count2,count1) = -136;
                            elseif ers2(count3,count2,count1)>= -145 && ers2(count3,count2,count1)<= -160
                ers2(count3,count2,count1) = -152;
            elseif ers2(count3,count2,count1)>= -161 && ers2(count3,count2,count1)<= -175
                ers2(count3,count2,count1) = -168;
                         elseif ers2(count3,count2,count1)>= -177 && ers2(count3,count2,count1)<= -191
                ers2(count3,count2,count1) = -184;
                            elseif ers2(count3,count2,count1)>= -193 && ers2(count3,count2,count1)<= -207
                ers2(count3,count2,count1) = -200;
            elseif ers2(count3,count2,count1)>= -209 && ers2(count3,count2,count1)<= -223
                ers2(count3,count2,count1) = -216;
                            elseif ers2(count3,count2,count1)>= -225 && ers2(count3,count2,count1)<= -239
                ers2(count3,count2,count1) = -232;
            elseif ers2(count3,count2,count1)>= -241 && ers2(count3,count2,count1)<= -255
                ers2(count3,count2,count1) = -248;

            end
    end
end
 %reciever side
 Xq = ers2 + Xhat;
 %%%%%%%%%%%%%%%%%%%%%%SNR%%%%%%%%%%%%%%
%  noise = Xq - X;
%  Px = 0;
%  Pn = 0;
%  for count = 1:300
%      for count1 = 1:m*n
%  Px = (X(1,count1,count))^2 + Px;
%  Pn = (noise(1,count1,count))^2 + Pn;
%      end
%  end
%  SNR4 = (Px/(m*n*300))/((Pn/m*n*300));
Xq = double(Xq);
  for count = 1:300
      magI(1,count) = norm(X(:,:,count));
      magerr(1,count) = norm(X(:,:,count)- Xq(:,:,count));
  end

    SNR4 = 20*log10 (norm(magI)/norm(magerr));

  for count = 1:300
 X2play4(:,:,count) = reshape(Xq(:,:,count),[m,n]);
  end
  X2play4 = uint8(X2play4);
 %implay(X2play4)
%%Plotting error signal histogram
histogram(ers2)
p2 = uint8(p2);
title('Error histogram: DPCM - 4 bits')


 %implay(X2playif5)
```

DPCM – 5-bit Code:

```
p2 = readData('output.avi');  %load the video and converts to double
p2 = double(p2);
[m,n] = size(p2(:,:,1));
%  % I want to have all the pixel values in one row
 for count = 1:300
 X(:,:,count) = reshape(p2(:,:,count),[1,(m*n)]);
 end
%The for loop is to store the next values of X in Y
count = 1;
for count1 = 1:300
for count = 1:((m*n)-1)
    Y(1,count,count1) = X(1,count+1,count1);
end
end
for count1 = 1:300
  Y(1,101376,count1) = X(1,1,count1);    %I did this so i dont lose the information in X(1)
end

%calculating the error signal for each frame
 XY = X.*Y;     %Multiplying the X and Y arrays together

 for count = 1:300
 Ex(1,1,count) = mean(X(:,:,count));
 Ey(1,1,count) = mean(Y(:,:,count));
 Exy(1,1,count) = mean(XY(:,:,count));
 sigx(1,1,count) = std(X(:,:,count)); %calculating the standard deviations
 sigy(1,1,count) = std(Y(:,:,count));
 end
 for count1 = 1:300
  rho(1,1,count1) = (Exy(1,1,count1) - (Ex(1,1,count1) * Ey(1,1,count1)))/(sigx(1,1,count1) * sigy(1,1,count1));
end
for count = 1:300
    Xhat(1,:,count) = ((rho(1,1,count)*(Y(1,:,count)-Ey(1,1,count))*sigx(1,1,count))/sigy(1,1,count)) + Ex(1,1,count);
end
% for count = 1:300
%     Xhat(1,:,count) = ((rho(1,1,count)*sigx(1,1,count)/sigy(1,1,count))*Y(1,:,count)) + Ex(1,1,count) -
((rho(1,1,count)*sigx(1,1,count))/sigy(1,1,count));
% end
 ers2 = X - Xhat; %error signal
 %%%%%%%Quantization%%%
 count3 = 1;
 for count1 = 1:300
    for count2 = 1:101376

