Class 5: solving the Phase II-c of the class proyect.

EE7790 Special Topics
VISUAL SIGNAL PROCESSING AND COMMUNICATIONS

SP11

Prof.: Dr. Luis M. Vicente

Project

Phase II: Baseline Image Encoding Decoding System using DCT

04/11 /2011

Your name here
Student number: xxxxxxx
E-mail: your@email.here

Your name here# xxxxxxx
your@emailhere.com

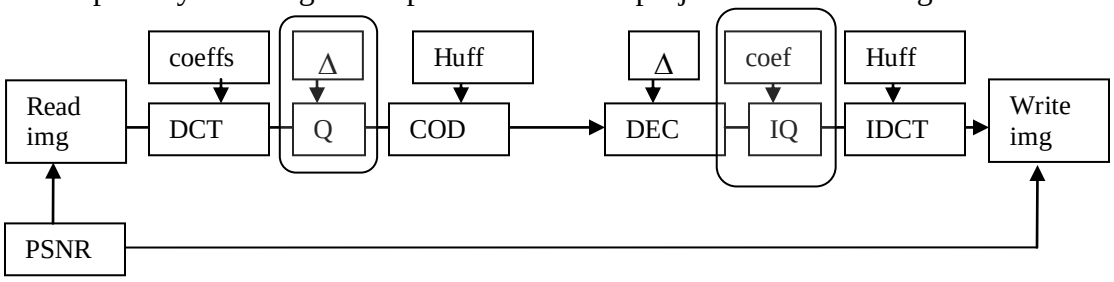
Objective:
Implement a baseline image encoding and decoding system with DCT.

Steps to implement:

1. Write functions to read and write image data (done in class3).
2. Write a function to measure PSNR between two images. The images are stored in files or in the memory (done in class3).
3. Write functions for DCT and IDCT at block and image levels.
4. **Write functions for Quantization and inverse quantization at block and image levels.**
5. Training:
   a. DC: 10-bit binary representation. (No prediction!)
   b. AC: (run, size) + magnitude representation. Run: [0 15], size: [0 10].
   c. Collect statistics on (run, size), and design a Huffman code table
6. Encoding: look up the Huffman table; count the number of bits of encoding.
7. Plot the rate-distortion curve by varying the quantization step size.

Methodology

The complete system diagram implemented in this project is the following:



I will work the system diagram by parts:

1. Read and write blocks.
2. PSNR block
3. DCT/IDCT blocks
4. **Quantization/Inverse Quantization blocks**
5. Image Coding/Image Decoding blocks

4. Quantization/Inverse Quantization blocks

The quantization/inverse quantization blocks were realized with the same strategy at the DCT/IDCT blocks.

First, is implemented the function *fblockq.m* to calculate the quantization of an 8x8 matrix. I used a "perceptual weighting table" (*PPT)* [1][2] stored as a look up table (using *mcreateqcoeff.m.* Matlab script file). The function was tested using a reference block written at the class handout [1] (page 9) and gave me the same results (*mtestblockq_iq.m* script file).

The quantization/inverse quantization implemented satisfies the following equations [1]:

$$qDCT(i,j) = round\left(\frac{DCT(i,j)}{\beta\left(PPT(i,j)\right)}\right)$$

$$iqDCT(i,j) = qDCT(i,j)\beta\left(PPT(i,j)\right)$$

β is que quantizer quality factor. The bigger the factor, the coarser will be the quantization, hence the PSNR will be smaller.

Secondly, is implemented a function *fimagq.m* that performs the quantization of a 512x512 matrix by calling *fblockq.m* for each 8x8 sub-blocks.

The dual functions *fblockiq.m,* and *fimagiq.m* were implemented to do the inverse operation at their respective level.

The functions *fimagq / fimagiq* were tested with the *mprojII.m* Matlab script, and verify that the image after DCT and IDCT processes was reconstructed with different resolutions when varying the β parameter (using the *fpsnr.m* function).

The code is explained next:

```
%fimagq
%Function to quantize an image
%calls custom functions: fblockq

function [qimage]=fimagq(iimage,beta)

[rowb,colb] = size(iimage);

%load quantization look up table
load qcoef

%I have to divide the image in 8x8 blocks
for rowblock=1:8:rowb
    for colblock=1:8:colb
        %disp(['row: ', num2str(rowblock),' col: ', num2str(colblock)]);
        %select a block
        iblock=iimage(rowblock:rowblock+7,colblock:colblock+7);
        %perform the block quantization
        oblock=fblockq(iblock,beta,qcoeff);
        %store in the output Quantized image
        qimage(rowblock:rowblock+7,colblock:colblock+7)=oblock;
    end
end
```

```
%fblockq
%Function to quantize the dct of a 8*8 block
%calls custom data: qcoef.mat

function [qblock]=fblockq(iblock,beta,qcoeff)

%multiply the scaling parameter
qij = beta.*qcoeff;

qblock = round(iblock./qij);
```

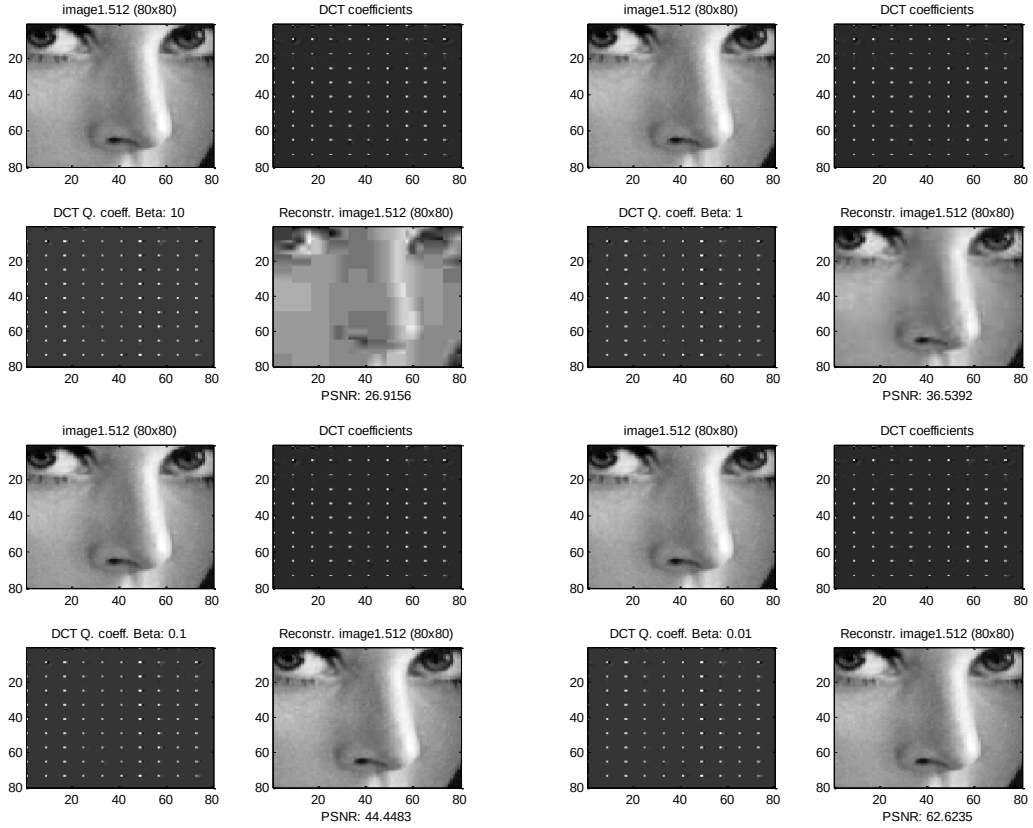
```
%fimagiq
%Function to create the dct of a ???x??? image
%calls custom functions: fblockiq

function [iqimage]=fimagiq(qimage,beta,qcoeff)

[rowb,colb] = size(qimage);

%load quantization look up table
load qcoef

%I have to divide the image in 8x8 blocks
for rowblock=1:8:rowb
    for colblock=1:8:colb
        %disp(['row: ', num2str(rowblock),' col: ', num2str(colblock)]);
        %select a block
        qiblock=qimage(rowblock:rowblock+7,colblock:colblock+7);
        %perform the block Inverse quantization
        oblock=fblockiq(qiblock,beta,qcoeff);
        %store in the output Quantized image
        iqimage(rowblock:rowblock+7,colblock:colblock+7)=oblock;
    end
```

