Image Steganography based on Block-DCT and Huffman Encoding

A Project Overview

submitted to

Faculty: Prof. Preetha K.S

Slot: B1+TB1

In

Information Theory and Coding

(ECE4007)

By:

J. D. P. Rithvik – 17BEC0223

Hari Maheswar Reddy-17BEC0379

B Siva Anil Kumar Reddy – 17BEC0117

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Electronics and Communication Engineering

Contents:

❖ Abstract:

Image steganography is the art of hiding information into a cover image. Hiding information into a cover image using Block – DCT (where DCT is used to transform original image (cover image) blocks from spatial domain to frequency domain) and Then Huffman encoding is also performed on the secret messages/images as this algorithm show that the information has a high capacity and a good invisibility before embedding and each bit of Huffman code of secret message/image is embedded in the frequency domain by altering the least significant bit of each of the Quantized DCT coefficients of cover image blocks.

❖ Introduction:

With the development of Internet technologies, digital media can be transmitted conveniently over the Internet. However, message transmissions over the Internet still have to face all kinds of security problems. Hiding information into a cover image for providing security. Because It doesn't seem fishy enough for hackers to see a normal image as it contains any secret information, it becomes more important for the users to maintain privacy. And to provide such security and privacy to the user, hiding information in an image and transmitting to other is very important to protect from any unauthorised user access.

❖ Literature Survey:

Steganography is the art and science of hiding information in a cover document such as digital images in a way that conceals the existence of hidden data. The word steganography in Greek means "covered writing" (Greek words "stegos" meaning "cover" and "grafia" meaning "writing"). The main objective of steganography is to communicate securely in such a way that the true message is not visible to the observer. That is unwanted parties should not be able to distinguish in any sense between cover-image (image not containing any secret

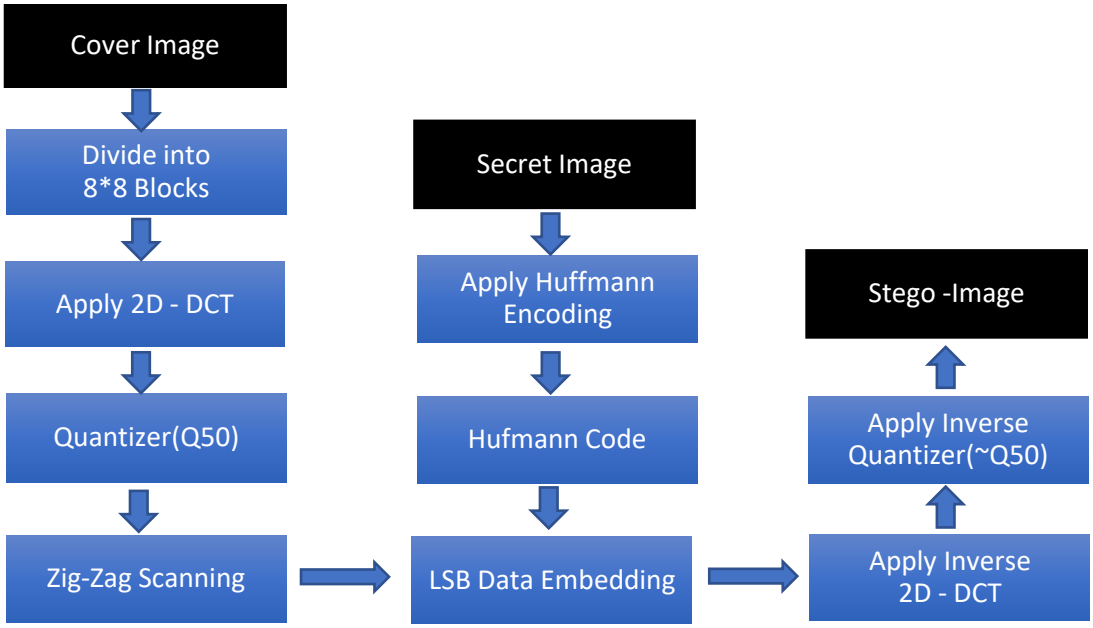
message) and stego-image (modified cover-image that containing secret message). Thus, the stego-image should not deviate much from original cover-image. Today steganography is mostly used on computers with digital data being the carriers and networks being the high-speed delivery channels.

