Practical Assignment Part 1

Shao Hsuan Hung (1723219)
s.hung@student.tue.nl

January 8, 2023

1 PCM and SNR

The code of PCM and SNR please refer to the file: "HW1_part1.m"
(a) Create your own simple PCM compression in MATLAB.
As shown in below code block,the procedure of simple PCM compression:

- Scaling the image: $x_s = \frac{x-\mu_x}{\sigma_x}$
- Given the bit per pixel parameter and quantize the image.
- Encoding the quantized image.
- Decoding the quantized image.
- Dequantized the image.

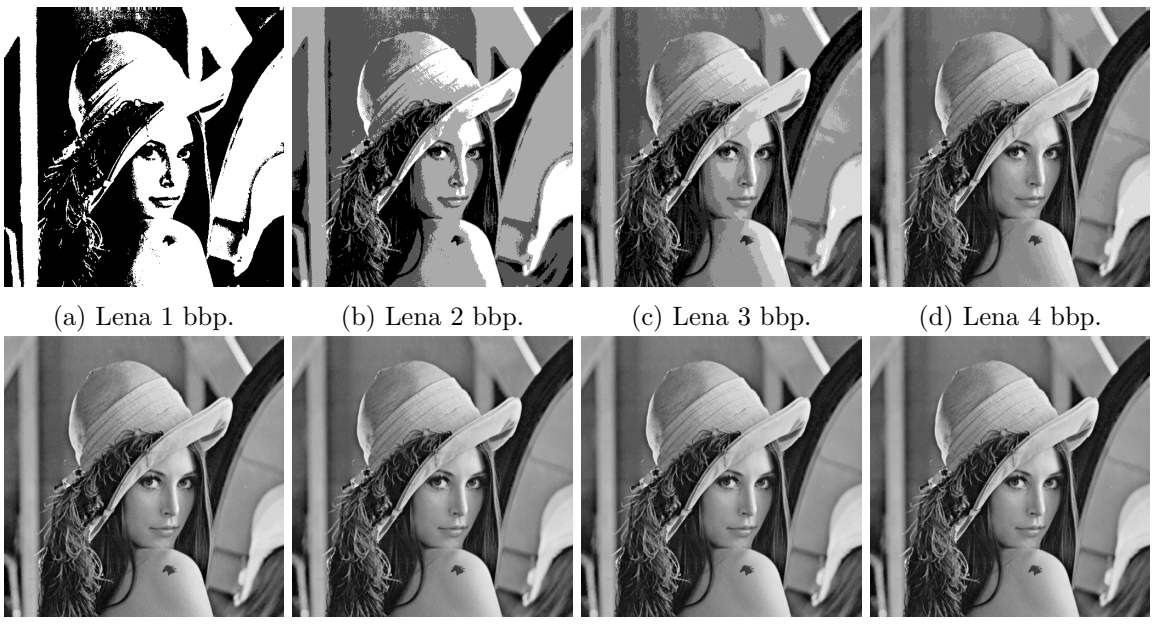
```
%% Implement PCM
function coding = PCM(img,bpp)
    % Scaling procedure
    n_img = double(img);
    mu = mean2(n_img);
    std = std2(n_img);
    normal_img = (n_img-mu)./std;
    % 1. Quantizer: uniform quantizer
    [q_level, q_img] = uniform_quantizer(normal_img,bpp);
    % 2. Encoding
    [dict,coding] = encoding(q_img,q_level,bpp);
    % 3. Decoding
    decode = huffmandeco(code,dict);
    % 4. Dequantized the image would return the same q_imag
end
%% Quantizer
function [q_level, new_img] = uniform_quantizer(img, bit_rate)
    img = double(img);
    maxValue =  max(max(img));
    minValue = min(min(img));
    Δ = (maxValue - minValue)/(2^bit_rate);
    q_level = zeros(bit_rate,1);
    for i =1:1:2^bit_rate
        q_level(i) = minValue + Δ*(i);
    end
    new_img = zeros(size(img));
    for j = 1:1:size(img,1)
         for k = 1:1:size(img,2)
               for i = 1:1:2^bit_rate
                    if(img(j,k) ≤ q_level(i))
                        new_img(j,k) = q_level(i);
                        break
                    end
                    if(img(j,k) ≥ q_level(2^bit_rate))
                        new_img(j,k) = q_level(2^bit_rate);
                    end
               end
         end
    end
```

```
end
%% Encoding
function code = encoding(img,q_level,bit_rate)
    img = reshape(img,[],1);
    q_level = reshape(q_level,[],1)
    probs = zeros(2^bit_rate,1);
    for k = 1:1:size(q_level,1)
        number = 0;
        for i = 1:1:size(img,1)
            if(img(i)==q_level(k))
               number = number + 1;
            end
        end
        probs(k) = number/(size(img,1));
    end
    [dict, avglen]= huffmandict(q_level,probs);
    code = huffmanenco(img,dict);
end
%% Decoding
function decode = decoding(code,dict)
    decode = huffmandeco(code,dict);
end
```