           if ers2(count3,count2,count1)>= 0 && ers2(count3,count2,count1)<= 7
                ers2(count3,count2,count1) = 4;
                            elseif ers2(count3,count2,count1)>= 8 && ers2(count3,count2,count1)<= 15
                ers2(count3,count2,count1) = 12;
            elseif ers2(count3,count2,count1)>= 16 && ers2(count3,count2,count1)<= 23
                ers2(count3,count2,count1) = 20;
                            elseif ers2(count3,count2,count1)>= 24 && ers2(count3,count2,count1)<= 31
                ers2(count3,count2,count1) = 28;
            elseif ers2(count3,count2,count1)>= 32 && ers2(count3,count2,count1)<= 39
                ers2(count3,count2,count1) = 36;
                            elseif ers2(count3,count2,count1)>= 40 && ers2(count3,count2,count1)<= 47
                ers2(count3,count2,count1) = 44;
            elseif ers2(count3,count2,count1)>= 48 && ers2(count3,count2,count1)<= 55
                ers2(count3,count2,count1) = 52;
                            elseif ers2(count3,count2,count1)>= 56 && ers2(count3,count2,count1)<= 63
                ers2(count3,count2,count1) = 60;
            elseif ers2(count3,count2,count1)>= 64 && ers2(count3,count2,count1)<= 71
                ers2(count3,count2,count1) = 68;
                            elseif ers2(count3,count2,count1)>= 72 && ers2(count3,count2,count1)<= 79
                ers2(count3,count2,count1) = 76;
            elseif ers2(count3,count2,count1)>= 80 && ers2(count3,count2,count1)<= 87
                ers2(count3,count2,count1) = 84;
                         elseif ers2(count3,count2,count1)>= 88 && ers2(count3,count2,count1)<= 95
                ers2(count3,count2,count1) = 92;
                            elseif ers2(count3,count2,count1)>= 96 && ers2(count3,count2,count1)<= 103
                ers2(count3,count2,count1) = 100;
            elseif ers2(count3,count2,count1)>= 104 && ers2(count3,count2,count1)<= 111
                ers2(count3,count2,count1) = 108;
                            elseif ers2(count3,count2,count1)>= 112 && ers2(count3,count2,count1)<= 119
                ers2(count3,count2,count1) = 116;
            elseif ers2(count3,count2,count1)>= 120 && ers2(count3,count2,count1)<= 127
                ers2(count3,count2,count1) = 124;
           elseif ers2(count3,count2,count1)>= 128 && ers2(count3,count2,count1)<= 135
                ers2(count3,count2,count1) = 132;
                            elseif ers2(count3,count2,count1)>= 136 && ers2(count3,count2,count1)<= 143
                ers2(count3,count2,count1) = 140;
            elseif ers2(count3,count2,count1)>= 144 && ers2(count3,count2,count1)<= 151
                ers2(count3,count2,count1) = 148;
                            elseif ers2(count3,count2,count1)>= 152 && ers2(count3,count2,count1)<= 159
                ers2(count3,count2,count1) = 156;
            elseif ers2(count3,count2,count1)>= 160 && ers2(count3,count2,count1)<= 167
                ers2(count3,count2,count1) = 164;
                            elseif ers2(count3,count2,count1)>= 168 && ers2(count3,count2,count1)<= 175
                ers2(count3,count2,count1) = 172;
            elseif ers2(count3,count2,count1)>= 176 && ers2(count3,count2,count1)<= 183
                ers2(count3,count2,count1) = 180;
                            elseif ers2(count3,count2,count1)>= 184 && ers2(count3,count2,count1)<= 191
                ers2(count3,count2,count1) = 188;
            elseif ers2(count3,count2,count1)>= 192 && ers2(count3,count2,count1)<= 199
                ers2(count3,count2,count1) = 196;
                            elseif ers2(count3,count2,count1)>= 200 && ers2(count3,count2,count1)<= 207
                ers2(count3,count2,count1) = 204;
            elseif ers2(count3,count2,count1)>= 208 && ers2(count3,count2,count1)<= 215
                ers2(count3,count2,count1) = 212;
                         elseif ers2(count3,count2,count1)>= 216 && ers2(count3,count2,count1)<= 223
                ers2(count3,count2,count1) = 220;
                            elseif ers2(count3,count2,count1)>= 224 && ers2(count3,count2,count1)<= 231
                ers2(count3,count2,count1) = 228;
            elseif ers2(count3,count2,count1)>= 232 && ers2(count3,count2,count1)<= 239
                ers2(count3,count2,count1) = 236;