❖ Tools to be used:

- We preferred **MATLAB Software** based on complexity of the code and other factors such as **Speed, Functionalities, Code Readability, Cost**

❖ Block Diagram / Flow chart:

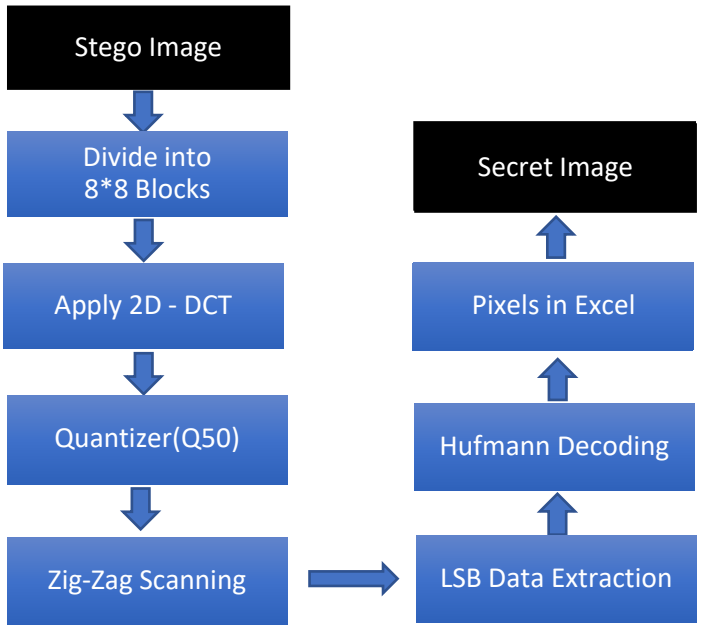
Embedding Algorithm:



The challenge in this work was to find a way to camouflage a secret message in an image without perceptible degrading the image quality and to provide better resistance against steganalysis process. Therefore, a combination of frequency domain by means of DCT and LSB technique of spatial domain steganography has been used to hide data. Two dimensional DCT converts the image block from

spatial domain to frequency domain and then data bits are embedded by altering LSB of DCT coefficients

Extraction Algorithm:



Methodology:

➢ Embedding Algorithm:

I. Image selection:

When only one LSB of each DCT coefficients of cover image of size M×N is used to embed the data, the amount of information that can be embedded in the cover image is M*N bits. Considering each pixel is represented using eight bits, the ideal size of a single image that can be embedded is X×Y, where

$$1*X*Y*8=M*N \quad \text{eq (1)}$$

To embed a single image, all eight bits of pixels in the secret image can be used. Embedding more than one secret image uses only the MSBs of pixels in secret image. The number of MSBs used depends on the number of images to be hidden. Four, two and one MSB(s) are selected to embed two, four and eight images respectively.

II. Discrete Cosine Transform(17BEC0223)

Discrete Cosine Transform (DCT) [25] is used to transform the image from spatial domain to frequency domain. The cover image of size M×N is first divided into 8×8 blocks and then DCT is applied on each block. DCT-II is used for two dimensional images of M×N. The equation for DCT-II is given by:

$$D(i,j) = \frac{1}{\sqrt{2N}} C(i)C(j) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} p(x,y) \cos\left[\frac{(2x+1)i\pi}{2N} \right] \cos\left[\frac{(2y+1)j\pi}{2N} \right]$$

$$C(u) = \begin{cases} \frac{1}{\sqrt{2}} & \text{if } u = 0 \\ 1 & \text{if } u > 0 \end{cases}$$

In DCT block lower frequency coefficients are at upper left positions and high frequency coefficients are lower right positions. Low frequency coefficients are of larger value than high frequency coefficients.

