Digital Signal Processing Lab

Laboratory report submitted for the partial fulfillment
of the requirements for the degree of

Bachelor of Technology
in
Electronics and Communication Engineering

by

Mohit Akhouri - 19ucc023

Course Coordinator
Dr. Divyang Rawal

Department of Electronics and Communication Engineering
The LNM Institute of Information Technology, Jaipur

September 2021

Contents

Chapter *6*

Experiment - 6

6.1 Aim of the Experiment

- Discrete Cosine Transform and it's energy compaction property
- Simulink based image compression

6.2 Software Used

- MATLAB
- Simulink

6.3 Theory

6.3.1 About Image Compression :

Image compression is a type of data compression applied to digital images, to reduce their cost for storage or transmission. Algorithms may take advantage of visual perception and the statistical properties of image data to provide superior results compared with generic data compression methods which are used for other digital data.

6.3.1.1 Lossy and Lossless Image Compression :

Image compression may be **lossy** or **lossless**. **Lossless compression** is preferred for **archival purposes** and often for medical imaging, technical drawings, clip art, or comics. Lossy compression methods, especially when used at **low bit rates**, introduce **compression artifacts**. **Lossy methods** are especially suitable for **natural images** such as photographs in applications where minor (sometimes imperceptible) loss of fidelity is acceptable to achieve a substantial reduction in bit rate. Lossy compression that produces negligible differences may be called visually lossless.

6.3.2 About Discrete Cosine Transform (DCT) :

A **discrete cosine transform (DCT)** expresses a finite sequence of data points in terms of a **sum of cosine functions oscillating at different frequencies**. The DCT, first proposed by **Nasir Ahmed** in 1972, is a widely used transformation technique in **signal processing** and **data compression**. It is used in most digital media, including **digital images** (such as JPEG and HEIF, where small high-frequency components can be discarded), **digital video** (such as MPEG and H.26x), **digital audio** (such as Dolby Digital, MP3 and AAC), **digital television** (such as SDTV, HDTV and VOD), **digital radio** (such as AAC+ and DAB+), and **speech coding** (such as AAC-LD, Siren and Opus). DCTs are also important to numerous other applications in science and engineering, such as spectral methods for the numerical solution of **partial differential equations**.

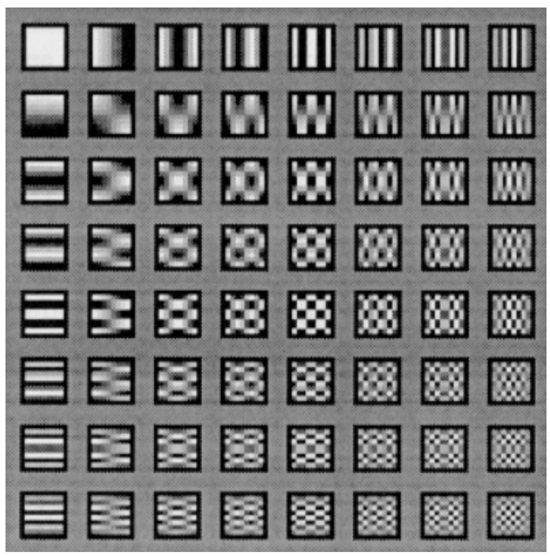

Figure 6.1 plot of the 64 (8 x 8) DCT basis functions

The **2D-Discrete Cosine Transform** (ImF) of a image **Im** of size NxN is given as :

$$ImF(k,l) = \frac{1}{\sqrt{2N}}\beta(k)\beta(l)\sum_{i=0}^{N-1}\sum_{j=0}^{N-1} Im(i,j)cos\left(\frac{\pi(2j+1)k}{2N}\right)cos\left(\frac{\pi(2i+1)l}{2N}\right) \tag{6.1}$$

In the above equation , **Beta function** (β) is defined as :

$$\beta(u) = \frac{1}{\sqrt{2}} \qquad u = 0$$

$$\beta(u) = 1 \qquad u > 0$$

DCT compression, also known as **block compression**, compresses data in sets of discrete DCT blocks. DCT blocks can have a number of sizes, including 8x8 pixels for the standard DCT, and varied integer DCT sizes between 4x4 and 32x32 pixels. The DCT has a strong **energy compaction property**, capable of achieving high quality at high data compression ratios. However, **blocky compression artifacts** can appear when heavy DCT compression is applied.