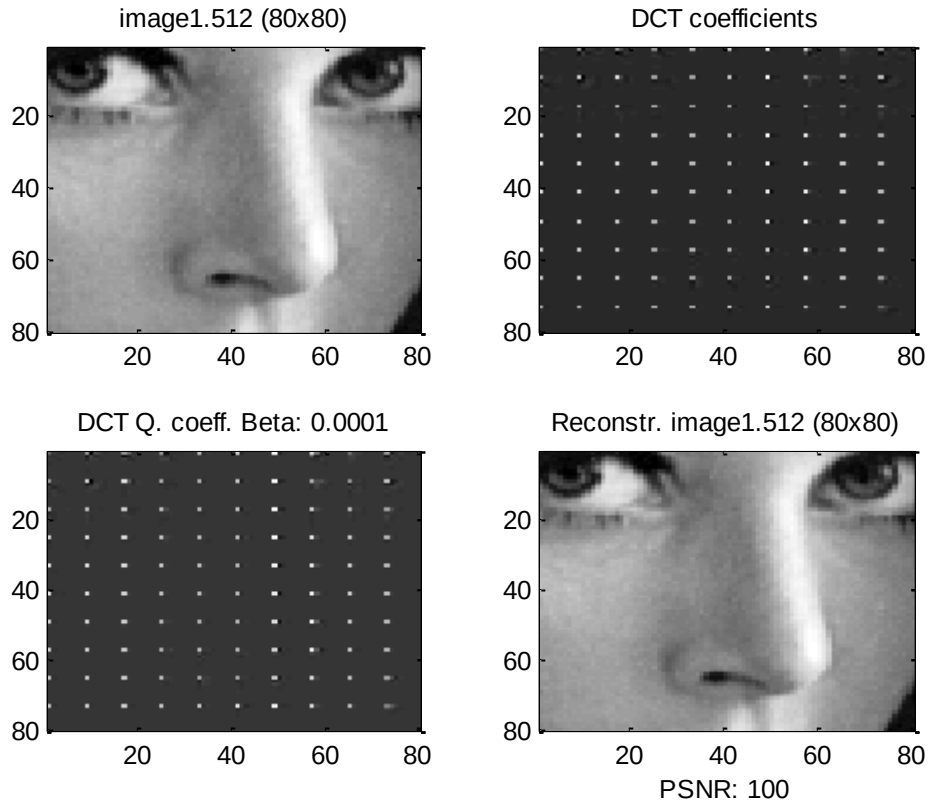
```
end
```

```
%fblockiq
%Function to inverse quantize the dct of a 8*8 block
%calls custom data: qcoef.mat

function [oblockiq]=fblockiq(Qxij,beta,qcoeff)

%load quantization look up table
% load qcoef

%multiply the scaling parameter
qij = beta.*qcoeff;

oblockiq = Qxij.*qij;
```

Because the processing of a 512x512 took too much time to process (back in 2005 times), I tested all blocks with a sub image of 80x80 pixels (100 blocks of 8x8).

We can see the results in next 80x80 images:

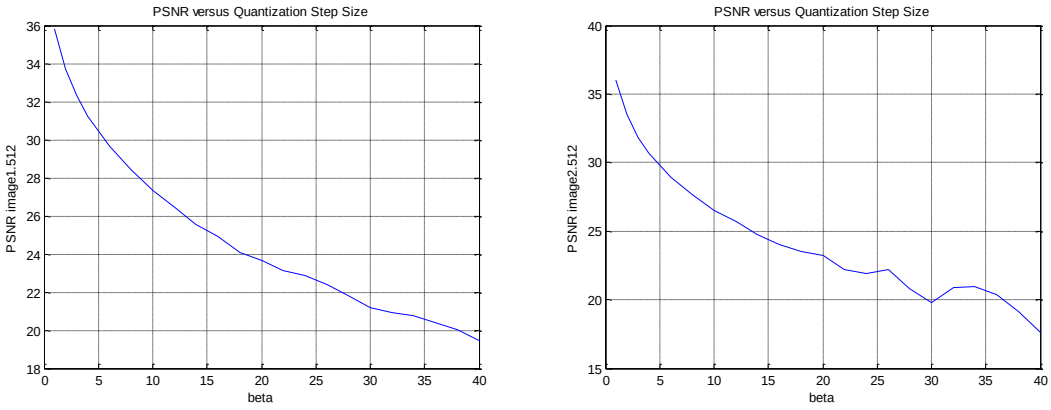


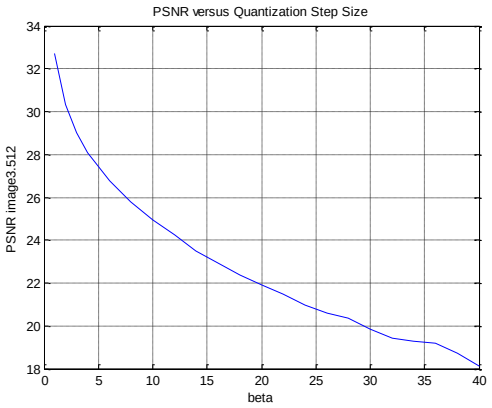


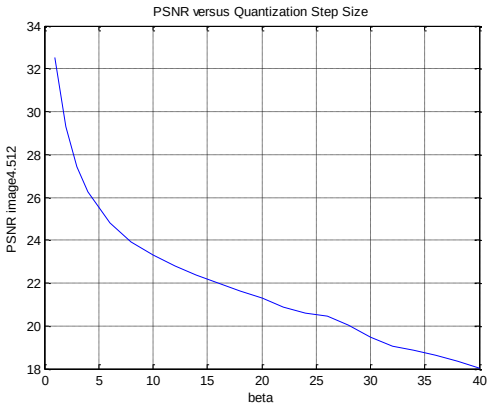
This part answer Step to implement #4

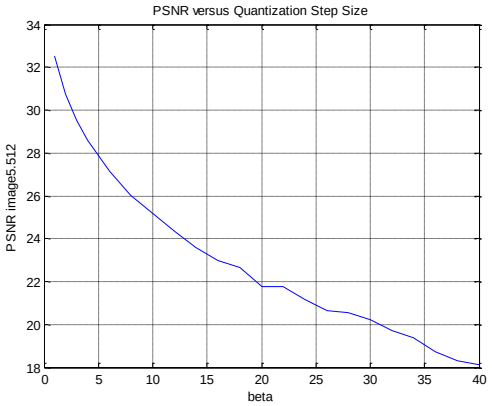
Rate Distortion Curve

Now we would like to know the Rate Distortion Curve for different values of the Quantizer quality factor: β. We run the algorithm for the 5 images and found the result shown in next figures and table:




PSNR versus Quantization Step Size
PSNR image3.512
beta
34
32
30
28
26
24
22
20
18
0
5
10
15
20
25
30
35
40


PSNR versus Quantization Step Size
PSNR image4.512
beta
34
32
30
28
26
24
22
20
18
0
5
10
15
20
25
30
35
40


PSNR versus Quantization Step Size
PSNR image5.512
beta
34
32
30
28
26
24
22
20
18
0
5
10
15
20
25
30
35
40

β	PSNR i1	PSNR i2	PSNR i3	PSNR i4	PSNR i5
1	**36**	**36**	**33**	**33**	**33**
2	34	34	30	29	31
3	32	32	29	27	30
4	31	31	28	26	29
6	**30**	**29**	**27**	**25**	**27**
8	28	28	26	24	26
10	27	27	25	23	25
12	26	26	24	23	24
14	26	25	24	22	24
16	25	24	23	22	23
18	24	23	22	22	23
20	24	23	22	21	22
22	23	22	21	21	22
24	23	22	21	21	21
26	22	22	21	20	21
28	22	21	20	20	21
30	21	20	20	19	20
32	21	21	19	19	20
34	21	21	19	19	19
36	20	20	19	19	19
38	20	19	19	18	18
40	19	18	18	18	18

We can see in next figures section than a PSNR of ~36 is acceptable ($\beta = 1$). For values of PNSR less than 30 ($\beta \geq 5$) the image degradation is noticeable.

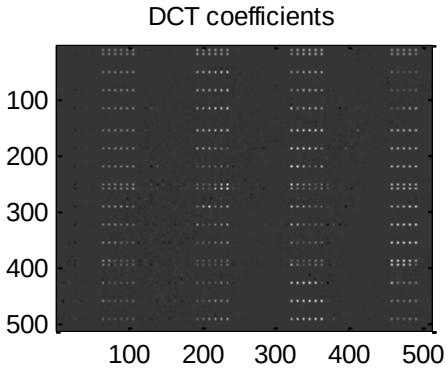
CONCLUSION

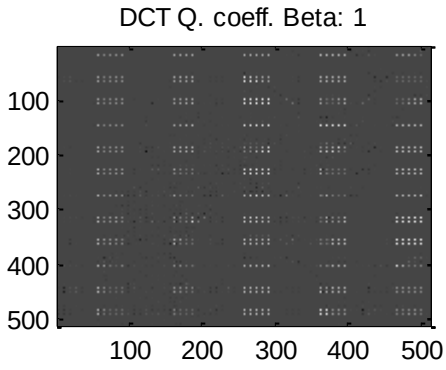
This project has been a very challenging project. The key to the success has been to work on blocks and verify that each block worked as designed by using separate test scripts. The DCT and Quantization blocks did not give me too much problem. However, the encoding blocks have been implemented after serious brainstorms. The choice of format to store the symbols and the different problem arisen from the zero run, size and amplitude position representation took me more than once to dead end solutions. Also, some serious clean up and management of look up tables and variables were needed to improve the speed of the algorithm.

In summary I enjoyed doing the project; which has not been short of stressful parts; but the joy of overcoming them was a rewarding experience.

FIGURES

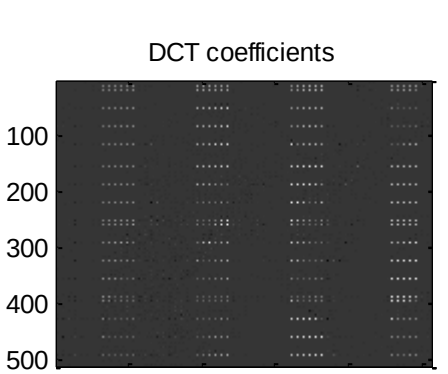

image1.512 (512x512)
100
200
300
400
500
100 200 300 400 500

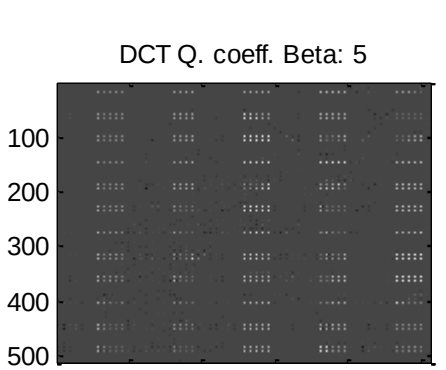

DCT coefficients
100
200
300
400
500
100 200 300 400 500


DCT Q. coeff. Beta: 1
100
200
300
400
500
100 200 300 400 500


Reconstr. image1.512 (512x512)
100
200
300
400
500
100 200 300 400 500
PSNR: 35.805


image1.512 (512x512)
100
200
300
400
500
100 200 300 400 500

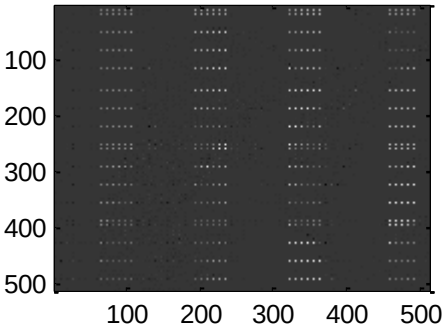

DCT coefficients
100
200
300
400
500
100 200 300 400 500


DCT Q. coeff. Beta: 5
100
200
300
400
500
100 200 300 400 500

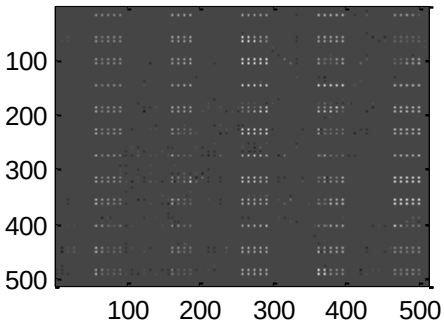

Reconstr. image1.512 (512x512)
100
200
300
400
500
100 200 300 400 500
PSNR: 30.3969

image1.512 (512x512)

DCT coefficients



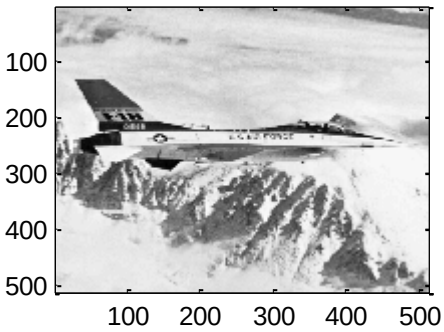
DCT Q. coeff. Beta: 10



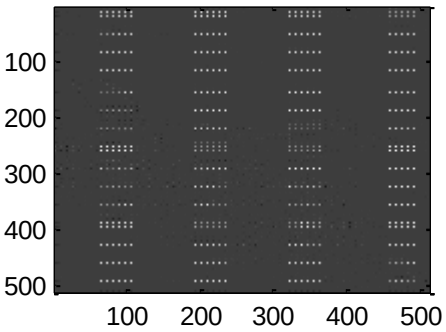
Reconstr. image1.512 (512x512)

