EE 522 – Data Compression – Homework 3

Bidimensional DCT and JPEG

Download all the .m and .pgm files in the same subdirectory, and set this subdirectory as your Matlab working sub-directory, where you will also save the Matlab script written following the instructions below.

1. Read with Matlab a PGM image (black and white image with levels of gray represented with 8 bits/pixel) selected among the test images (command **imread**), show it on the screen (command **imshow**) and save it in a matrix of real numbers **img_var** (command **double**)

```
image=imread('lena512.pgm');
figure(1);
imshow(image);
title(' original image ');
% convert image into floating point numbers
img_var=double(image);
```

2. Measure the number of rows **M** and columns **N** of **img_var** (command **size**)

```
siz=size(img_var);
M=siz(1);
N=siz(2);
```

3. Initialize two empty matrices, **quantized** and **reconstructed** , with the same size as **img_var** (command **zeros**)

```
quantized=zeros(M,N);
reconstructed=zeros(M,N);
```

4. Fix an initial value of the quality parameter **q=10**, and select a value for the parameter **S** (the parameter S is explained later)

```
q = 10;
S = s_eval(q);
```

5. Initialize the values of the number of vertical 8x8 blocks per image M1, the number of horizontal blocks per image N1, and a vector b8 containing the indexes 1 2 3 4 5 6 7 8

```
b8=1:1:8;
M1=M/8;  % n. of vertical blocks in image
N1=N/8;  % n. of horizontal blocks in image
```

6. Write a double loop on the number of horizontal and vertical blocks (with indexes **i_row** and **i_col**)

```
for i_row=0:M1-1;
    for i_col=0:N1-1;
    …
    end
end
```

that will perform the following steps:

- Iteratively copies all the 8x8 blocks of the matrix **img_var** in the 8 x 8 submatrix **block.** This can be done with the command

```
block=img_var(i_row*8+b8,i_col*8+b8);
```

- Performs a bidimensional DCT (command **dct2**) of **block**, obtaining the matrix **block_dct**

```
block_dct=dct2(block);
```

- Simulates on **block_dct** the effect of quantization, generating the quantized block **block_dct_qt.** This can be done with the command

```
block_dct_dqt=block_dct_qt.*qmat*S;
```
where

  - **block_dct** is the 8x8 block currently being processed
  - **qmat** is a 8x8 weight matrix contained in the file **qmat.m** (provided with the lab)
  - **S** is a value obtained from the expression (2) (can be generated with the function **s_eval.m** provided with the lab)

- Assembles all the quantized 8x8 blocks **block_dct_qt** in the DCT transformed and quantized matrix **quantized** (previously defined)**.** This can be done with the command

```
quantized(i_row*8+b8,i_col*8+b8) = block_dct_qt;
```

Note: S is a function of the parameter **q**, that specifies the quality of the compressed image. q must assume values between 10 and 90. For good qualities q (large values of q), S assumes small values, and the quantization has better resolution. For worse quality (small values of q), S assumes large values, and a heavier quantization is performed.

$$(1)\ MSE = \frac{1}{NM}\sum_{i}^{N}\sum_{j}^{M}[A(i,j) - B(i,j)]^2$$

$$PSNR = 10\ log_{10}\left(\frac{255^2}{MSE}\right)$$

$$(2)\quad S = \begin{cases} \frac{100-q}{50} & \text{for } q > 50 \\ \frac{50}{q} & \text{for } q \leq 50 \end{cases}$$

- Dequantizes the block **block_dct_qt** and writes it the variable **block_dct_dqt,** applying the expression

```
block_dct_dqt=block_dct_qt.*qmat*S;
```

- Performs inverse DCT transform of each block **block_dct_dqt** (command **idct2**), and reassemble the reconstructed matrix **reconstructed**

```
block_dct_dqt_idct=idct2(block_dct_dqt);
reconstructed(i_row*8+b8,i_col*8+b8)=block_dct_dqt_idct;
```

7. Convert the matrix of real numbers **reconstructed** into a matrix of 8 bits/pixel **image_reconstructed** (command **uint8**)

```
image_reconstructed=uint8(reconstructed);
```