(b) Read in the images 'lena.pgm' and 'peppers.pgm', compress them with your PCM encoder, change the number of bits per pixel (bpp) and show a diagram depicting the SNR versus the bitrate after compression. Also show the resulting (compressed) images for 'lena.pgm'.

The PCM encoding with different of bbp are shown in figure 1. To measure the image quality, the Signal-to-Noise-Ratio (SNR) shown in equation 1 is used, where $f(x, y)$ represent the pixel values on the corresponding position and $\hat{f}(x, y)$ represent the pixel values of the encoded image, M and N represent the height and width of the image by pixel. The SNR of different bbp value has been calculated in figure 2 for both lena.pgm and peppers.pgm from 2 bpp to 7 bpp. The lines are nearly linear on a logarithmic scale, which means increasing the bpp would make the SNR increase exponentially.

$$SNR = \frac{\sum_{x=0}^{M}\sum_{y=0}^{N}\hat{f}(x,y)^2}{\sum_{x=0}^{M}\sum_{y=0}^{N}\left[f(x,y)-\hat{f}(x,y)\right]^2} \tag{1}$$



(a) Lena 1 bbp. (b) Lena 2 bbp. (c) Lena 3 bbp. (d) Lena 4 bbp.

(e) Lena 5 bbp. (f) Lena 6 bbp. (g) Lena 7 bbp. (h) Lena 8 bbp.

Figure 1: PCM encoding with diiferent of bpp on lena.pgm

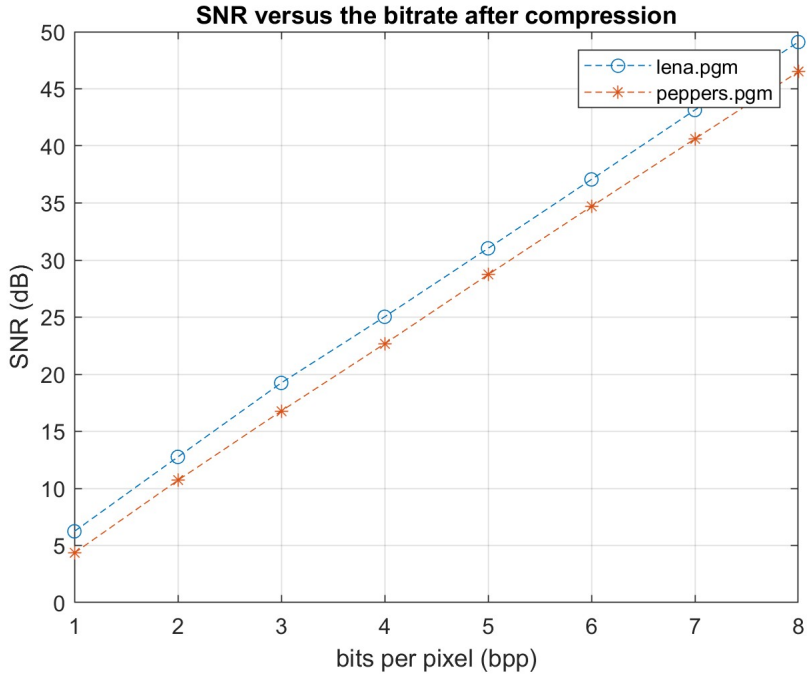
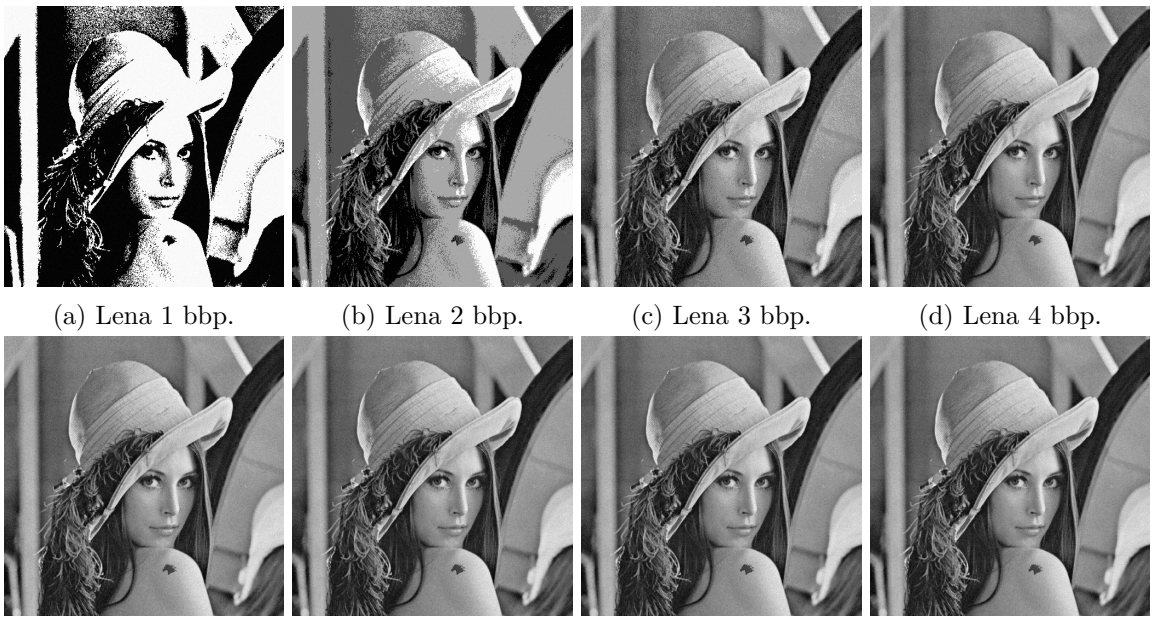