PSNR: 27.3179

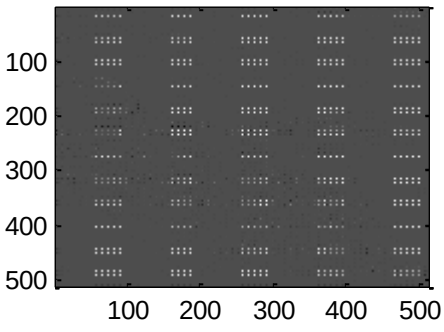
image2.512 (512x512)



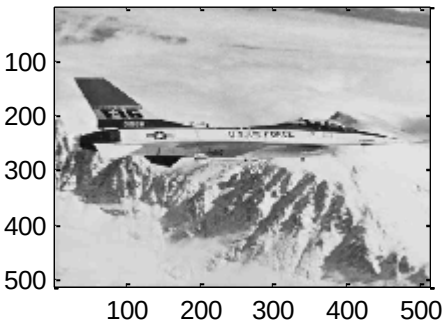
DCT coefficients



DCT Q. coeff. Beta: 1



Reconstr. image2.512 (512x512)



PSNR: 36.0185

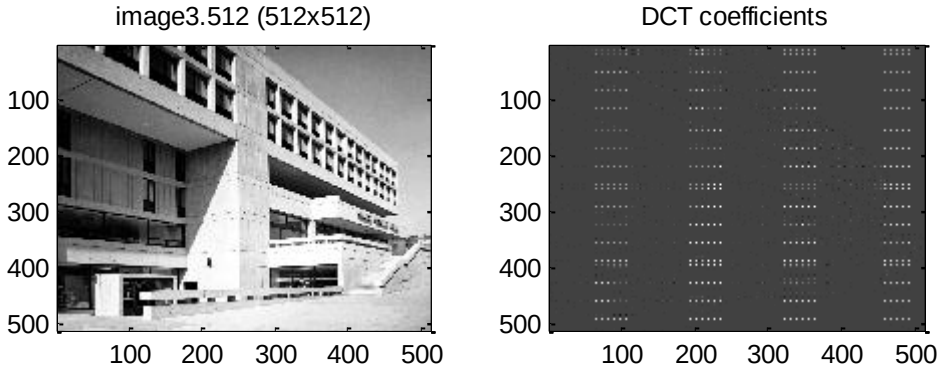


image3.512 (512x512)
DCT coefficients
DCT Q. coeff. Beta: 1
Reconstr. image3.512 (512x512)
PSNR: 32.7245
image4.512 (512x512)
DCT coefficients
DCT Q. coeff. Beta: 1
Reconstr. image4.512 (512x512)
PSNR: 32.5445

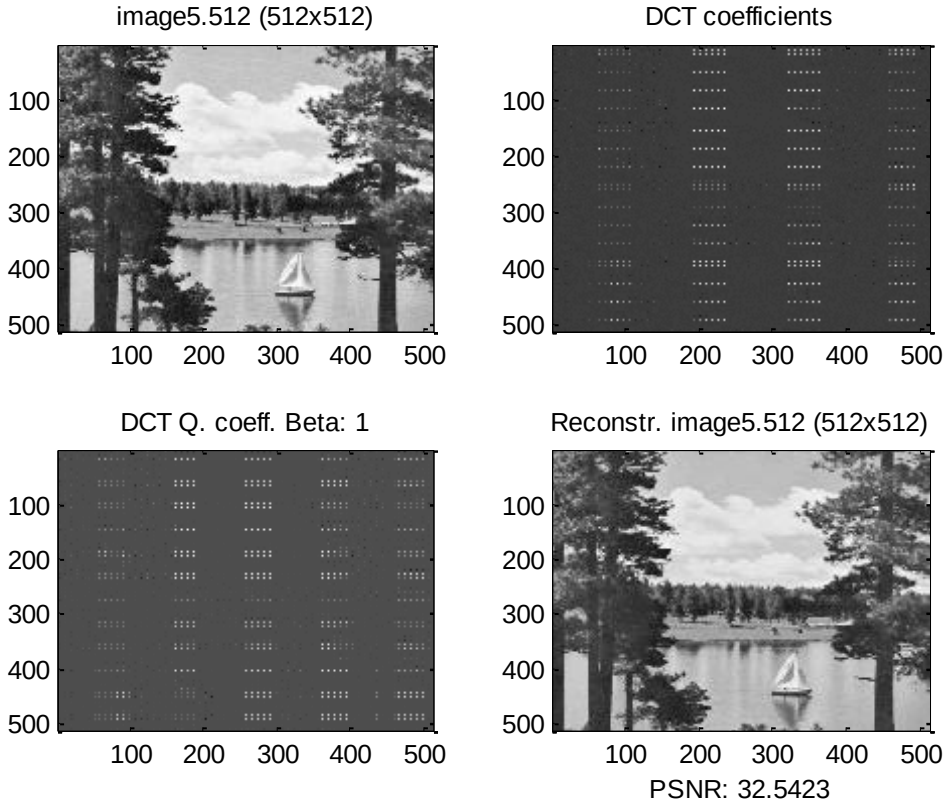


image5.512 (512x512)
DCT coefficients
DCT Q. coeff. Beta: 1
Reconstr. image5.512 (512x512)
100
200
300
400
500
PSNR: 32.5423

APPENDIX 1 (Source Code)

The main script is called *mprojII.m.* This script read an image, does the DCT, then the quantization, the zigzag scan for each block, and encode the image giving us the binary code based in a Huffman code. Then do the inverse process to reconstruct the image. The code writes out the PSNR as a measure of the quality of reconstruction. It also plots the original and reconstructed image, as well as the DCT values before and after quantization.

It calls the functions *freadimg fimagdct fimagq fimagzzag fencodeimage fimagizzag fimagiq fimagidct fpsnr fwriteimg*. These functions also call several functions which operate down to the block and vector level.

To use this script, please note that it needs a large quantity of custom functions; therefore, is advisable that the user unzip all the files at the **mprojII.zip** file in the Matlab working directory beforehand.

Before run the script (by typing at the Matlab working directory *mprojII*), open the script with the Matlab editor and select the image to read, and set the beta parameter that suits your needs.