                            elseif ers2(count3,count2,count1)>= 240 && ers2(count3,count2,count1)<= 247
                ers2(count3,count2,count1) = 244;
            elseif ers2(count3,count2,count1)>= 248 && ers2(count3,count2,count1)<= 255
                ers2(count3,count2,count1) = 252;
           elseif ers2(count3,count2,count1)>= 0 && ers2(count3,count2,count1)<= -7
                ers2(count3,count2,count1) = -4;
                            elseif ers2(count3,count2,count1)>= -8 && ers2(count3,count2,count1)<= -15
                ers2(count3,count2,count1) = -12;
            elseif ers2(count3,count2,count1)>= -16 && ers2(count3,count2,count1)<= -23
                ers2(count3,count2,count1) = -20;
                            elseif ers2(count3,count2,count1)>= -24 && ers2(count3,count2,count1)<= -31
                ers2(count3,count2,count1) = -28;
            elseif ers2(count3,count2,count1)>= -32 && ers2(count3,count2,count1)<= -39
                ers2(count3,count2,count1) = -36;
                            elseif ers2(count3,count2,count1)>= -40 && ers2(count3,count2,count1)<= -47
                ers2(count3,count2,count1) = -44;
            elseif ers2(count3,count2,count1)>= -48 && ers2(count3,count2,count1)<= -55
                ers2(count3,count2,count1) = -52;
                            elseif ers2(count3,count2,count1)>= -56 && ers2(count3,count2,count1)<= -63
                ers2(count3,count2,count1) = -60;
            elseif ers2(count3,count2,count1)>= -64 && ers2(count3,count2,count1)<= -71
                ers2(count3,count2,count1) = -68;
                            elseif ers2(count3,count2,count1)>= -72 && ers2(count3,count2,count1)<= -79
                ers2(count3,count2,count1) = -76;
            elseif ers2(count3,count2,count1)>= -80 && ers2(count3,count2,count1)<= -87
                ers2(count3,count2,count1) = -84;
                         elseif ers2(count3,count2,count1)>= -88 && ers2(count3,count2,count1)<= -95
                ers2(count3,count2,count1) = -92;
                            elseif ers2(count3,count2,count1)>= -96 && ers2(count3,count2,count1)<= -103
                ers2(count3,count2,count1) = -100;
            elseif ers2(count3,count2,count1)>= -104 && ers2(count3,count2,count1)<= -111
                ers2(count3,count2,count1) = -108;
                            elseif ers2(count3,count2,count1)>= -112 && ers2(count3,count2,count1)<= -119
                ers2(count3,count2,count1) = -116;
            elseif ers2(count3,count2,count1)>= -120 && ers2(count3,count2,count1)<= -127
                ers2(count3,count2,count1) = -124;
           elseif ers2(count3,count2,count1)>= -128 && ers2(count3,count2,count1)<= -135
                ers2(count3,count2,count1) = -132;
                            elseif ers2(count3,count2,count1)>= -136 && ers2(count3,count2,count1)<= -143
                ers2(count3,count2,count1) = -140;
            elseif ers2(count3,count2,count1)>= -144 && ers2(count3,count2,count1)<= -151
                ers2(count3,count2,count1) = -148;
                            elseif ers2(count3,count2,count1)>= -152 && ers2(count3,count2,count1)<= -159
                ers2(count3,count2,count1) = -156;
            elseif ers2(count3,count2,count1)>= -160 && ers2(count3,count2,count1)<= -167
                ers2(count3,count2,count1) = -164;
                            elseif ers2(count3,count2,count1)>= -168 && ers2(count3,count2,count1)<= -175
                ers2(count3,count2,count1) = -172;
            elseif ers2(count3,count2,count1)>= -176 && ers2(count3,count2,count1)<= -183
                ers2(count3,count2,count1) = -180;
                            elseif ers2(count3,count2,count1)>= -184 && ers2(count3,count2,count1)<= -191
                ers2(count3,count2,count1) = -188;
            elseif ers2(count3,count2,count1)>= -192 && ers2(count3,count2,count1)<= -199
                ers2(count3,count2,count1) = -196;
                            elseif ers2(count3,count2,count1)>= -200 && ers2(count3,count2,count1)<= -207
                ers2(count3,count2,count1) = -204;
            elseif ers2(count3,count2,count1)>= -208 && ers2(count3,count2,count1)<= -215
                ers2(count3,count2,count1) = -212;
                         elseif ers2(count3,count2,count1)>= -216 && ers2(count3,count2,count1)<= -223
                ers2(count3,count2,count1) = -220;
```