8. Save the matrix **quantized** in a file **file1,** compress it with a lossless compressor (**winzip**) and check its dimensions, and considering that **image** contains **NM** pixels, evaluate the number of bits per pixel **R** after compression. This can also be achieved with the commands

```
fid = fopen('file1.txt','w');          % opens a file “file1.txt”
fwrite(fid, quantized, 'int8');        % stores quantized into file1.txt
fclose(fid);                           % closes the file
zip('file1','file1.txt');              % compresses file1.txt into file1.zip
sizQ = dir('file1.zip');               % writes information about “file1.zip”
                                       % into the variable sizQ (getFileStat function)
filesizeQ = sizQ.bytes;                % writes in filesizeQ the number of
                                       % bytes of “file1.zip”
R = 8*filesizeQ/(M*N);                 % number of bits/pixel
```

9. Evaluate the MSE (expression (1)) between the original image **image** and **image_reconstructed.** This will be the distortion parameter **D.**

```
errorQ = abs(img-image_reconstructed);
error2Q= errorQ.^2;
D=mean(mean(error2Q));
```

10. Repeat for different quality parameters (i.e. for **q =** 10,20,30,40, 50,60,70,80) and store the values of **R** and **D** for all values of **q.**

11. Plot a graph linking the MSE **D** and the rate **R** (in bpp derived from the dimension of the compressed file)

The attached figures graphically depict the flow of operations.

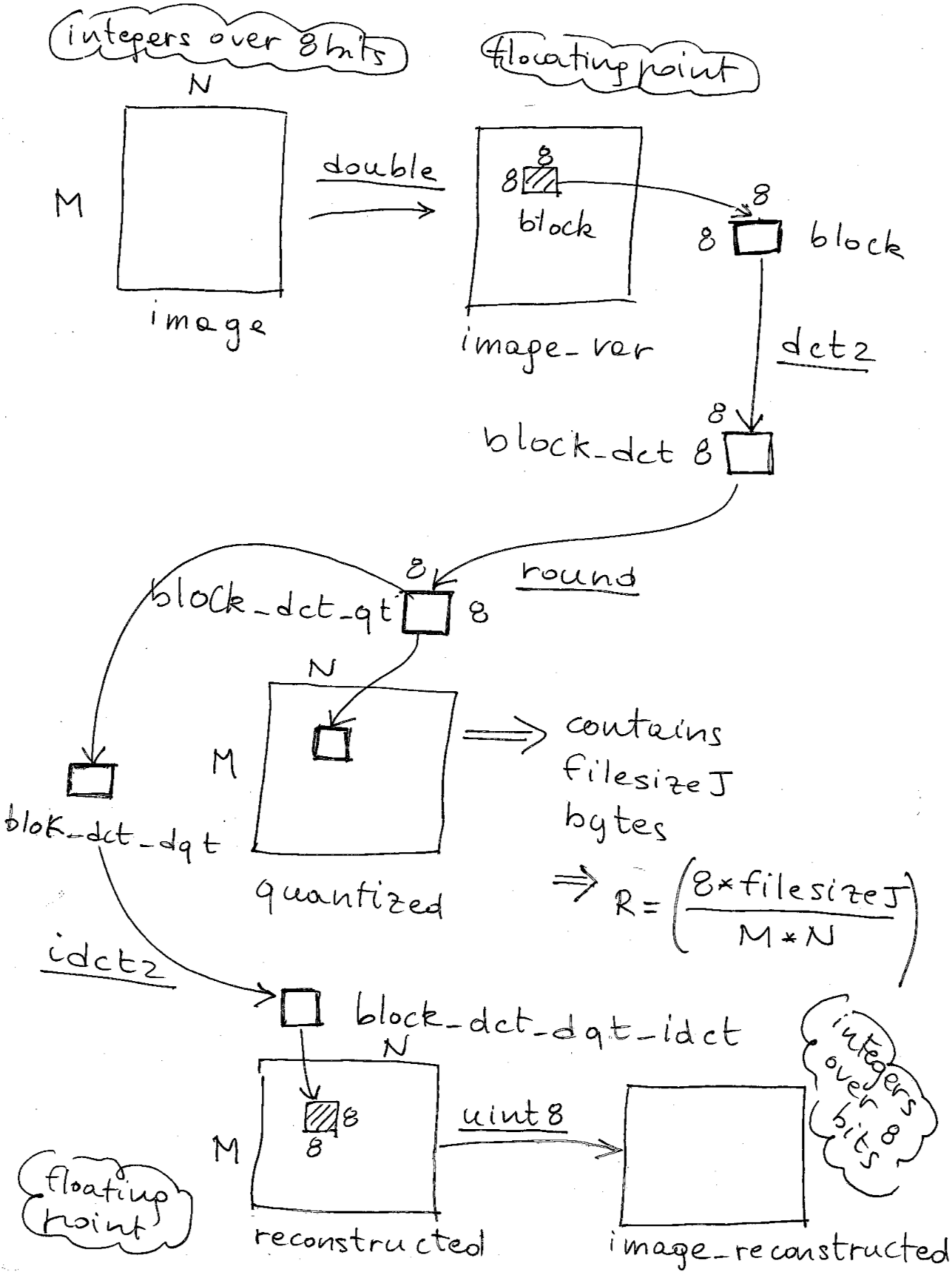
Comparison with the built-in Matlab JPEG compressor