6.4 Code and results

6.4.1 Using Inbuilt DCT and IDCT to compress and reconstruct any input image :

```
% 19ucc023
% Mohit Akhouri
% Experiment 6 - Observation 1

% This code will apply inbuilt functions "dct2" and "idct2" on the
 image
% "cameraman.tif" and observe the compressed image
% Lastly , after knocking off half the pixels , we will observe the
% artifact effects in the image

clc;
clear all;
close all;

img = imread('cameraman.tif'); % Reading of image file in variable
 'img'

figure;
subplot(1,2,1);
imshow(img);
title('Original Image - cameraman.tif'); % Plot of Original Image

dct_img = dct2(img); % computing Discrete Cosine Transform ( DCT ) of
 img

subplot(1,2,2);
imshow(dct_img);
title('DCT of image - cameraman.tif'); % Plot of DCT of camerman.tif
sgtitle('19ucc023 - Mohit Akhouri');

% Compression Algorithm for the image starts from here
% ALGORITHM : knocking off half the pixels in the compressed image
for i=1:256
    for j=1:256
        if j<=128
            dct_img(i,j)=dct_img(i,j); % keeping half the pixels
        else
            dct_img(i,j)=0; % knocking off other half of pixels
        end
    end
end

figure;
subplot(1,2,1);
imshow(dct_img); % plot of DCT of image after knocking off half pixels
title('DCT of image - cameraman.tif after KNOCKING OFF half pixels');

idct_img = uint8(idct2(dct_img)); % computing IDCT of compressed image
 (dct_img) after knocking off half pixels

subplot(1,2,2);
```

Figure 6.2 Part 1 of the code for observation 1

```
imshow(idct_img); % plot of IDCT of compressed image demonstrating
 ARTIFACT EFFECTS
title('IDCT of compressed image - cameraman.tif after KNOCKING OFF
 half pixels (ARTIFACT EFFECTS)');
sgtitle('19ucc023 - Mohit Akhouri');
```

Figure 6.3 Part 2 of the code for observation 1

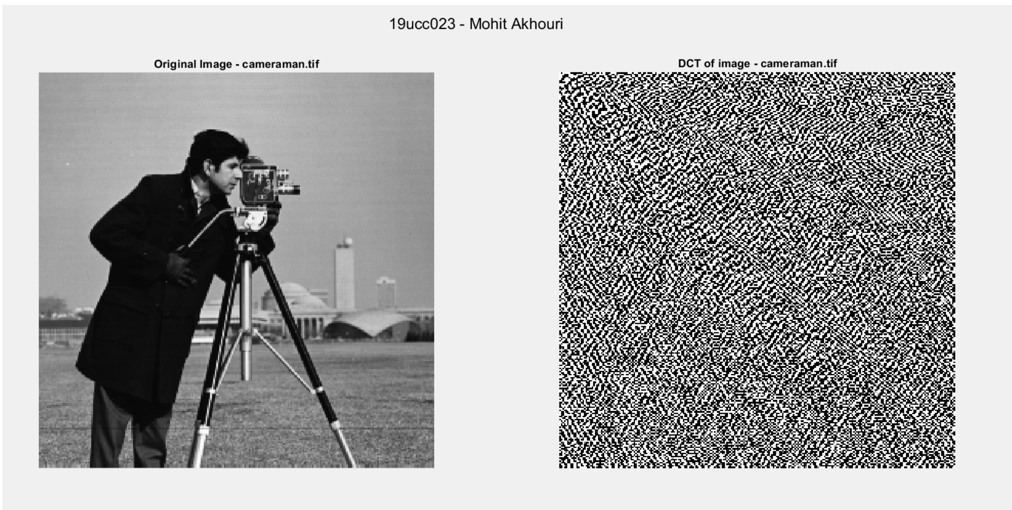

Figure 6.4 Plot of Original Image and DCT of the Image

Figure 6.5 Plot of DCT after **knocking half pixels** and reconstructed Image using IDCT