Figure 2: SNR versus the bbp after compression for both lena.pgm and peppers.pgm

(c).Prior to PCM compression, a small amount of uniform noise can be added to the image. This causes PCM compression artifacts at low bit rates to be less objectionable. Since the noise is generated deterministic, it can be subtracted at the decoder side. Add this so-called 'dither' to the image prior to PCM compression and repeat exercise 1(b). Do not forget to subtract the dither at the decoding side.

The result of the PCM encoding with change bpp on dither noise lena.pgm are shown in figure 3, Since we subtract the dither after the decoding, there are some noise on the compressed image, which make the value of SNR degraded, as shown in the figure 3. However, to human eye precision, it seems contain more detail and shadow especially in the low bit rate image. (Please refer to the sub-figure (b) of figure 1 and 3)



(a) Lena 1 bbp. (b) Lena 2 bbp. (c) Lena 3 bbp. (d) Lena 4 bbp.

(e) Lena 5 bbp. (f) Lena 6 bbp. (g) Lena 7 bbp. (h) Lena 8 bbp.

Figure 3: PCM encoding with diiferent of bpp on lena.pgm with dither noise after decoding

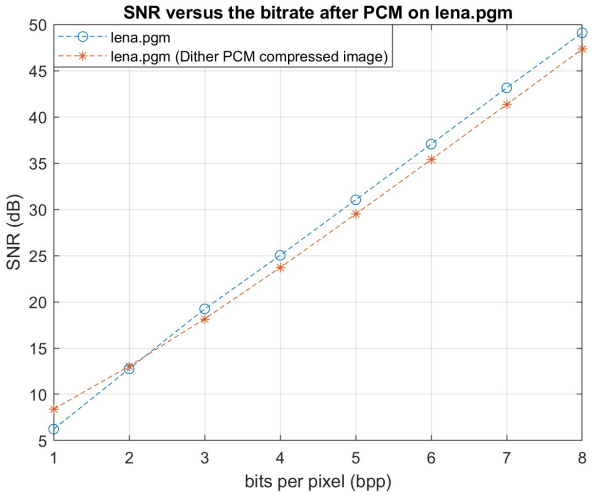


Figure 4: SNR versus bpp after adding dither noise on lean.pgm

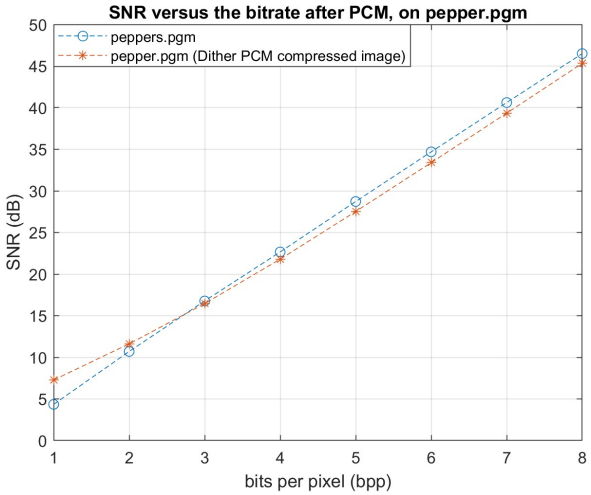


Figure 5: SNR versus bpp after adding dither noise on peppers.pgm

(d)Compare the visual quality of a PCM compressed and dithered PCM compressed image.