The source code for the main script is the following:

```
%mprojII
%Luis M Vicente 945 995  started 4/2/2005
%script to read files for phase II
%calls custom functions:  freadimg fimagdct fimagq fimagzzag fencodeimage
% fimagizzag fimagiq fimagidct fpsnr fwriteimg
%calls data: image5.512

clear all, close all, clc
disp(['//////////////////////Project II Main Program//////////////////////'])
%Name of the image to analyze and the reconstructed image to write to
strim = 'image1.512';
strom = ['rec_',strim];
imsize = 512;
subi = imsize;
isigd = freadimg(strim,imsize);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%create a sub image to lesser the processing time COMMENT OUT IF NOT USED
%This one is good for image5.512 to see the boat
% subi = 160;
% xi=170;
% yi=190;
% isigd = isigd(xi:xi+subi-1,yi:yi+subi-1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%create a sub image to lesser the processing time COMMENT OUT IF NOT USED
%This one is good for image1.512 to see the face
% subi = 80;
% xi=260;
% yi=260;
% isigd = isigd(xi:xi+subi-1,yi:yi+subi-1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Do the DCT of the whole image
tic;
DCTsigd = fimagdct(isigd);
disp(['Progress message: DCT done']);
toc;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Do the quantization of the whole DCT image
beta =10^0;
disp(['Beta: ', num2str(beta)]);

[qimage]=fimagq(DCTsigd,beta);
disp(['Progress message: Q done']);
toc;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%% ENCODING PART %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Do the zig zag of the whole quantized image
qzzimage=fimagzzag(qimage);
disp(['Progress message: ZZ done']);
toc;

%Save the zigzag-d of the whole quantized image to check its size
save qzzimage qzzimage

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Do the encoding of the zig zag of the whole quantized image
[allsymbl,lsymbl]=fencodeimage(qzzimage);
disp(['Progress message: Encode Image done']);
toc;

sumsymbl = sum(lsymbl);
sumsymbl2 = length(allsymbl);
disp(['Size in bits of the Encoded Image: ',num2str(sumsymbl)]);

%Save the encoded bits for future use.
strbin = ['bin',strim(1:6)];
save(strbin,'allsymbl', 'lsymbl')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Do the decoding of the allsymbl binary string
[qzzimagedecoded]=fdecodeimage(allsymbl,lsymbl);
disp(['Progress message: Decode Image done']);
toc;

%test difference between qzzimage and qzzimagedecoded
if(max(max(abs(qzzimage-qzzimagedecoded))))
    disp('ERROR @ mprojII: image was not encoded/decoded right')
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Do the inverse zig zag of the whole quantized image
qizzimage=fimagizzag(qzzimagedecoded);
disp(['Progress message: IZ done']);
toc;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Do the Inverse quantization of the whole image
[Iqimage]=fimagiq(qizzimage,beta);
disp(['Progress message: IQ done']);
toc;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Do the IDCT of the whole image
rsigd = fimagidct(Iqimage);
disp(['Progress message: IDCT done']);
toc;

%compare the images
PSNR = fpsnr(isigd, rsigd);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Write the image in a new file for later comparisons
fwriteimg(strom,rsigd);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
disp('If you want to see the images, wait one second');
pause(1)
%PLOT SECTION
%Plot input image
figure
subplot(2,2,1)
imagesc(isigd);
colormap('gray');zoom on;
title([strim, ' (', num2str(subi),'x',num2str(subi),')']);

%Plot DCT image
subplot(2,2,2)
imagesc(DCTsigd);
colormap('gray');zoom on;
title(['DCT coefficients'])

%Plot Quantized DCT image
subplot(2,2,3)
imagesc(qimage);
colormap('gray');zoom on;
title(['DCT Q. coeff. Beta: ',num2str(beta)])

%Plot reconstructed image
subplot(2,2,4)
imagesc(rsigd);
colormap('gray');zoom on;
title(['Reconstr. ',strim, ' (', num2str(subi),'x',...
    num2str(subi),')'])
xlabel(['PSNR: ',num2str(PSNR)])

disp(['//////////////////////Project II End of Main Program//////////////////////'])
```