```
                                elseif ers2(count3,count2,count1)>= -224 && ers2(count3,count2,count1)<= -231
                    ers2(count3,count2,count1) = -228;
                elseif ers2(count3,count2,count1)>= -232 && ers2(count3,count2,count1)<= -239
                    ers2(count3,count2,count1) = -236;
                                elseif ers2(count3,count2,count1)>= -240 && ers2(count3,count2,count1)<= -247
                    ers2(count3,count2,count1) = -244;
                elseif ers2(count3,count2,count1)>= -248 && ers2(count3,count2,count1)<= -255
                    ers2(count3,count2,count1) = -252;

               end
        end
    end
     %reciever side
     Xq = ers2 + Xhat;
     %%%%%%%%%%%%%%%%%%%%%%SNR%%%%%%%%%%%%%%

     Xq = double(Xq);
      for count = 1:300
          magI(1,count) = norm(X(:,:,count));
          magerr(1,count) = norm(X(:,:,count)- Xq(:,:,count));
      end

        SNR5 = 20*log10 (norm(magI)/norm(magerr));

      for count = 1:300
     X2play5(:,:,count) = reshape(Xq(:,:,count),[m,n]);
      end
      X2play5 = uint8(X2play5);
     %implay(X2play5)
    %%Plotting error signal histogram
    histogram(ers2)
    title('Error histogram: DPCM - 5 bits')
```

DCT Code:

```
I = readData('output.avi');
I = double(I);
[m,n] = size(I(:,:,1));
DCT3 = zeros([288,352,300]);
I3 = zeros([288,352,300]);

for count = 1:300
T = dctmtx(8); %returns an 8 x 8 DCT matrix
dct = @(block_struct) T * block_struct.data * T';
DCT3(:,:,count) = blockproc(I(:,:,count),[8 8],dct);
end


for count1 = 1:8
    for count2 = 1:8
 tt(count1,count2) = C(count1)*C(count2)/4;
    end
end

for count1 = 1:7:281
    for count2 = 1:7:345
        norm (count1:count1+7,count2:count2+7) = tt;
    end
end
norm(:,352) = norm(:,351);

for count = 1:300
DCTnorm(:,:,count) = DCT3(:,:,count).*norm;
end
%%%%quantization%%%%
for count = 1:300
    DCTnormq(:,:,count) = PCMfun(DCTnorm(:,:,count),8);
end
%%%%denormalization%%%
for count1 = 1:8
    for count2 = 1:8
 ttt(count1,count2) = 4/(C(count1)*C(count2));
    end
end

for count1 = 1:7:281
    for count2 = 1:7:345
        denorm (count1:count1+7,count2:count2+7) = ttt;
    end
end
denorm(:,352) = denorm(:,351);

for count = 1:300
DCTdenorm(:,:,count) = DCTnorm(:,:,count).*denorm;
end
%%%%inverse DCT%%%%%
count2 = 1
for count = 1:300
invdct = @(block_struct) T' * block_struct.data * T;
I3(:,:,count) = blockproc(DCTdenorm(:,:,count),[8 8],invdct);
end

I3 = uint8(I3);
I = uint8(I);
implay(I3)
histogram(DCTnorm)
```