To get the matrix form of DCT, we use the following equation

$$T_{i,j} = \begin{cases} \frac{1}{\sqrt{N}} & \text{if } i = 0 \\ \sqrt{\frac{2}{N}} \cos\left[\frac{(2j+1)i\pi}{2N} \right] & \text{if } i > 0 \end{cases}$$

A DCT matrix of an 8×8 block is

$$T = \begin{bmatrix} .3536 & .3536 & .3536 & .3536 & .3536 & .3536 & .3536 & .3536 \\ .4904 & .4157 & .2778 & .0975 & -.0975 & -.2778 & -.4157 & -.4904 \\ .4619 & .1913 & -.1913 & -.4619 & -.4619 & -.1913 & .1913 & .4619 \\ .4157 & -.0975 & -.4904 & -.2778 & .2778 & .4904 & .0975 & -.4157 \\ .3536 & -.3536 & -.3536 & .3536 & .3536 & -.3536 & -.3536 & .3536 \\ .2778 & -.4904 & .0975 & .4157 & -.4157 & -.0975 & .4904 & -.2778 \\ .1913 & -.4619 & .4619 & -.1913 & -.1913 & .4619 & -.4619 & .1913 \\ .0975 & -.2778 & .4157 & -.4904 & .4904 & -.4157 & .2778 & -.0975 \end{bmatrix}$$

On applying DCT on a block of image matrix I, a matrix F is obtained. Matrix I have pixels as its entries whereas matrix F has DCT coefficients as its entry. The obtained matrix F has higher values at the top left. This higher value indicates that it corresponds to lower frequencies and higher signal energies. The values other than that are very small enough such that it can be neglected with no or less distortion.

III. Quantization(17BEC0117)

Quantization is done to compress the DCT coefficients obtained in previous step. Since human eyes are insensitive to high frequency components, these are made zero using standard Quantization matrix which is

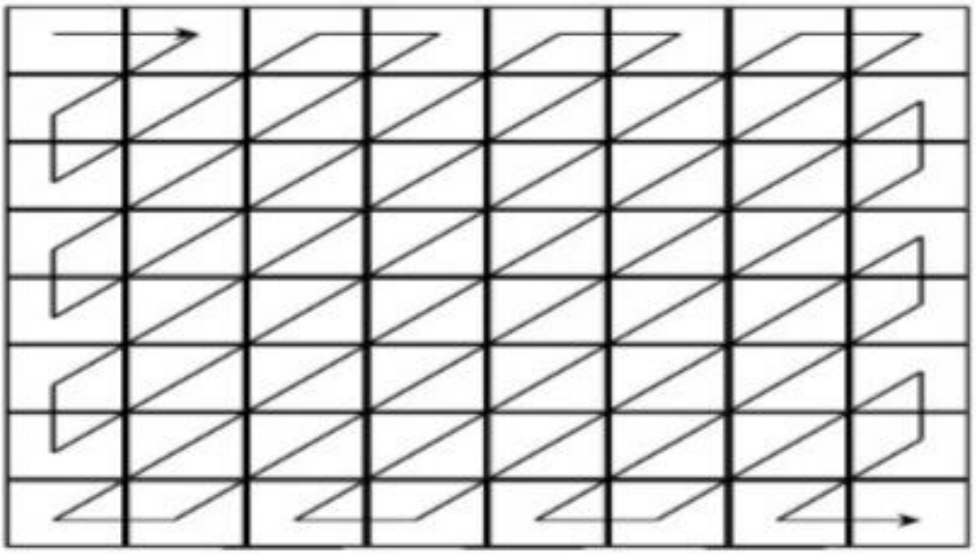
$$Q_{50} = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

Quantization is done by dividing each value in matrix K with corresponding values in quantization matrix Q. This will result in matrix P, which has the quantized DCT coefficients

$$P(i, j) = F(i, j) / Q(i, j)$$

IV. Zig – Zag Scanning: (17BEC0223)

The matrix P, obtained in previous step may have intra block correlation. To remove this, zigzag scanning is done on matrix P which results in one-dimensional array A. The values in this array will be in ascending order which means that low frequency elements are in the beginning.



Zigzag algorithm is used for scanning in Zigzag manner. The coefficient scanning plays an important role in block-based image and video coding standards, such as JPEG, MPEG etc. in this coding standards method, the Zigzag method is used for image or frame coding. The Zigzag method scans the coefficients of image.

This method enhances security and the quality of image instead of high capacity of concealed information. Zigzag PVD uses the difference of each pair of pixels to determine the number of message bits that can be embedded into that pair of pixels. It starts at the upper-left corner of the Cover image and scans the image in a zigzag manner.