Next, the functions used in this script:

```
%freadimg
%Function to read an image

function [isigd]=freadimg(strim,imsize)
fid = fopen(strim,'rb');
isigc = fread(fid,[imsize,imsize],'uchar');
isigd = double(isigc)';
fclose(fid);
```

```
%fimagdct
%Function to create the dct of a 512x512 image
%calls custom functions: fblockdct

function [oimage]=fimagdct(iimage)

[rowb,colb] = size(iimage);

%load the dct coefficients stored in mat file
load dctcoeff

%I have to divide the image in 8x8 blocks
for rowblock=1:8:rowb
    for colblock=1:8:colb
        %disp(['row: ', num2str(rowblock),' col: ', num2str(colblock)]);
        %select a block
        iblock=iimage(rowblock:rowblock+7,colblock:colblock+7);
        %performthe dct
        oblock=fblockdct(iblock,lut);
        %store in the output DCT image
        oimage(rowblock:rowblock+7,colblock:colblock+7)=oblock;
    end
end
```

```
%fblockdct
%Function to create the dct of a 8*8 block
%calls custom functions: fdct

function [oblock]=fblockdct(iblock,lut)

[rowb,colb] = size(iblock);

%For each row make the dct
for ir = 1:rowb
    xsignal=iblock(ir,:);
    Stemp(ir,:)=fdct(xsignal,lut);
end

%For each column make the dct
for ic = 1:colb
    %transpose to create a row vector
    xsignal=Stemp(:,ic)';
    %transpose the output of fdct to store in a column
    oblock(:,ic)=fdct(xsignal,lut)';
end
```

```
%fdct
%Function to calculate the DCT of a 8x8 vector
%calls data: dctcoeff.mat

function [S]=fdct(xsignal,lut)

% %load the dct coefficients stored in mat file
% load dctcoeff
slut = size(lut);
n = slut(1,1);

%remember in Matlab indexes start at 1
C = ones(1,n);
C(1) = 2^(-1/2);

%Find the DCT of an input signal
for u=0:n-1
    uin=u+1;

    %matrix multiplication of the signal with the DCT coefficients
    mmsdct = xsignal*lut(uin,:)';

    %DCT coefficient
    S(uin)=sqrt(2/n)*C(uin)*mmsdct;
end
```

```
%fimagq
%Function to quantize an image
%calls custom functions: fblockq

function [qimage]=fimagq(iimage,beta)

[rowb,colb] = size(iimage);

%load quantization look up table
load qcoef

%I have to divide the image in 8x8 blocks
for rowblock=1:8:rowb
    for colblock=1:8:colb
        %disp(['row: ', num2str(rowblock),' col: ', num2str(colblock)]);
        %select a block
        iblock=iimage(rowblock:rowblock+7,colblock:colblock+7);
        %perform the block quantization
        oblock=fblockq(iblock,beta,qcoeff);
        %store in the output Quantized image
```

```
        qimage(rowblock:rowblock+7,colblock:colblock+7)=oblock;
    end
end
```

```
%fblockq
%Function to quantize the dct of a 8*8 block
%calls custom data: qcoef.mat

function [qblock]=fblockq(iblock,beta,qcoeff)

%load quantization look up table
% load qcoef

%multiply the scaling parameter
qij = beta.*qcoeff;

qblock = round(iblock./qij);
```

```
%fimagzzag
%Function to create the zigzag of a ???x??? image
%calls custom functions: fblockzigzag

function [zzimage]=fimagzzag(iimage)

[rowb,colb] = size(iimage);

%I have to divide the image in 8x8 blocks
zzrow = 0;
for rowblock=1:8:rowb
    for colblock=1:8:colb
        zzrow = zzrow +1;
        %disp(['row: ', num2str(rowblock),' col: ', num2str(colblock)]);
        %select a block
        iblock=iimage(rowblock:rowblock+7,colblock:colblock+7);
        %perform the block zig zag
        azz = fblockzigzag(iblock);
        %store in the zig zagged vector as a row of a big matrix
        zzimage(zzrow,:)=azz';
    end
end
```

```
%fblockzigzag
%Function to create a zigzag scan of an 8x8 block image
%the output is a column vector

function [azz] = fblockzigzag(a)
azz = [
    a(1,1) a(1,2) ...
    a(2,1) a(3,1) a(2,2) a(1,3) a(1,4) a(2,3) a(3,2)...
    a(4,1) a(5,1) a(4,2) a(3,3) a(2,4) a(1,5) a(1,6) a(2,5) a(3,4) a(4,3)...
    a(5,2) a(6,1) a(7,1) a(6,2) a(5,3) a(4,4) a(3,5) a(2,6) a(1,7) a(1,8)...
    a(2,7) a(3,6) a(4,5) a(5,4) a(6,3) a(7,2) a(8,1) a(8,2) a(7,3)...
    a(6,4) a(5,5) a(4,6) a(3,7) a(2,8) a(3,8) a(4,7) a(5,6) a(6,5) a(7,4)...
    a(8,3) a(8,4) a(7,5) a(6,6) a(5,7) a(4,8) a(5,8) a(6,7) a(7,6) a(8,5)...
    a(8,6) a(7,7) a(6,8) a(7,8) a(8,7) a(8,8)];
azz=azz';
```

```
%fencodeimage.m
%this function gets all the rows from a zigzag image and find out the
%coded binary string based on a Huffman code. it also gives out the size of
%each variablelength code string per each zigzagged row.
%Calls functions: fencodeblock, facmtobinary

function [allsymbl,lsymbl]=fencodeimage(qzzimage)

%AC is stored with its symol (z,s) by look up table and then the amplitude
load symblutc
```

```
load bec;


[rowizz,colizz] = size(qzzimage);

%Now for each row of qzzimage encode it.
totallength = 0;
allsymbl = [];
for irow = 1:rowizz
    %Find the run level representation of each zig zag block
    acm = fencodeblock(qzzimage(irow,:)',bec);

    %create binary string
    symbl = facmtobinary(acm,symblutc);

    %concatenate the binary code
    allsymbl = [allsymbl symbl];

    %Store its size in a vector for when we wanted to recover the acm
    lsymbl(irow)=length(symbl);
    totallength =totallength +lsymbl(irow);
    %disp(['Encoded row: ',num2str(irow)]);
end
```

```
%%function to encode a block
%this function encode a 8x8 block
%calls functions: findintdccode

function [acm] = fencodeblock(a,bec);

%read DC coefficient
inda = 1;
dc = a(inda,1); %one column vector

%find int code custom function
[r,c]=findintdccode(dc,bec);

%concatenate in to the code string
acm = [r,c];

%Encode the AC values.
%read zeros until we get a nonzero value
izrun = 2;
inda = 2;
while(inda<=64)
    zerorun = 0;
    while(a(inda,1)==0 && inda<64 && zerorun<15)
        inda=inda+1;
        zerorun = zerorun+1;
    end
    izrun = izrun +1;
    %Store zerorun and ENCODED number
    [ra,ca]=findintdccode(a(inda,1),bec);
    acm(izrun) = zerorun;
    acm(izrun+1) = ra;
    acm(izrun+2) = ca;
    izrun = izrun +2;
    inda=inda+1;
end
```

```
%findintdccode
%function to find the dc code from an amplitude
%calls data: bec.mat


function [r,c]=findintdccode(amp,bec)

%account for amplitude zero
if amp==0
    r=0;
```

```
    c=0;
else
    [r,c]=find(bec==amp);
    if(min(size(r))==0)
        disp(['Error, amplitude ',num2str(amp),' Not coded']);
    end
end
```

```
%%facmtobinary
%this function gets the acm vector (from fencodeblock) and give us a vlc
%binary string.
%This new version adds the EOB symbol.
%call functions:

function [symbl]=facmtobinary(acm,symblutc)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Symbolize the block IN BINARY STRING.
%DC is stored differently
%size is converted in BIN number from 0 to 1010 and stored in the FOUR
%first bites of the symbolized block
dcsize = acm(1,1);
dcbin4 = dec2bin(dcsize);
ldcbin4 = length(dcbin4);

%zeropad so the total number of binary digits is 4
if ldcbin4==1
    dcbin4=['000' dcbin4];
elseif ldcbin4==2
    dcbin4=['00' dcbin4];
elseif ldcbin4==3
    dcbin4=['0' dcbin4];
elseif ldcbin4==0
    disp('Error Error dcbin4 has not length')
end

%APPEND DC SIZE
symbl(1:4) = dcbin4;
%disp(['Pst Coded vector DC: ',symbl]);

%amplitude position is stored as well. The number of bits is its size.

if (dcsize>0)
    camp = dec2bin(acm(1,2)-1); %>>>THE AMPLITUDE IS ZERO BASED, TAKE OUT ONE
else
    camp = '0'; %When Size is zero, the amplitude is zero as well
    dcsize = 1; %set size to one, since we have to allocate a symbol
end
%APPEND DC AMP POSITION
%In the case that camp has less digits than dcsize we have to append zeros
%to the left
lcamp = length(camp);
if (lcamp==dcsize)
    symbl(5:dcsize+4)=camp;
elseif(lcamp<dcsize)
    zzst = [];
    for i=lcamp:dcsize-1
        zzst = [zzst,'0'];
    end
    camp = [zzst,camp];
    symbl(5:dcsize+4)=camp;
else
    if camp ~= '0'
        disp('ERROR @ facmtobinary the dc size is smaller than the amplitude')
        camp
        lcamp
        dcsize
    end
end
%disp(['Pst Coded vector AM: ',symbl]);
```

```
%Now we will work in blocks of 3 elements, the first two are the zrun and
%size, the third is the amplitude.
%the Zrun and Size are encoded with Huffman. the Amp is stored in binary.
%This goes until the end of the vector acm
lacm = length(acm);
%start at the third element (AC1Zrun)
i=3;
while(i<=lacm-2)
    zrun = acm(1,i);
    asize = acm(1,i+1);
    complexn = zrun+j*asize;
    %Find its position in the look up table
    posfound = find(cell2mat(symblutc(:,1))==complexn);

    %Find the codeword at the third column
    codefound = char(symblutc(posfound,3));

    %APPEND the codeword to the encoded vector
    symbl = [symbl codefound];
    %disp([num2str(i),' Pst Coded vector HU: ',codefound,' from ',num2str(complexn)]);

    %Now encode in binary the amplitude position of the AC
    amppos = acm(1,i+2);
    if (amppos>0)
        camp = dec2bin(amppos-1); %>>>THE AMPLITUDE IS ZERO BASED, TAKE OUT ONE
    else
        camp = '0'; %When Size is zero, the amplitude is zero as well
        %disp('Warning size is zero')
    end

    %In the case that camp has less digits than size we have to append zeros
    %to the left
    lcamp = length(camp);
    if(lcamp<asize)
        zzst = [];
        for i2=lcamp:asize-1
            zzst = [zzst,'0'];
        end
        camp = [zzst,camp];
    elseif(lcamp>asize)
        if camp ~= '0'
            disp('ERROR @ facmtobinary the dc size is smaller than the amplitude')
            lcamp
            asize
        end
    end

    %APPEND the encoded amplitude position
    symbl = [symbl camp];
    %disp([num2str(i+2),' Pst Coded vector AM: ',symbl]);
    i = i+3;

end

%Now I have to append the EOB code which is at the last position of
%symblutc
[EOBplace coln]= size(symblutc);
EOBcode = char(symblutc(EOBplace,3));

%Append it
symbl = [symbl EOBcode];
```

```
% fdecodeimage.m
%this function gets the coded binary string based on a Huffman code and
%gives out the zigzaged image in rows
%Calls functions: fbinarytoacmr, fdecodeblock

function [qzzimage]=fdecodeimage(allsymbl,lsymbl)

%AC is stored with its symbol (z,s) by look up table and then the amplitude
```

```
load symblutc