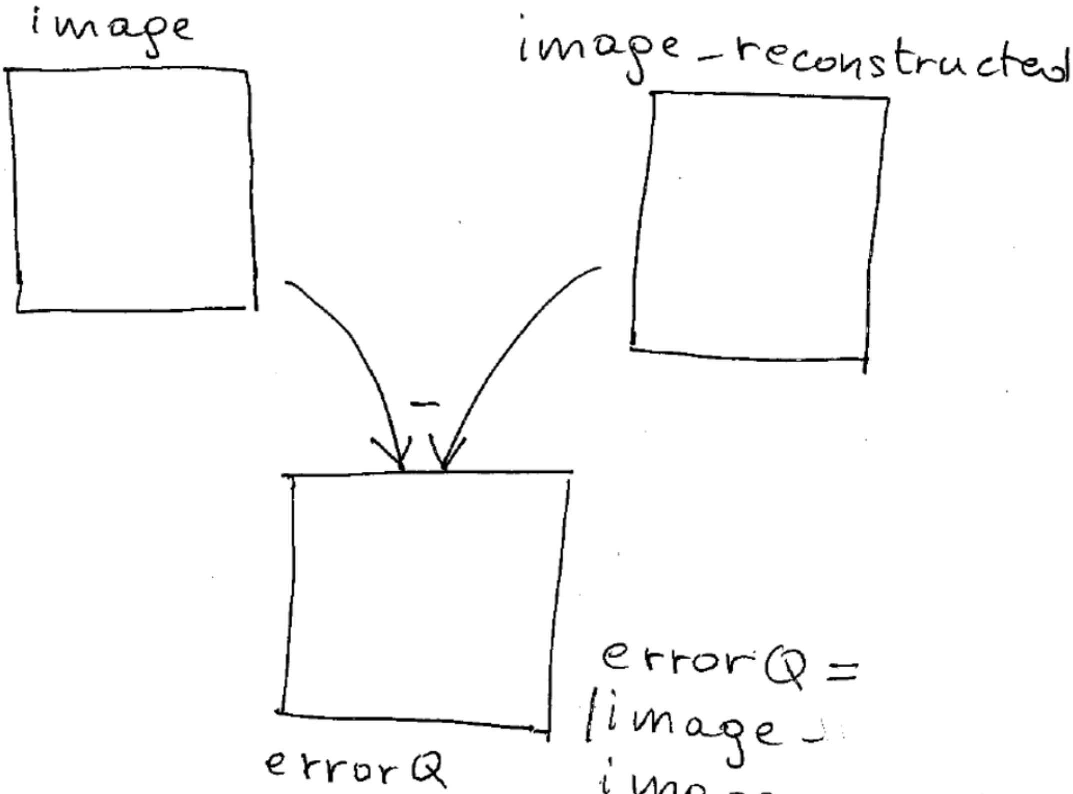
12. Using the **imwrite** function with ‘JPEG’ parameter, encode the PMG image **image** with the same quality parameter **q,** and compare the values of the MSE **D** and the rate **R** obtained in this case. You can use the command

```
imwrite(image,'file1.jpg','Quality',q);
```

13. Plot the rate distortion curve (R versus D) obtained with the built-in JPEG

Write a report describing the algorithm and commenting the obtained results and the observed visual effects.


integers over 8 bits
floating point
N
M
image
double
8
8
block
image_ver
8
8
block
dct2
8
8
block_dct
round
8
block_dct_qt
8
N
M
quantized
contains
filesizeJ
bytes
$\Rightarrow R = \left(\frac{8 * filesizeJ}{M * N}\right)$
blok_dct_dqt
idct2
block_dct_dqt_idct
N
M
8
8
reconstructed
uint8
image_reconstructed
integers over 8 bits
floating point



errorQ = |image - image_reconstructed|

$$D = \frac{1}{MN} \sum_{i=1}^{N} \sum_{j=1}^{M} |errorQ(i,j)|^2$$

in Matlab

$\Rightarrow$ D = mean(mean(errorQ.^2));