6.4.2 Using User-Defined function myCompression.m to verify the results of Observation 1 :

```
% 19ucc023
% Mohit Akhouri
% Experiment 6 - Observation 2

% This code will use the function "myCompression.m" function to
 calculate
% the DCT of image "cameraman.tif" and compare the results with
 inbuilt
% function dct2 and idct2

clc;
clear all;
close all;

img = imread('cameraman.tif'); % Reading of image file in variable
 'img'

dct_inbuilt = dct2(img); % INBUILT DCT of image - cameraman.tif
dct_myCompression = myCompression(img); % USER-DEFINED DCT of image -
 cameraman.tif

figure;
imshow(img); % Plot of image - cameraman.tif
title('Original Image - cameraman.tif');
sgtitle('19ucc023 - Mohit Akhouri');

figure;
subplot(1,2,1);
imshow(dct_inbuilt); % Plot of INBUILT DCT of image - cameraman.tif
title('DCT of image - cameraman.tif from INBUILT FUNCTION dct2(im)');

subplot(1,2,2);
imshow(dct_myCompression); % Plot of DCT calculated from USER-DEFINED
 Function 'myCompression'
title('DCT of image - cameraman.tif from USER DEFINED FUNCTION
 myCompression.m');
sgtitle('19ucc023 - Mohit Akhouri');
```