load bec;

%find the number of rows
lenlsy = length(lsymbl);

%Now for each row of qzzimage decode it.
for irow = 1:lenlsy

    %Find the variable length code for a row
    vlength = lsymbl(irow);

    %Find the binary string of that length
    clear symblrow
    symblrow = allsymbl(1:vlength);

    %take out the string from the whole binary
    allsymbl = allsymbl(vlength+1:length(allsymbl));

    %Decode it find the acmrecovered string
    acmr=fbinarytoacmr(symblrow,symblutc);

    %find the dct zigzagged string of length 64
    qzzimage(irow,:) = fdecodeblock(acmr,bec)';

    %disp(['Decoded row: ',num2str(irow)]);
end
```

```
%this function decode a 8x8 block
%calls functions: findintdcamp
function [adecoded] = fdecodeblock(acm,bec)

%first, decode the amplitude
ampfound=findintdcamp(acm(1),acm(2),bec);

%Test point commented
%disp(['DC amplitude decoded: ',num2str(ampfound)]);

adecoded(1,1)=ampfound;

%Decode the AC values.
%read zeros until we get a nonzero value
inda = 2;   %index of the output vector
inm = 3;    %index of the ac code matrix
while(inda<=64)
    %read zerorun
    zrun = acm(inm);
    inm=inm+1;
    %create zeros
    for iz = 1:zrun
        adecoded(inda,1)=0;
        inda = inda+1;
    end
    %read coded number
    size = acm(inm);
    apmpos = acm(inm+1);
    ampfound=findintdcamp(size,apmpos,bec);
    adecoded(inda,1) = ampfound;
    inda = inda+1;
    inm=inm+2;
end
```

```
%fbinarytoacmr
%this function gets the acm vector (from fencodeblock) and give us a vlc
%binary string.
%call functions:

function [acmr]=fbinarytoacmr(symbl,symblutc)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%From the symbolized binary string get the acm
%we know the EOB is the last piece of code of symbl Therefore lets find it
%first and then decode the other part

%Find the EOB code which is at the last position of symblutc
[EOBplace coln]= size(symblutc);
EOBcode = char(symblutc(EOBplace,3));
%Find its length
EOBcodelength = length(EOBcode);

%Find the length of the symbol array
lensymbl = length(symbl);

%verify is the same code
if(~strcmp(EOBcode,symbl(lensymbl-EOBcodelength+1:lensymbl)))
    disp('ERROR @ fbinarytoacmr, EOB not found in code')
end

%Take it out from the symbl
symbl=symbl(1:lensymbl-EOBcodelength);

%Find the length of the new symbol array
lensymbl = length(symbl);

%Get the first four bites to find the DC size
pointsy = 4;
dcsize = symbl(1:pointsy);

%Dc size is binary code
dccsize = bin2dec(dcsize);
acmr(1,1) = dccsize;

%If size is 0 change to 1 because we need a space to put the '0'
if (dccsize == 0)
    dccsizereal = 1;
else
    dccsizereal = dccsize;
end
%disp(['PST recov vector DC size: ',num2str(dcsize),' : ',num2str(acmr(1,1))]);
%Get the next size bits and generate the amplitude position
pointsy= pointsy+dccsizereal;
ampos = symbl(5:pointsy);

%add one to the amp position if size is >0
if (dccsize >0)
    dampos = bin2dec(ampos)+1; %I HAVE TO PUT THE ONE TAKEN BEFORE
    acmr(1,2) = dampos;
else %if size is zero theamplitude position should be zero as well.
    dampos = bin2dec(ampos);
    acmr(1,2) = dampos;
    disp('Warning dc size is zero')
end
%disp(['PST recov vector DC amp : ',num2str(ampos),' : ',num2str(dampos)]);

accounter = 3;
while(pointsy < lensymbl-2)
    %GET THE HUFFMAN CODE (WE DONOT KNOW THE SIZE OF THE NEXT CODEWORD, WE
    %HAVE TO FIND IT FROM THE LOOK UP TABLE BIT BY BIT)
    pointsy= pointsy+1;
    off = pointsy;
    lfind = 3;
    while(lfind>1)
        codetofind = symbl(pointsy:off);
        match = strmatch(codetofind,symblutc(:,3));
        lfind = length(match);
        off= off+1;
    end
    hcomplex = cell2mat(symblutc(match,1));
    zrun = real(hcomplex); %This is the zero run
    acmr(1,accounter) = zrun;
    accounter = accounter +1;
```

```
    acsize = imag(hcomplex); %This is the size.
    acmr(1,accounter) = acsize;

    %now I have to read the amp position, I know the size though.
    pointsy = off; %get back the pointer
    %Advance to read the size
    %If size is 0 in fact it is 1 because we need a space to put the '0'
    if (acsize == 0)
        acsizereal = 1;
    else
        acsizereal = acsize;
    end
    pointsy = pointsy + acsizereal-1;
    ampos = symbl(off:pointsy);
    accounter = accounter +1;

    %add one to the amp position if size is >0
    if (acsize >0)
        dampos = bin2dec(ampos)+1; %I HAVE TO PUT THE ONE TAKEN BEFORE
        acmr(1,accounter) = dampos;
    else
        dampos = bin2dec(ampos);
        acmr(1,accounter) = dampos;
        %disp('Warning ac size is zero')
    end

%     disp(['PST recov vector AC Zr size apos: ',num2str(codetofind), ...
%         ' ',num2str(ampos),' : ',num2str(zrun),' (', ...
%         num2str(acsize),' ',num2str(dampos),')']);
    %disp(['Pre Coded vector: ',num2str(acm)]);
    %disp(['Rec Coded vector: ',num2str(acmr)]);

    %Now go back and find the others
    accounter = accounter +1;
end
```

```
% fimagizzag
%Function to create the inverse zigzag of a ???x??? image
%calls custom functions: fblockzigzag

function[qizzimage]=fimagizzag(qzzimage);


%The rows represent the number of blocks. The columns are always 64.
[rowizz,colizz] = size(qzzimage);
rowb = sqrt(rowizz*colizz);
colb = rowb; %this is only valid for square images

%I have to divide the image in 8x8 blocks
zzrow = 0;
for rowblock=1:8:rowb
    for colblock=1:8:colb
        zzrow = zzrow +1;
        block = fblockizigzag(qzzimage(zzrow,:));
        qizzimage(rowblock:rowblock+7, ...
            colblock:colblock+7)=block;
    end
end
```

```
%fblockizigzag
%Function to create 8x8 block image from zigzag scan.
%the output is a column vector

function [a] = fblockizigzag(azz)
a(1,1) = azz(1);
a(1,2) = azz(2);
a(2,1) = azz(3);
a(3,1) = azz(4);
```

```
a(2,2) = azz(5);
a(1,3) = azz(6);
a(1,4) = azz(7);
a(2,3) = azz(8);
a(3,2) = azz(9);

a(4,1)= azz(10);
a(5,1)= azz(11);
a(4,2)= azz(12);
a(3,3)= azz(13);
a(2,4)= azz(14);
a(1,5)= azz(15);
a(1,6)= azz(16);
a(2,5)= azz(17);
a(3,4)= azz(18);
a(4,3)= azz(19);

a(5,2)= azz(20);
a(6,1)= azz(21);
a(7,1)= azz(22);
a(6,2)= azz(23);
a(5,3)= azz(24);
a(4,4)= azz(25);
a(3,5)= azz(26);
a(2,6)= azz(27);
a(1,7)= azz(28);
a(1,8)= azz(29);

a(2,7)= azz(30);
a(3,6)= azz(31);
a(4,5)= azz(32);
a(5,4)= azz(33);
a(6,3)= azz(34);
a(7,2)= azz(35);
a(8,1)= azz(36);
a(8,2)= azz(37);
a(7,3)= azz(38);

a(6,4)= azz(39);
a(5,5)= azz(40);
a(4,6)= azz(41);
a(3,7)= azz(42);
a(2,8)= azz(43);
a(3,8)= azz(44);
a(4,7)= azz(45);
a(5,6)= azz(46);
a(6,5)= azz(47);
a(7,4)= azz(48);

a(8,3)= azz(49);
a(8,4)= azz(50);
a(7,5)= azz(51);
a(6,6)= azz(52);
a(5,7)= azz(53);
a(4,8)= azz(54);
a(5,8)= azz(55);
a(6,7)= azz(56);
a(7,6)= azz(57);
a(8,5)= azz(58);

a(8,6)= azz(59);
a(7,7)= azz(60);
a(6,8)= azz(61);
a(7,8)= azz(62);
a(8,7)= azz(63);
a(8,8)= azz(64);
```

```
%fimagiq
%Function to create the dct of a ???x??? image
%calls custom functions: fblockiq
```

```
function [iqimage]=fimagiq(qimage,beta,qcoeff)

[rowb,colb] = size(qimage);

%load quantization look up table
load qcoef

%I have to divide the image in 8x8 blocks
for rowblock=1:8:rowb
    for colblock=1:8:colb
        %disp(['row: ', num2str(rowblock),' col: ', num2str(colblock)]);
        %select a block
        qiblock=qimage(rowblock:rowblock+7,colblock:colblock+7);
        %perform the block Inverse quantization
        oblock=fblockiq(qiblock,beta,qcoeff);
        %store in the output Quantized image
        iqimage(rowblock:rowblock+7,colblock:colblock+7)=oblock;
    end
end
```

```
%fblockiq
%Function to inverse quantize the dct of a 8*8 block
%calls custom data: qcoef.mat

function [oblockiq]=fblockiq(Qxij,beta,qcoeff)

%load quantization look up table
% load qcoef

%multiply the scaling parameter
qij = beta.*qcoeff;

oblockiq = Qxij.*qij;
```

```
%fimagidct
%Function to create the Idct of a 512x512 image
%calls custom functions: fblockidct

function [oimage]=fimagidct(iimage)

[rowb,colb] = size(iimage);

%load the idct coefficients stored in mat file
load idct8

%I have to divide the image in 8x8 blocks
for rowblock=1:8:rowb
    for colblock=1:8:colb
        %disp(['row: ', num2str(rowblock),' col: ', num2str(colblock)]);
        %select a block
        iblock=iimage(rowblock:rowblock+7,colblock:colblock+7);
        %performthe dct
        oblock=fblockidct(iblock,liut);
        %store in the output DCT image
        oimage(rowblock:rowblock+7,colblock:colblock+7)=oblock;
    end
end
```

```
%fblockidct
%Function to create the dct of a 8*8 block
%calls custom functions: fidct

function [oblock]=fblockidct(iblock,liut)

[rowb,colb] = size(iblock);

%For each row make the dct
```

```
for ir = 1:rowb
    Ssignal=iblock(ir,:);
    xtemp(ir,:)=fidct(Ssignal,liut);
end

%For each column make the dct
for ic = 1:colb
    %transpose to create a row vector
    Ssignal=xtemp(:,ic)';
    %transpose the output of fdct to store in a column
    oblock(:,ic)=fidct(Ssignal,liut)';
end
```

```
%fidct
%Function to calculate the IDCT of a 8x8 vector
%calls data: dctcoeff.mat
function [xsignal]=fidct(S,liut)

%load the idct coefficients stored in mat file
% load idct8
sliut = size(liut);
n = sliut(1,1);

%Find the IDCT of an input signal
for u=0:n-1
    uin=u+1;

    %IDCT coefficient
    xsignal(uin) = S*liut(uin,:)';

end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%function PSNR process

function [PSNR]=fpsnr(image1, image2)

%substract one image from another
%dij = abs(image1 - image2);
dij = (image1 - image2);
%get the mean
% mdij = mean(mean(dij))
mdij = mean(dij(:));
%mdij2 = sum(sum(dij))/prod(size(dij))

%square without the mean
teee =(dij-mdij).^2;

%find the mean square error
MSE = abs(sum(sum(teee))/prod(size(dij)));


%when MSE is small.
if(MSE < 6.5025e-006)
    MSE = 6.5025e-006;
    disp(['Achieved higher Limit of PSNR. The images are the same'])
end
%find the inverse multiplied by the peak value of pixel
IMSE = 255^2/MSE;

%find the PSNR
PSNR = 10*log10(IMSE);
% disp(['MSE: ',num2str(MSE)])
disp(['PSNR: ',num2str(PSNR)])
```

```
%fwriteimg
%Function to write an image