V. Huffman Coding of Secret Image:(17BEC0223)

Huffman coding generate the smallest probable number of code symbols for any one source symbol. The source symbol is likely to be either intensities of an image or intensity mapping operation output. In the first step of Huffman procedure, a chain of source reductions by arranging the probabilities in descending order and merging the two lowest probable symbols to create a single compound symbol that replaces in the successive source reduction. This procedure is repeated continuously up to two probabilities of two compound symbols are only left.

Table 6.1a: Huffman Source reduction

Original Source		Source Reduction			
Source	Probability	1	2	3	4
a_2	0.4	0.4	0.4	0.4	0.6
a_6	0.3	0.3	0.3	0.3	0.4
a_1	0.1	0.1	0.2	0.3	
a_4	0.1	0.1	0.1		
a_3	0.06	0.1			
a_5	0.04				

The list of symbols and their corresponding probabilities are placed in descending order. A compound symbol with probability ‘0.1’ is obtained by merging the least probabilities ‘0.06’ and ‘0.04’. This is located in source reduction column ‘1’. Once again, the probabilities of source symbols are arranged in descending order and the procedure is continued

until only two probabilities are recognized. These probabilities observed at far right are ‘0.6’ and ‘0.4’.

Table 6.1b : Huffman Assignment Procedure

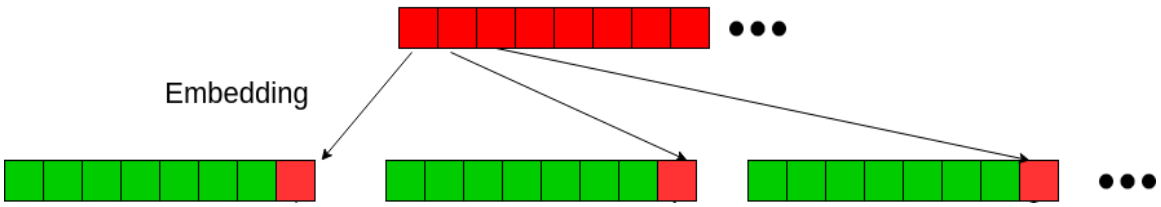
Original Source			Source Reduction							
Source	Probability	Code	1	Code	2	Code	3	Code	4	Code
a_2	0.4	1	0.4	1	0.4	1	0.4	1	0.6	0
a_6	0.3	00	0.3	00	0.3	00	0.3	00	0.4	1
a_1	0.1	011	0.1	011	0.2	010	0.3	01		
a_4	0.1	0100	0.1	0100	0.1	011				
a_3	0.06	01010	0.1	0101						
a_5	0.04	01011								

In the second step, Huffman procedure is to prepare a code for each compact source starting with smallest basis and operating back to its original source. As shown in the table 6.1b, the code symbols 0 and 1 are assigned to the two symbols of probabilities ‘0.6’ and ‘0.4’ on the right. The source symbol with probability 0.6 is generated by merging two symbols of probabilities ‘0.3’ and ‘0.3’ in the reduced source. So, to code both of these symbols the code symbol 0 is appended along with 0 and 1 to make a distinction from one another. This procedure is repeated for each reduced source symbol till the original source code is attained. The final code become visible at the far-left in table 6.1b. The average code length is defined as the product of probability of the symbol and number of bits utilized to represent the same symbol.

VI. LSB – Embedding:(17BEC0223)

Secret image is embedded by altering the LSB of the values in the array A. Value in A is made even if the data bit is even, else odd.

For embedding multiple images, LSBs of the data are neglected since it carries less information. To embed two images, four MSBs are considered and other four bits are neglected. Thus, it satisfies eq (1) with 2*X*Y*4=M*N where 2 is the number of images and 4 is the number of bits. Four images can be embedded by considering only two MSBs, neglecting the remaining six bits. Thus eq (1) becomes 4*X*Y*2=M*N.



Similarly, 8 images can be embedded by considering only one MSB. Single image is embedded by using all bits in its pixels.