Published with MATLAB® R2020b

Figure 6.6 Code for the observation 2

Figure 6.7 Plot of Original Image cameraman.tif

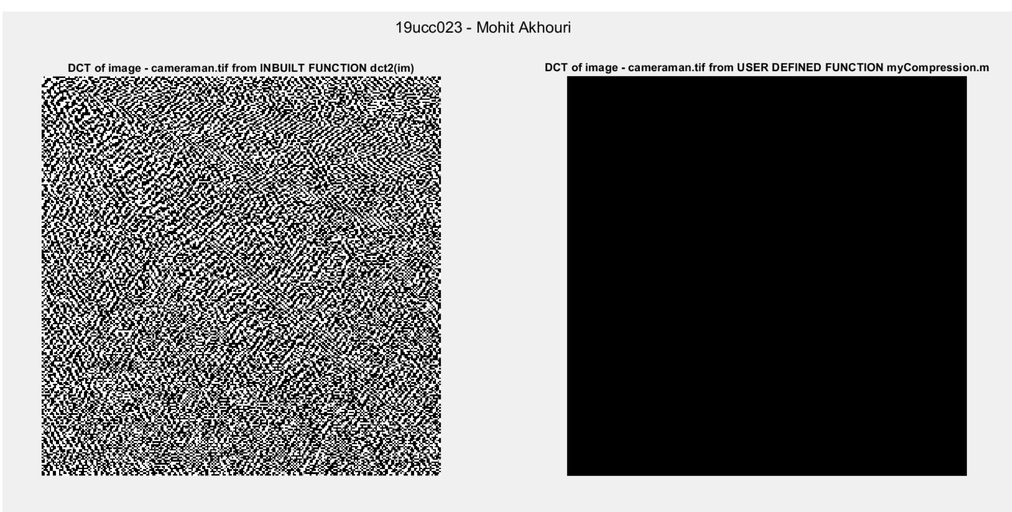

Figure 6.8 Plots of DCT obtained via inbuilt and user-defined functions

6.4.3 Observing Artfifact effects for different values of compression ratio (ρ) :

```
% 19ucc023
% Mohit Akhouri
% Experiment 6 - Observation 3 and Observation 4

% ALGORITHM :
% This code will apply the compression algorithm for 4 different
 cases :
% top 8 coefficients out of 64 , top 16 coefficients out of 64 ,
% top 32 coefficients out of 64 , top 48 coefficients out of 64
% and observe the artifact effects

% This code will also plot the graph between the mean square error and
% compression ratio

clc;
clear all;
close all;

img = imread('cameraman.tif'); % Reading of image file in variable
 'img'
N = size(img,1); % storing the size of the image (256x256) in variable
 'N'

% CASE 1 : Keeping top 48 coefficients out of 64

recon_img = zeros(N,N); % to store the reconstructed image

for i=1:8:N
    for j=1:8:N
        block_8x8 = img(i:i+7,j:j+7); % selecting a 8x8 block
        block_dct = dct2(block_8x8); % Finding the DCT of the selected
 block
        block_dct(7:8,1:8)=zeros(2,8); % knocking off the remaining 16
 coefficients
        recon_img(i:i+7,j:j+7)=idct2(block_dct); % taking IDCT of the
 remaining coefficients
    end
end

% Plot of original image and reconstructed image
figure;
subplot(1,2,1);
imshow(img);
title('Original Image - cameraman.tif');
subplot(1,2,2);
imshow(uint8(recon_img));
title('Reconstructed Image after keeping TOP 48 coefficients out of
 64');
sgtitle('19ucc023 - Mohit Akhouri');

% calculating compression ratio and mean square error
```

Figure 6.9 Part 1 of the code for observation 3 and 4

```
mse_case_1 = mse(img,uint8(recon_img)); % mean square error
 calculation
knocked_off_coeff_1 = 1024*(64-48); % calculating and storing the
 knocked off coefficients
p_case_1 = ((N*N - knocked_off_coeff_1)/(N*N)); % calculating
 compression ratio


% CASE 2 : Keeping top 32 coefficients out of 64

recon_img = zeros(N,N); % to store the reconstructed image

for i=1:8:N
    for j=1:8:N
        block_8x8 = img(i:i+7,j:j+7); % selecting a 8x8 block
        block_dct = dct2(block_8x8); % Finding the DCT of the selected
 block
        block_dct(5:8,1:8)=zeros(4,8); % knocking off the remaining 32
 coefficients
        recon_img(i:i+7,j:j+7)=idct2(block_dct); % taking IDCT of the
 remaining coefficients
    end
end

% Plot of original image and reconstructed image
figure;
subplot(1,2,1);
imshow(img);
title('Original Image - cameraman.tif');
subplot(1,2,2);
imshow(uint8(recon_img));
title('Reconstructed Image after keeping TOP 32 coefficients out of
 64');
sgtitle('19ucc023 - Mohit Akhouri');

% calculating compression ratio and mean square error
mse_case_2 = mse(img,uint8(recon_img)); % mean square error
 calculation
knocked_off_coeff_2 = 1024*(64-32); % calculating and storing the
 knocked off coefficients
p_case_2 = ((N*N - knocked_off_coeff_2)/(N*N)); % calculating
 compression ratio


% CASE 3 : Keeping top 16 coefficients out of 64

recon_img = zeros(N,N); % to store the reconstructed image

for i=1:8:N
    for j=1:8:N
        block_8x8 = img(i:i+7,j:j+7); % selecting a 8x8 block
        block_dct = dct2(block_8x8); % Finding the DCT of the selected
 block
```

Figure 6.10 Part 2 of the code for observation 3 and 4

```
        block_dct(3:8,1:8)=zeros(6,8); % knocking off the remaining 32
 coefficients
        recon_img(i:i+7,j:j+7)=idct2(block_dct); % taking IDCT of the
 remaining coefficients
    end
end

% Plot of original image and reconstructed image
figure;
subplot(1,2,1);
imshow(img);
title('Original Image - cameraman.tif');
subplot(1,2,2);
imshow(uint8(recon_img));
title('Reconstructed Image after keeping TOP 16 coefficients out of
 64');
sgtitle('19ucc023 - Mohit Akhouri');

% calculating compression ratio and mean square error
mse_case_3 = mse(img,uint8(recon_img)); % mean square error
 calculation
knocked_off_coeff_3 = 1024*(64-16); % calculating and storing the
 knocked off coefficients
p_case_3 = ((N*N - knocked_off_coeff_3)/(N*N)); % calculating
 compression ratio


% CASE 4 : Keeping top 8 coefficients out of 64

recon_img = zeros(N,N);

for i=1:8:N
    for j=1:8:N
        block_8x8 = img(i:i+7,j:j+7); % selecting a 8x8 block
        block_dct = dct2(block_8x8);  % Finding the DCT of the
 selected block
        block_dct(2:8,1:8)=zeros(7,8); % knocking off the remaining 56
 coefficients
        recon_img(i:i+7,j:j+7)=idct2(block_dct); % taking IDCT of the
 remaining coefficients
    end
end

% Plot of original image and reconstructed image
figure;
subplot(1,2,1);
imshow(img);
title('Original Image - cameraman.tif');
subplot(1,2,2);
imshow(uint8(recon_img));
title('Reconstructed Image after keeping TOP 8 coefficients out of
 64');
sgtitle('19ucc023 - Mohit Akhouri');
```