```

```
function []=fwriteimg(strim,rsigd)
rsigd = round(rsigd');
rsigc = char(rsigd);
fid = fopen(strim,'wb');
isigc = fwrite(fid,rsigc,'uchar');
fclose(fid);
```

Screenshot

```
////////////////////Project II Main Program////////////////////
Progress message: DCT done
Elapsed time is 16.504000 seconds.
Beta: 1
Progress message: Q done
Elapsed time is 17.465000 seconds.
Progress message: ZZ done
Elapsed time is 39.106000 seconds.
Progress message: Encode Image done
Elapsed time is 256.259000 seconds.
Size in bits of the Encoded Image: 264655
Progress message: Decode Image done
Elapsed time is 494.651000 seconds.
Progress message: IZ done
Elapsed time is 495.763000 seconds.
Progress message: IQ done
Elapsed time is 496.664000 seconds.
Progress message: IDCT done
Elapsed time is 508.201000 seconds.
PSNR: 35.805
If you want to see the images, wait one second
////////////////////Project II End of Main Program////////////////////
```

To see the testing functions source code and the lookup table and coefficients creator, please refer to the file attached **mprojII.zip**

REFERENCES

[1] Dr. Zhihai (Henry) He, *"Visual Signal Processing and Communitation Class Notes"*. JPEG Image Compression Standard. Missouri University at Columbia. 2005.

[2] Yao Wang, Jörn Ostermann, Ya-Qin Zhang., *"Video Processing and Communications"*, Prentice Hall, Inc 2002.

[3] Matlab, *"Special Topics, Signal Processing ToolBox"*. The MathWorks Inc. 2004.