Compare the SNR curve with dither noise with the SNR curve without adding the noise, after adding dither noise, the value of SNR is decreased for the same bit rate. If the uniform noise increase, then the value of SNR is decreased more. However, in the low bit rate (1 bits to 2 bits as shown in the figure 4 and 5), the SNR of the compressed dither noise images (about 10 dB) is larger than the PCM compressed images (about 5dB). The reason is that the dither noise causes PCM compression artifacts at low bit rates to be less objectionable. When the bit rate increase (3 bits to 8 bits), the dither noise would not help to be less objectionable anymore, and make the value of SNR decrease compare to the original PCM compressed image.

(e) Describe the dependency of the SNR on the bits per pixel.

Overall, the value of SNR is proportional to the bits per pixel becaise the larger bits rate means the compressed image can keep moer detailed information after the quantization. If the noise is added in the image, then the value of SNR would decrease compare to the compressed image without noise. As shown in the figure 4 and 5, after adding the dither noise, the SNR of 8-bpp compressed "lena.pgm" decrease from 47 dB to 49 dB, the SNR of 8-bpp compressed "peppers.pgm" decrease from 47 dB to 45 dB. The added dither noise become larger, the larger decreased value.

(f)How many bits per pixel would you suggest as a minimum for applications such as (I) video monitoring, (II) videophone and (III) television? Please motivate your answer for PCM compression and dithered PCM compression.

Based on the value of SNR for PCM compression and dithered PCM compression, I would suggest the minimum for applications is 3 bits. Since the application of video monitoring, videophone and television would want to have a relative stable, low transmission time and fine visual equation. So the small bit rate but not-so-bad visual equation is preferred. Below 3 bits, the image look too objectionable even with the dither noise. Above 3 bits, the image looks almost same to the original image.

2 Quantization

The code of this part please refer to: "HW1_part2_DCT.m" and "HW1_part2_DPCM.m"
(a) Draw a diagram to show the change of SNR with respect to the change of the bit rate, e.g. use quantization steps that can be achieved using 1 to 8 bits. For DCT, a uniform quantizer is sufficient, although more complex methods are allowed.

The result of SNR versus bit per pixels for pipeline(I) and pipeline(II) are shown in the figure 6 and 7 separately. From the diagram of the pipeline (I), we can observe that the DCT compression cannot perform well compare to DPCM compression on "lena.pgm" and "peppers.pgm". Generally speaking, the bit per pixel seems proportional to the value of SNR for both DCT compression and DPCM compression. For DCT compression, increasing the bpp from 1 bit to 8 bits, the SNR change from 3dB to 25 dB almost linearly. For DPCM compression, increasing the bpp from 1 but to 8 bits, the SNR change from about 10 dB to 45 dB linearly. The reason why the DPCM compression is perform better than the DCT compression is because the DPCM have "predictor" to store the image pixel, and the data stored in the predictor is not quantized, only the error is quantized (based on the DPCM encoder scheme), so it can recover a lot of the information when generating the compressed image. Thus, the DPCM can have higher value of SNR. (The visualize result is shown in the figure 8 and 9.

For DCT compression pipeline (I), when the bpp increasing from 5 bits to 8 bit, the DCT compression have better performance on the peppers.pgm than the lena.pgm. For the DPCM compression pipeline(II), there is not obvious different on the performance on different image. To be brief, for different images (lean.pgm and peppers.pgm), the same pipeline have similar performance on the different image. For different pipeline compress the same image, the DPCM pipeline have better performance on SNR than the DCT pipeline.