Figure 6.11 Part 3 of the code for observation 3 and 4

```
% calculating compression ratio and mean square error
mse_case_4 = mse(img,uint8(recon_img)); % mean square error
 calculation
knocked_off_coeff_4 = 1024*(64-8); % calculating and storing the
 knocked off coefficients
p_case_4 = ((N*N - knocked_off_coeff_4)/(N*N)); % calculating
 compression ratio

% Plot of Mean Square error vs. Compression Ratio

mse_array = zeros(1,4); % For storing different values of MSE
p_array = zeros(1,4); % For storing different values of compression
 ratio (p)

% storage of different values of MSE
mse_array(1) = mse_case_1;
mse_array(2) = mse_case_2;
mse_array(3) = mse_case_3;
mse_array(4) = mse_case_4;

% storage of different values of compression ratio (p)
p_array(1) = p_case_1;
p_array(2) = p_case_2;
p_array(3) = p_case_3;
p_array(4) = p_case_4;

% Plot of MSE vs. Compression ratio (p)
figure;
stem(p_array,mse_array,'Linewidth',1.5);
xlabel('Compression ratio (\rho) ->');
ylabel('Mean Square Error (\epsilon) ->');
title('19ucc023 - Mohit Akhouri','Plot of Mean Square Error (\epsilon)
 vs. Compression ratio (\rho)');
grid on;
```

Figure 6.12 Part 4 of the code for observation 3 and 4

Figure 6.13 Plot of Original Image and **Compressed Image** obtained by keeping **top 48 / 64 coefficients**

Figure 6.14 Plot of Original Image and **Compressed Image** obtained by keeping **top 32 / 64 coefficients**

Figure 6.15 Plot of Original Image and **Compressed Image** obtained by keeping **top 16 / 64 coefficients**

Figure 6.16 Plot of Original Image and **Compressed Image** obtained by keeping **top 8 / 64 coefficients**

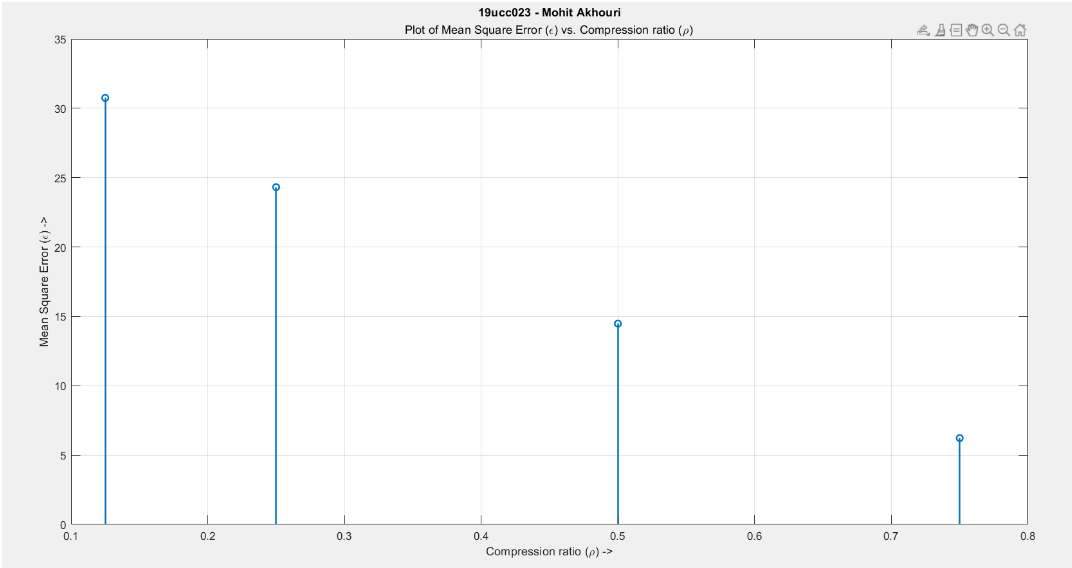

Figure 6.17 Graph of Mean Square Error (ϵ) vs. Compression Ratio (ρ)