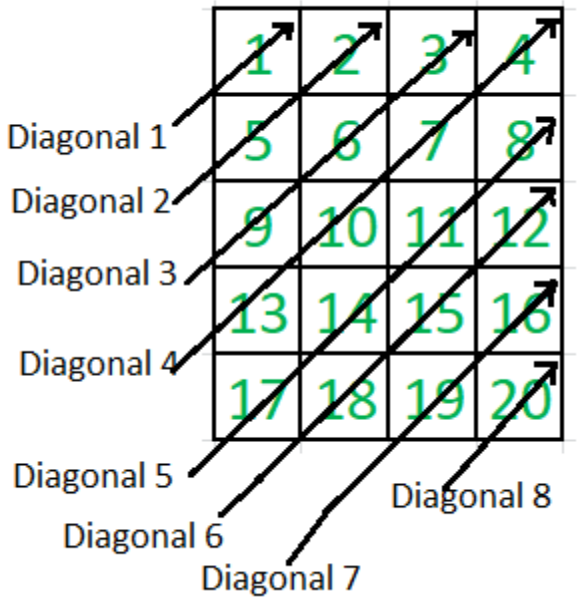
VII. Image construction:

After embedding, inverse zigzag is applied on array A to convert it back to the 1024×1024 matrix. Then this block is dequantized by multiplying each value in matrix with the corresponding value in quantization matrix Q and Inverse Discrete Cosine Transform (IDCT) is applied to transform it back to spatial domain. Finally, all 8×8 blocks are combined to form the stego-image S.

VIII. Inverse Zig – Zag Scanning: (17BEC0223)

inverse zigzag is applied on array A to convert it back to the 1024×1024 matrix

izigzag (in, vmax, hmax) takes a one-dimensional array and converts it back to (vmax x hmax) matrix.



IX. Inverse Quantization: (17BEC0117)

Each of the decoded DCT coefficients in block should be inverse quantized through multiplication by the corresponding value from the weighting matrix and by the quantizer scale factor, which are read from the bit stream. Before this operation is performed, the DCT coefficients should be rearranged from zigzag scanning sequence to linear sequence.

X. Inverse Discrete Cosine Transform: (17BEC0117)

Two-Dimensional IDCT Equation

$$f[m, n] = \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} c[u]\, c[v]\, F[u, v] \cos[(2m + 1)u\pi/2N] \cos[(2n + 1)v\pi/2N]$$

where:

m, n = image result pixel indices(0, 1, 2, …, N – 1),
F[u, v] = N by N DCT result,
c[λ] = 1 for λ=0 and c[λ]=2 for λ=1,2,3,…N-1
f[m, n] = N by N IDCT result

➢ Extraction Algorithm:

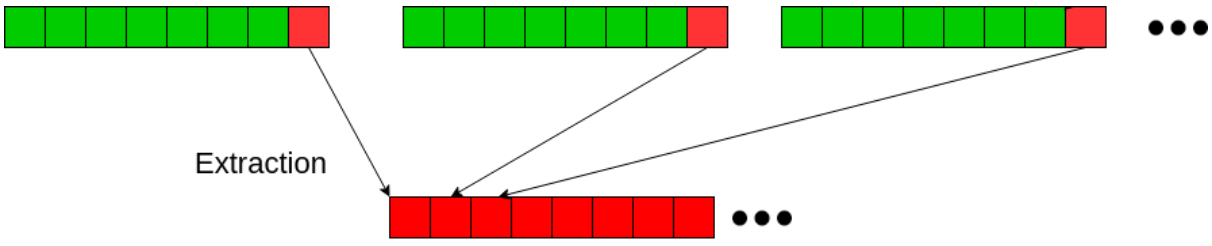
Extraction of secret images from stego image.

XI. 2D – DCT, Quantization, and Zig-Zag Scanning of Stego - Image:(17BEC0379)

The stego image to be broken into 8×8 blocks, transform it to frequency domain by applying DCT, quantize the coefficients, perform zigzag scanning and extract the data. The extracted bit is 0 if the value is even, else the extracted bit is 1. Concatenate 8 bits together to form a pixel. These pixels are arranged to reconstruct the secret image.

XII. LSB - Extraction:(17BEC0379)

The extraction process is simple. We need to first calculate how many pixels is the text stored in. For example, the text “Secret Image” has ‘len1’ Pixels. Each Pixel is represented in Huffman Coded bits. So, the number of pixels in which the text is stored will be length of Huffman code.