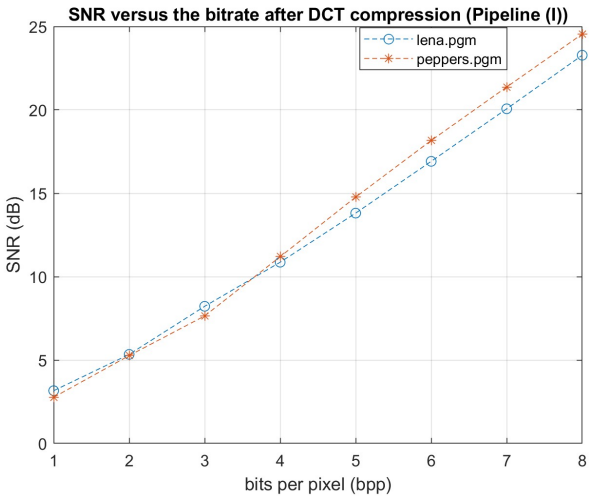


Figure 6: SNR versus bpp (DCT, pipeline (I))

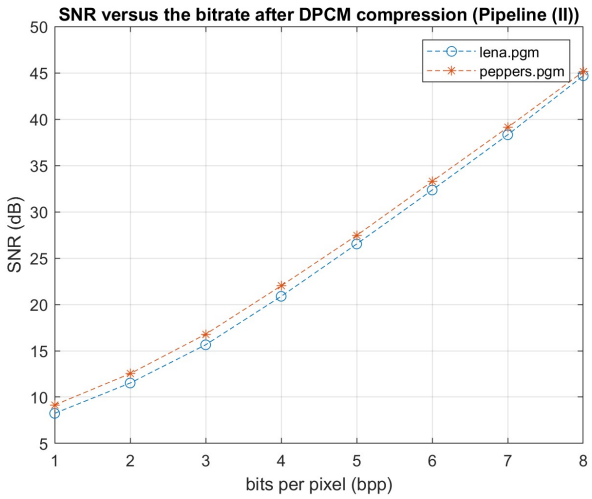


Figure 7: SNR versus bpp (DPCM, pipeline(II))

(b) Draw a diagram to show the change of the subjective picture quality with respect to the change of the bit rate. You can grade the visual quality of images as 1) Bad, 2) Poor, 3) Fair, 4) Good, and 5) Excellent (the criteria for defining these quality scales can be found in course module 1A, slide 17). Motivate your answer!

The result of compressed images done by the pipeline(I) and pipeline(II) are shown as figure 8 and 9. The diagram of the subjective picture quality w.r.t the change of the bit rate is shown in the figure 10. For the pipeline(I), the compressed image is really blur and only see some contour of the face when the bit rate is low (1 to 2 bpp), when the bit rate is up from 3 to 5 bits that the image start to become clear, so I only grade it as "Fair" since some details are still ambiguous. For the bit rate increase to 6 to 8 bits, the detail in the image is much more clearer, but still can see some aliasing noise block. So I grade it as "Good".

For the pipeline(II), the compressed image are relatively clear than the pipeline(I), for bit rate is 1 to 4 bits, the contour of the woman is clear but with some noise. These shape of the noise is line-shape, that is because one pixel is decoded based on the error from the pixels from previous column, rows and the upper left. For the bit rate is 5 bits, the line-shape noise disappear, but still can see little noise on the edge of the image

compare to the original one. For bit rate is 6 to 8 bits, the images look no different to the original image.

Figure 8: DCT pipeline (I) with changing bpp on lena.pgm (put images by ascending order of the bit rate)

Figure 9: DPCM pipeline (II) with changing bpp on lena.pgm (put images by ascending order of the bit rate)

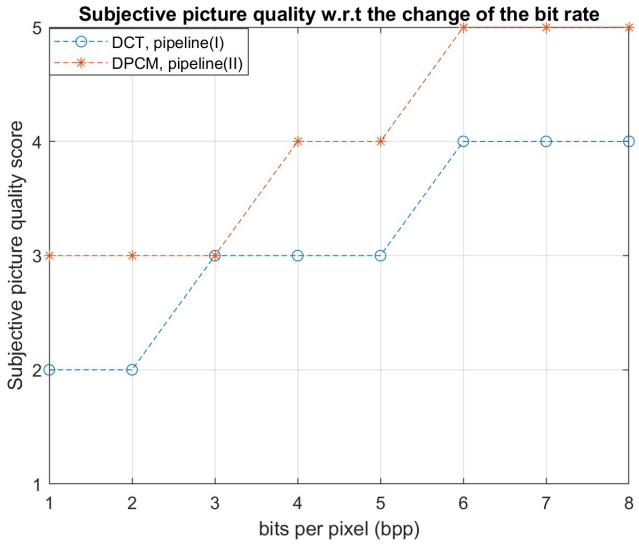


Figure 10: Subjective picture quality of changing bit rate with pipeline(I) and pipeline(II)

(c) Performance advice: which codec pipeline would you use for different compression factors/quality.