6.4.4 Simulink based Image Compression :

```
% 19ucc023
% Mohit Akhouri
% Experiment 6 - Observation 5

% ALGORITHM :
% This code will use the model "Simulink_Observation_5" and calculate
 the
% DCT of the received image signal for four different cases by
 knocking off
% 16,32,48 and 56 coeffcients

sim('Simulink_Observation_5'); % calling the simulink model

N = 256; % Total size of image ( 256 rows with 256 columns )
output_case_1 = zeros(N,N); % Output for case 1 = top 48/64
 coefficients
output_case_2 = zeros(N,N); % Output for case 2 = top 32/64
 coefficients
output_case_3 = zeros(N,N); % Output for case 3 = top 16/64
 coefficients
output_case_4 = zeros(N,N); % Output for case 4 = top 8/64
 coefficients

[output_case_1,output_case_2,output_case_3,output_case_4] =
 DCT_2D(out.y1.data,out.y2.data,out.y3.data,out.y4.data);

% Plot of Figures for case 1 and case 2
figure;
subplot(1,2,1);
imshow(uint8(output_case_1));
title('Compressed Image obtained via SIMULINK MODEL for Case 1 - top
 48/64 coefficients kept');
subplot(1,2,2);
imshow(uint8(output_case_2));
title('Compressed Image obtained via SIMULINK MODEL for Case 2 - top
 32/64 coefficients kept');
sgtitle('19ucc023 - Mohit Akhouri');

% Plot of Figures for case 3 and case 4
figure;
subplot(1,2,1);
imshow(uint8(output_case_3));
title('Compressed Image obtained via SIMULINK MODEL for Case 3 - top
 16/64 coefficients kept');
subplot(1,2,2);
imshow(uint8(output_case_4));
title('Compressed Image obtained via SIMULINK MODEL for Case 4 - top
 8/64 coefficients kept');
sgtitle('19ucc023 - Mohit Akhouri');
```