Now after knowing this, we need to traverse through the image, one pixel at a time. We store the Least Significant Bit (LSB) of each pixel in an array extracted_bits. After extracting the LSBs of the required pixels, we need to take every 256 bits from

extracted_bits and convert it to the corresponding array of pixels. In this way, the Image stored in the stego image can be extracted.

XIII. Huffman Decoding of Secret Image:(17BEC0379)

In Huffman coding the optimal code is produced for a set of symbols. A unique error less decoding scheme is achieved in a simple look-up-table approach. Sequence of Huffman encoded symbols can be deciphered by inspecting the individual symbols of the sequence from left-to-right. Using the binary code present in Table of Huffman Dictionary, a left-to-right inspection of the encoded string ‘1010100111100’ unveils the decoded message as “a2 a3 a1 a2 a2 a6”. Hence with Huffman decoding process the compressed data of the image can be decompressed.

❖ Results and Discussion:

Inputs:

Figure 1: Secret Image of Resolution 256*256

Figure 1: Cover Image of Resolution 1024*1024

Output:

Figure 3 Grey Scaled Stego Image of Resolution 1024*1024

Figure 4 Stego Image of Resolution 1024*1024

Figure 5: Extracted Image of Resolution 256*256

❖ References:

1. Danny Adiyan Z.1 , Tito Waluyo Purboyo2 and Ratna Astuti Nugrahaeni International Journal of Applied Engineering Research ISSN 0973-4562 Volume 13, Number 1 (2018) pp. 442-448 © Research India Publications. http://www.ripublication.com
2. Devadath C Prabhu Department of CS&E, PESIT South Campus, Bengaluru, India Multiple Image Steganography using LSB-DCT Technique, International Journal of Engineering Research & Technology (IJERT) ISSN: 2278-0181 Published by, www.ijert.org ICACT - 2016 Conference Proceedings
3. Ketaki Bhaskar, Mitali Bakale, Priyanka Chaure, Priti Shirke Image Steganography for data hiding Using Huffman code, Zigzag and OPAP International Journal of Emerging Trends & Technology in Computer Science (IJETTCS) Volume 4, Issue 6, November - December 2015 ISSN 2278-6856
4. Deepak Singla, Rupali Syal /Data Security Using LSB & DCT Steganography In Images International Journal Of Computational Engineering Research/ ISSN: 2250–3005
5. Papakostas, George & Koulouriotis, Dimitrios & Karakasis, Evangelos G.. (2009). Efficient 2-D DCT Computation from an Image Representation Point of View. 10.5772/7043
6. Poljicak, A., Botella, G., Garcia, C. et al. J Real-Time Image Proc (2016). doi:10.1007/s11554-016-0616-9

❖ MATLAB Code:

✓ Coding Techniques Used:

- 2d - DCT, Quantizing and Zig Zag Scanning of Cover Photo

```
% Cover Image
a1 = imread('wall.jpg');
a1 = imresize(a1,[1024 1024]);
figure(),imshow(a1),title('cover image');
a=a1(:,:,1);
t1=a1(:,:,2);
t2=a1(:,:,3);
a=rgb2gray(a1);
[r,c]=size(a);
a=im2double(a);
a=a*255;
b=8;
n1=(floor(r/(b)))*b;
n2=(floor(c/b))*b;
a=imresize(a,[n1,n2]);
t1=imresize(t1,[n1,n2]);
t2=imresize(t2,[n1,n2]);
I=a;
z=zeros(n1,n2);


for i=1:b:n1
    for j=1:b:n2
      f=I(i:i+b-1,j:j+b-1);
      f1=dct2(f);
      z(i:i+b-1,j:j+b-1)=f1;
    end
end
figure(2),imshow(z/255);


quantization_table = [
 16  11  10  16  24   40   51   61;
 12  12  14  19  26   58   60   55;
 14  13  16  24  40   57   69   56;
 14  17  22  29  51   87   80   62;
 18  22  37  56  68   109  103  77;
 24  35  55  64  81   104  113  92;
 49  64  78  87  103  121  120  101;
 72  92  95  98  112  100  103  99];
quant1=zeros(n1,n2);
    for i=1:b:n1
        for j=1:b:n2
            for ii=1:b
                for jj=1:b
                    aa=z(i+ii-1,j+jj-1);
                    quant1(i+ii-1,j+jj-1)=(aa/quantization_table(ii,jj));
```

```
                end
            end
        end
    end
quant=round(quant1);
B2 = quant;
figure(3),imshow(quant/255);
B2(1:8,1:8)
B2 = zigzag(B2);
len = length(B2)
```