I would use the DPCM pipeline(II) if I need high quality compression, since the pipeline(II) can have fair compression quality even in the low bit rate. However, the disadvanges of the pipeline(II) is that it need more memory to store the parameter in predictor when calculating the error and thus having longer computation time. So if I need fast processing time and havce limit computation resource, I would choose DCT pipeline(I) with higer bit rate, to have faster processing time to have high quality compressed image.

3 DCT - frequency domain

The code of this part refer to "HW1_part3.m"

(a) Change of the picture brightness: $f(x, y) = f(x, y) + \Delta h$**.**

To change the brightness of the image, a uniform matrix matrix is being transformed in to the DCT domain and added to the DCT coefficient of the original image, (in this case, I use 90 and -90 (a scalar) times to a 512-by-512 matrix of ones). The result is shown as figure 11. The result shows that for a positive scalar, the overall image would looks brighter, while for a negative scalar, the overall image would looks dimmer.

Figure 11: Changing the brightness by adding constant to the DCT coefficient

(b) Change of the image contrast: $f(x, y) = f(x, y) \times c$**.**

To change the contrast of the image, a scalar is multiply to the DCT coefficient. In this case, I multiply $c = 0.5$ and $c = 1.5$ as shown in the figure 12. The result shows that when the scalar c is smaller than 1, the image would become less contrast, the difference of pixel value of features would be less different, while when the scalar c is larger that 1, the contrast of the image is enhanced, the difference of pixel value of features would be more distinct. For example, the fur on the sunhat of Lena looks more detail when enhancing the contrast. On the other hand, the fur looks blur when the contrast of the image is low.

Figure 12: Changing the contrast by timing a constant to the DCT coefficient

(c) Rotate the image by 180 degree.

Assume the $r(x, y)$ is a new block after rotating the 8x8 block $f(x, y)$ rotated by 180°, so the relation between the two blocks is: $r(x, y) = f(7 - x, 7 - y)$, where $x = 0, 1, ..., 7$ and $y = 0, 1, ..., 7$. After the DCT transform, $DCT\{f(x, y)\} = F(u, v)$, $DCT\{r(u, v)\} = R(u, v)$, and the $F(u, v)$ and $R(u, v)$ are shown as below:

$$F(u,v) = \frac{1}{4}C(u)C(v)\sum_{x=0}^{7}\sum_{y=0}^{7} f(x,y)\cos\left(\frac{(2x+1)u\pi}{16}\right)\cos\left(\frac{(2y+1)v\pi}{16}\right) \tag{2}$$

$$R(u,v) = \frac{1}{4}C(u)C(v)\sum_{x=0}^{7}\sum_{y=0}^{7} g(x,y)\cos\left(\frac{(2x+1)u\pi}{16}\right)\cos\left(\frac{(2y+1)v\pi}{16}\right) \tag{3}$$

We can substitute $g(x, y) = f(7 - x, 7 - y)$, so the equation(3) become:

$$R(u,v) = \frac{1}{4}C(u)C(v)\sum_{x=0}^{7}\sum_{y=0}^{7} f(7-x,7-y)\cos\left(\frac{(2x+1)u\pi}{16}\right)\cos\left(\frac{(2y+1)v\pi}{16}\right) \tag{4}$$

Now let assume $x' = 7 - x\, y' = 7 - y$, so the $R(u, v)$ become:

$$R(u,v) = \frac{1}{4}C(u)C(v)\sum_{x=0}^{7}\sum_{y=0}^{7} f(x,y)\cos\left(\frac{(2(7-x)+1)u\pi}{16}\right)\cos\left(\frac{(2(7-y)+1)v\pi}{16}\right) \tag{5}$$

Then change the numerates of the cosine term from $2(7-x)+1$ to $16-(2x+1)$, $2(7-y)+1$ to $16-(2y+1)$,so $R(u,v)$ becomes:

$$R(u,v) = \frac{1}{4}C(u)C(v)\sum_{x=0}^{7}\sum_{y=0}^{7} f(x,y)\cos\left((1-\alpha)u\pi\right)\cos\left((1-\beta)v\pi\right) \tag{6}$$

Where $\alpha = (2x+1)/16$ and $\beta = (2y+1)/16$.
Since we have

$$\cos(1-\alpha)u\pi = \begin{cases} -\cos\alpha u\pi, & u = 2n+1, \quad n = 0,1,2,...,7 \\ \cos\alpha u\pi, & u = 2n, \quad n = 0,1,2,...,7 \end{cases} \tag{7}$$

$$\cos(1-\beta)v\pi = \begin{cases} -\cos\beta v\pi, & v = 2n+1, \quad n = 0,1,2,...,7 \\ \cos\beta v\pi, & v = 2n, \quad n = 0,1,2,...,7 \end{cases} \tag{8}$$