Figure 6.18 Code for the observation 5

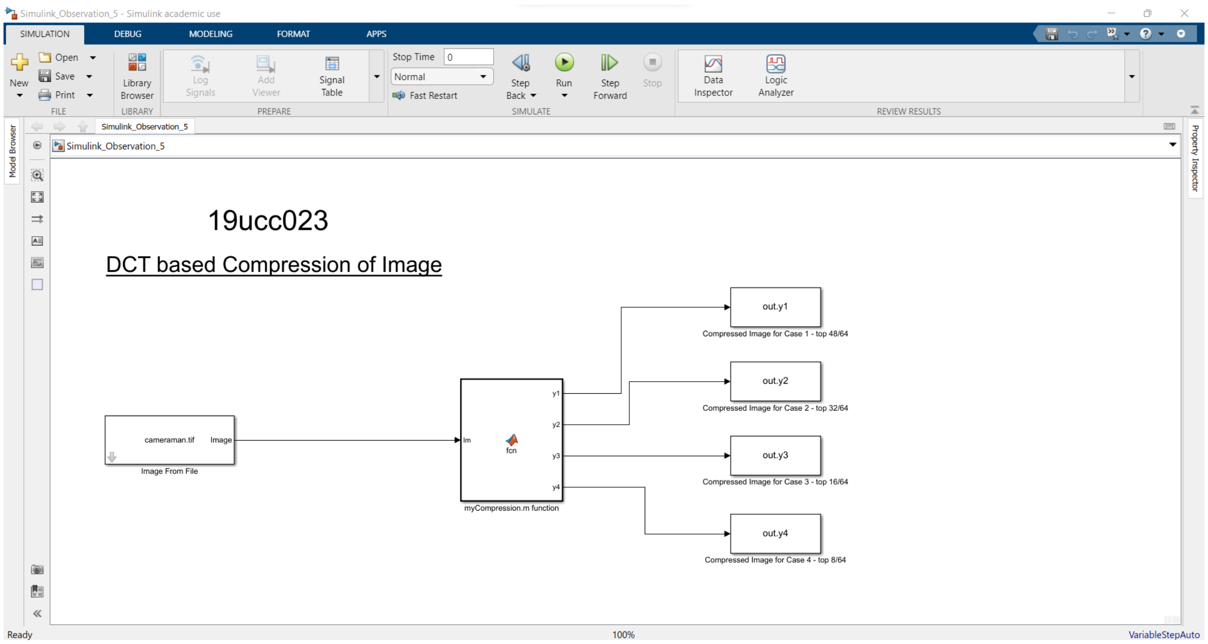

Figure 6.19 Simulink Model used for Image Compression

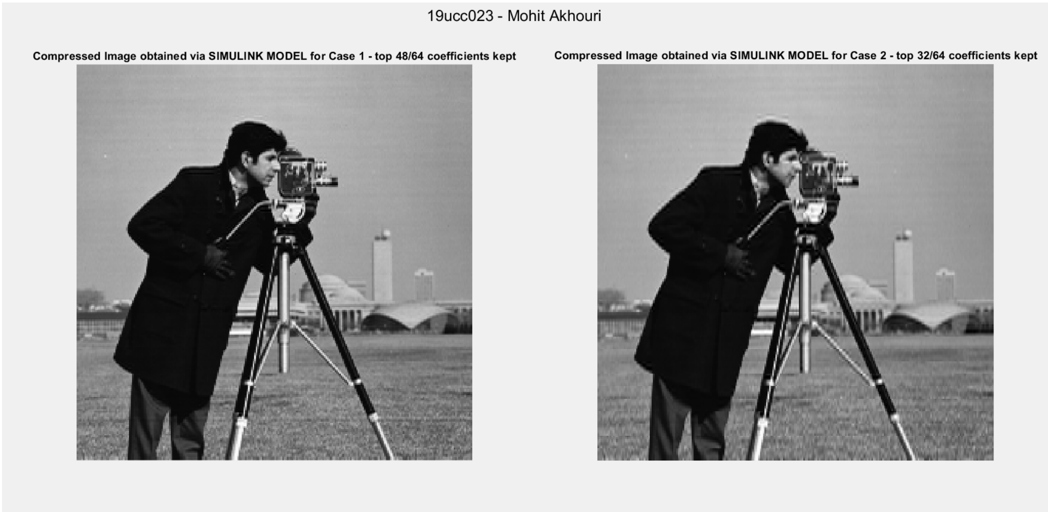

Figure 6.20 Plots of **Compressed Images** obtained via Simulink Model (for Case 1 and Case 2)

Figure 6.21 Plots of **Compressed Images** obtained via Simulink Model (for Case 3 and Case 4)

6.4.5 Functions used in main codes for DCT and Image Compression :

6.4.5.1 myCompression.m function code :

```
function [ImF] = myCompression(Im)

% 19ucc023
% Mohit Akhouri

% ALGORITHM :
% This function will compute the DCT of given image signal "Im"
% It uses the expression of "sum of cosines" to calculate the DCT

N = size(Im,1); % Size of the image (256x256)

ImF = zeros(N,N); % to store the final computed DCT

Beta = zeros(1,N); % variable that takes value 1/root(2) if i=1 else
 takes value 1

% algorithm for calculation of different values of variable "beta" is
 given
% belo , here a loop is run and correspondingly the value is set.
for i=1:N
    if i==1
        Beta(i)=1/sqrt(2);
    else
        Beta(i)=1.0;
    end
end

% Main ALGORITHM for the calculation of DCT is as follows :
for k=1:N
    for l=1:N
        sum = 0.0;
        for i=1:N
            for j=1:N
                sum = sum + (Im(i,j)*cos((pi*((2.0*(j-1)+1))*(k-1))/
(2.0*N))*cos((pi*((2.0*(i-1))+1)*(l-1))/(2.0*N)));
            end
        end
        sum = sum * (1.0/sqrt(2.0*N)) * Beta(k) * Beta(l) ;
    end
end

end
```