- Huffman Encoding of Secret Image

```
% Image to be Encoded
inputimg = imread('cameraman.tif');
imshow('cameraman.tif');
vecImg = reshape(inputimg,1,[]);
Data = vecImg;
% Perform Huffman Encoding
HEAD=0;
%%%%%%%%%%%%%%%%%%%
%--Compute Header--------
POS=0;
S_=size(Data);
for i=1:S_(2)
    if (POS~=0)
      S=size(HEAD); F=0;
      k=1;
      while (F==0 && k<=S(2))
         if (Data(i)==HEAD(k))  F=1; end
         k=k+1;
      end
    else F=0;
    end
    if (F==0)
      POS=POS+1;
      HEAD(POS)=Data(i);
    end
end
fprintf('Header:\n');
display(HEAD);
%%%%%%%%Compute probability for symbols%%%%%%%%%%%%
S_H=size(HEAD);
Count(1:S_H(2))=0;
for i=1:S_H(2)
    for j=1:S_(2)
        if (Data(j)==HEAD(i))
            Count(i)=Count(i)+1;
        end
    end
end
Count=Count./S_(2);
fprintf('probability for symbols\n');
display(Count);
```

```
%%%Sort accoridng to maximum
number%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:S_H(2)-1
    for j=i+1:S_H(2)
        if (Count(j)>Count(i))
            T1=Count(i); Count(i)=Count(j); Count(j)=T1;
            T1=HEAD(i); HEAD(i)=HEAD(j); HEAD(j)=T1;
        end
    end
end
fprintf('Sort Results\n');
display(HEAD);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[dict,avglen] = huffmandict(HEAD,Count); % Create dictionary.
hcode = huffmanenco(Data,dict); % Encode the data.
display(hcode);
len1 = size(hcode,2)
bin_num_message = hcode(:);
[n5,n6]=size(hcode);
```

- Embedding of Image

```
%embedding using LSB
input = B2;
input(:,1:8)
output = input;
embed_counter = 1;
for i = 1 : len
    if(embed_counter <= len1)
        % Finding Least Significant Bit of the current pixel
        LSB = mod(input(:,i), 2);
        % Find whether the bit is same or needs to change
        temp = double(xor(LSB, bin_num_message(embed_counter)));
        % Updating the output to input + temp
        output(i) = input(i)+temp;
        % Increment the embed counter
        embed_counter = embed_counter+1;
    end
end
output(:,1:8)
```

- Inverse Zig – Zag Scanning, Inverse Quantization and Inverse Dct to get Stego Image

```
%Performing Inverse Quantization By Multiplying with q_mtx and 2D - DCT
B2 = izigzag(output,1024,1024);

quant=zeros(n1,n2);


    for i=1:b:n1
        for j=1:b:n2
            for ii=1:b
                for jj=1:b
                    aa=B2(i+ii-1,j+jj-1);
                    quant(i+ii-1,j+jj-1)=(aa*quantization_table(ii,jj));
```

```
                end
            end
        end
    end
z11=zeros(n1,n2);
for i=1:b:n1
    for j=1:b:n2
      f=quant(i:i+b-1,j:j+b-1);
      f1=idct2(f);
      z11(i:i+b-1,j:j+b-1)=f1;
    end
end
figure(),imshow(z11/255),title('final grayscale image');
%z1=round(z11);
fi=cat(3,z11,t1,t2);
figure(),imshow(fi),title('stego image');
imwrite(fi,'stego01.jpg');
```