This mean that the position of (u,v) would make the value of consine become positive or negative. For $u+v$ is odd, the multiplication of the cosine would be negative as shown in equation (11):

$$R(u,v) = \frac{1}{4}C(u)C(v)\sum_{x=0}^{7}\sum_{y=0}^{7} f(x,y)\cos\left(\frac{(2(7-x)+1)u\pi}{16}\right)\cos\left(\frac{(2(7-y)+1)v\pi}{16}\right) \tag{9}$$

$$= \frac{1}{4}C(u)C(v)\sum_{x=0}^{7}\sum_{y=0}^{7} f(x,y)\cos(1-\alpha)u\pi\cos(1-\beta)v\pi \tag{10}$$

$$= \frac{1}{4}C(u)C(v)\sum_{x=0}^{7}\sum_{y=0}^{7} [-f(x,y)\cos\alpha u\pi\cos\beta v\pi] \tag{11}$$

Same reason, for $u+v$ is even, the multiplication of the cosine would be positive as shown in equation (13):

$$R(u,v) = \frac{1}{4}C(u)C(v)\sum_{x=0}^{7}\sum_{y=0}^{7} f(x,y)\cos\left(\frac{(2(7-x)+1)u\pi}{16}\right)\cos\left(\frac{(2(7-y)+1)v\pi}{16}\right) \tag{12}$$

$$= \frac{1}{4}C(u)C(v)\sum_{x=0}^{7}\sum_{y=0}^{7} [f(x,y)\cos\alpha u\pi\cos\beta v\pi] \tag{13}$$

In conclusion, in order to rotate the image by 180 degree, the DCT coefficient should times to -1 when the sum of the row and column index is odd, and keep the coefficient when the sum of the row and column index is even. The formula is shown in the equation (14).

$$R(u,v) = \begin{cases} F(u,v), & u+v = 2n, \quad n = 0,1,2...,7 \\ -F(u,v), & u+v = 2n+1, \quad n = 0,1,2...,7 \end{cases} \tag{14}$$

The result is shown in figure 13, and the MATLAB code is also shown in below:

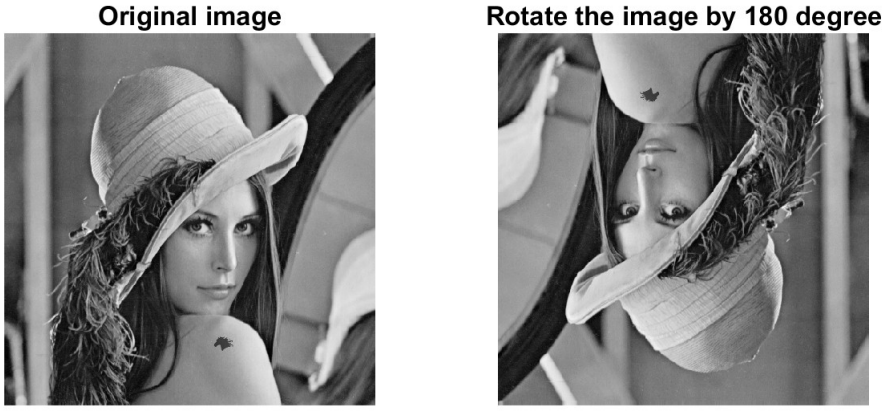


Figure 13: Rotate 180 degree by modifying the DCT coefficient

```
lena = double(imread("lena.pgm"));
H = [1 -1 1 -1 1 -1 1 -1;
     -1 1 -1 1 -1 1 -1 1;
     1 -1 1 -1 1 -1 1 -1;
     -1 1 -1 1 -1 1 -1 1;
     1 -1 1 -1 1 -1 1 -1;
     -1 1 -1 1 -1 1 -1 1;
     1 -1 1 -1 1 -1 1 -1;
     -1 1 -1 1 -1 1 -1 1;]; %if u+v = odd, *(-1)
% function handler
dct = @(block_struct) dct2(block_struct.data);
idct = @(block_struct) idct2(block_struct.data);
odd_pos_minus_one = @(block_struct) (block_struct.data).*H;
central_sym = @(block_struch) block_struch.data(end:-1:1,end:-1:1)';
% DCT
dct_coeff_img = (blockproc(lena,[8 8],dct));
% Rotate
for i = 1:512/8
    for j = 1:512/8
        dct_coeff_img_revese(1+8*(i-1):1+8*(i-1)+7,1+8*(j-1):1+8*(j-1)+7) = ...
    dct_coeff_img(512-8*i+1:512-8*i+8,512-8*j+1:512-8*j+8);
    end
end
rotate_dct_coeff_img = blockproc(dct_coeff_img_revese,[8 8],odd_pos_minus_one);
% IDCT
rotate_img = blockproc(rotate_dct_coeff_img,[8 8],idct);
```