Published with MATLAB® R2020b

Figure 6.22 myCompression.m function used to calculate the **DCT** of the given input image

6.4.5.2 DCT_2D.m function code :

```
function [y1,y2,y3,y4] = DCT_2D(Im1,Im2,Im3,Im4)

% 19ucc023
% Mohit Akhouri

% ALGORITHM : This function will calculate the DCT of image img for
 various
% cases : knocking off 16,32,48 and 56 coefficients ( higher frequency
% terms )


N = size(Im1,1); % size of image (256x256)
img = Im1; % temporary variable to store the image

y1 = zeros(N,N); % output variable for case 1
y2 = zeros(N,N); % output variable for case 2
y3 = zeros(N,N); % output variable for case 3
y4 = zeros(N,N); % output variable for case 4

% ALGORITHM for compression is as follows :

recon_img = zeros(N,N); % to store the reconstructed image

for i=1:8:N
    for j=1:8:N
        block_8x8 = img(i:i+7,j:j+7); % selecting a 8x8 block
        block_dct = dct2(block_8x8); % Finding the DCT of the selected
 block
        block_dct(7:8,1:8)=zeros(2,8); % knocking off the remaining 16
 coefficients
        recon_img(i:i+7,j:j+7)=idct2(block_dct); % taking IDCT of the
 remaining coefficients
    end
end

y1=recon_img;

% ALGORITHM for compression is as follows :

recon_img = zeros(N,N); % to store the reconstructed image

for i=1:8:N
    for j=1:8:N
        block_8x8 = img(i:i+7,j:j+7); % selecting a 8x8 block
        block_dct = dct2(block_8x8); % Finding the DCT of the selected
 block
        block_dct(5:8,1:8)=zeros(4,8); % knocking off the remaining 32
 coefficients
        recon_img(i:i+7,j:j+7)=idct2(block_dct); % taking IDCT of the
 remaining coefficients
    end
end
```

Figure 6.23 Part 1 of the code for the function DCT_2D.m used for **Image Compression**

```
y2 = recon_img;

% ALGORITHM for compression is as follows :

recon_img = zeros(N,N); % to store the reconstructed image

for i=1:8:N
    for j=1:8:N
        block_8x8 = img(i:i+7,j:j+7); % selecting a 8x8 block
        block_dct = dct2(block_8x8); % Finding the DCT of the selected
 block
        block_dct(3:8,1:8)=zeros(6,8); % knocking off the remaining 32
 coefficients
        recon_img(i:i+7,j:j+7)=idct2(block_dct); % taking IDCT of the
 remaining coefficients
    end
end

y3 = recon_img;

% ALGORITHM for compression is as follows :

recon_img = zeros(N,N);

for i=1:8:N
    for j=1:8:N
        block_8x8 = img(i:i+7,j:j+7); % selecting a 8x8 block
        block_dct = dct2(block_8x8);  % Finding the DCT of the
 selected block
        block_dct(2:8,1:8)=zeros(7,8); % knocking off the remaining 56
 coefficients
        recon_img(i:i+7,j:j+7)=idct2(block_dct); % taking IDCT of the
 remaining coefficients
    end
end

y4 = recon_img;

end
```

Published with MATLAB® R2020b

Figure 6.24 Part 2 of the code for the function DCT_2D.m used for **Image Compression**

6.5 Conclusion

In this experiment , we learnt the concepts of **Discrete Cosine Transform** , **Inverse DCT** and **Transform Based Lossy Image Compression** of Digital Signal Processing. We learnt about DCT matrix and how to compute the DCT of any given image signal. We learnt the significance of DCT in **Image Compression**. We also observed the compression of image for various cases - by keeping top 48,32,16 and 8 coeffcients out of 64 and rejecting the rest. We observed the **Artifact effects** due to lossy compression. We also learnt about the relation between **Compression Ratio** (ρ) and **Mean square error** (ϵ). We also plotted the graph of Mean square error vs. Compression ratio for different values of Compression ratio. We also implemented the Image Compression in Simulink and compared the results obtained from MATLAB coding.