- Decoding od Stego Image using LSB Extraction

```
%Decoding of Image
s=z11/255;
s1=im2double(s);
s1=s1*255;
% DCT and Quantiztion
[n7,n8]=size(s1);
s2=zeros(n7,n8);
for i=1:b:n7
    for j=1:b:n8
        s3=s1(i:i+b-1,j:j+b-1);
        s4=dct2(s3);
        s2(i:i+b-1,j:j+b-1)=s4;
    end
end
s2=round(s2);
quant1=zeros(n1,n2);


    for i=1:b:n1
        for j=1:b:n2
            for ii=1:b
                for jj=1:b
                    aa=s2(i+ii-1,j+jj-1);
                    quant1(i+ii-1,j+jj-1)=(aa/quantization_table(ii,jj));
                end
            end
        end
    end
quant1=round(quant1);
B2 = zigzag(quant1);
B2(1:10)
% Get height and width for traversing through the image
message_length =len1;
% counter to keep track of number of bits extracted
```

```
counter = 1;
% Traverse through the image
for i = 1 : len1
        % If more bits remain to be extracted
        if (counter <= message_length)
        % Store the LSB of the pixel in extracted_bits
        extracted_bits(i) = mod(B2(i),2);
        % Increment the counter
        counter = counter + 1;
        end
end
```

```
display(extracted_bits);
h2=huffmandeco(extracted_bits,dict);
display(h2);
SecretImage = reshape(h2,[256 256]);
uint8(SecretImage);
display(SecretImage);
xlswrite('SecretImage.xls',SecretImage);

input_image = xlsread('SecretImage.xls');
```

- Functions for using Zig Zag Scanning

```
function output = zigzag(in)
h = 1;
v = 1;
vmin = 1;
hmin = 1;
vmax = size(in, 1);
hmax = size(in, 2);
i = 1;
output = zeros(1, vmax * hmax);
%----------------------------------
while ((v <= vmax) && (h <= hmax))

    if (mod(h + v, 2) == 0)                 % going up
        if (v == vmin)
            output(i) = in(v, h);        % if we got to the first line
            if (h == hmax)
             v = v + 1;
           else
              h = h + 1;
            end
            i = i + 1;
        elseif ((h == hmax) && (v < vmax))   % if we got to the last column
            output(i) = in(v, h);
            v = v + 1;
            i = i + 1;
```

```
        elseif ((v > vmin) && (h < hmax))    % all other cases
            output(i) = in(v, h);
            v = v - 1;
            h = h + 1;
            i = i + 1;
     end

    else                                     % going down
       if ((v == vmax) && (h <= hmax))       % if we got to the last line
            output(i) = in(v, h);
            h = h + 1;
            i = i + 1;

       elseif (h == hmin)                    % if we got to the first column
            output(i) = in(v, h);
            if (v == vmax)
            h = h + 1;
          else
              v = v + 1;
            end
            i = i + 1;
       elseif ((v < vmax) && (h > hmin))     % all other cases
            output(i) = in(v, h);
            v = v + 1;
            h = h - 1;
            i = i + 1;
        end
    end
    if ((v == vmax) && (h == hmax))          % bottom right element
        output(i) = in(v, h);
        break
    end
end
end



function output = izigzag(in, vmax, hmax)
% initializing the variables
%----------------------------------
h = 1;
v = 1;
vmin = 1;
hmin = 1;
output = zeros(vmax, hmax);
i = 1;
%----------------------------------
while ((v <= vmax) & (h <= hmax))
    if (mod(h + v, 2) == 0)                % going up
        if (v == vmin)
            output(v, h) = in(i);
            if (h == hmax)
            v = v + 1;
```

```
          else
              h = h + 1;
            end;
            i = i + 1;
        elseif ((h == hmax) & (v < vmax))
            output(v, h) = in(i);
            i;
            v = v + 1;
            i = i + 1;
        elseif ((v > vmin) & (h < hmax))
            output(v, h) = in(i);
            v = v - 1;
            h = h + 1;
            i = i + 1;
        end;

    else                                    % going down
       if ((v == vmax) & (h <= hmax))
            output(v, h) = in(i);
            h = h + 1;
            i = i + 1;

       elseif (h == hmin)
            output(v, h) = in(i);
            if (v == vmax)
            h = h + 1;
          else
              v = v + 1;
            end;
            i = i + 1;
       elseif ((v < vmax) & (h > hmin))
            output(v, h) = in(i);
            v = v + 1;
            h = h - 1;
            i = i + 1;
        end;
    end;
    if ((v == vmax) & (h == hmax))
        output(v, h) = in(i);
        break
    end;
end;


end
```

Report Done By: J.D.P. Rithvik(17BEC0223)

THANK YOU