4 Haar Transform

The code of this part refer to file:"HW1_part4.m"

(a) Give the 8 × 8 Haar matrix.

The 8 × 8 Haar matrix is shown below, and the implementation of the haar matrix MATLAB code is shown in below code block.

$$H_3 = \frac{1}{\sqrt{8}} \begin{Bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ \sqrt{2} & \sqrt{2} & -\sqrt{2} & -\sqrt{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sqrt{2} & \sqrt{2} & -\sqrt{2} & -\sqrt{2} \\ 2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & -2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & -2 \end{Bmatrix}$$

```
%% Generate the nxn haar matrix
function haar_matrix = Haar_Matrix(N)
    % Check the input argument
    if ((log2(N)-floor(log2(N)))≠0 || N<2)
        error('The input argument should be of form 2^k');
    end
    n = log2(N);
    % Build the p q matrix 2^p + q -1 = k
    p(1:2)= [0 0]; q(1:2) = [0 1];
    for i = 1:n-1
        p= [p i*ones(1,2^i)];
        q= [q  1:2^(i)];
    end
    haar_matrix = zeros(N); haar_matrix(1,:) = 1;
    for i = 2:N
        P = p(i);
        Q = q(i);
        for j = N*(Q-1)/(2^P):N*(Q-0.5)/(2^P)-1
```

```
            haar_matrix(i,j+1) = 2^(P/2);
        end
        for j = N*(Q-0.5)/(2^P):N*(Q)/(2^P)-1
            haar_matrix(i,j+1) = -2^(P/2);
        end
    end
    haar_matrix = haar_matrix;/sqrt(N);
end
```

(b) Implement an 8-point Haar transformation and the inverse transformation in MATLAB.

```
function [compressed_img] = HaarTransform(img,N,bpp)
    % Input parameter: N, is dimension of the haar matrix, in this case, is 8
    % Step1, Generate the Haar matrix by NxN
    haar = Haar_Matrix(N);
    % Step2. Haar transform
    haar_transform  = @(blockstruc) haar*blockstruc.data*haar';
    inv_haar_transform = @(blockstruc) haar'*blockstruc.data*haar;
    transformed_img = blockproc(img,[8 8],haar_transform);
    % Step3. Quantized the harr coefficient
    [haar_quant_img, q_level, Δ] = Haar_quantization(transformed_img,bpp);
    % Step4. Encoding the haar coefficient
    [decode_dict,code] = encoding(haar_quant_img,q_level,bpp);
    % Step5. Decoding the haar coefficient
    haar_quant_img = decoding(code,decode_dict,size(img,2));
    % Step6. Dequantized the haar coefficient
    haar_dequant_img = Haar_dequantization(haar_quant_img,Δ);
    % Step7. inverse haar transform
    compressed_img = blockproc(haar_quant_img,[8 8],inv_haar_transform);
end
function [new_img,q_level,Δ] = Haar_quantization(img,bit_rate)
    % Return:
    %   q_level: for encoding
    %   Δ  : for dequantization
    MaxValue = max(img(:));
    MinValue = min(img(:));
    Δ = (max(img(:))-min(img(:)))/(2^bit_rate);
    new_img = (fix(img./Δ));
    q_level = unique(new_img);
end
function new_img = Haar_dequantization(quanted_img,Δ)
    new_img = (quanted_img.*Δ)+Δ*.5;
end
function [dict,code] = encoding(img,q_level)
    img = reshape(img,[],1);
    q_level = reshape(q_level,[],1);
    probs = zeros(size(q_level,1),1);
    for k = 1:1:size(q_level,1)
        number = 0;
        for i = 1:1:size(img,1) % Calculate the frequency of quantized value
            if(img(i)==q_level(k))
               number = number + 1;
            end
        end
        probs(k) = number/(size(img,1));
    end
    [dict, ¬]= huffmandict(q_level,probs);
    code = huffmanenco(img,dict);
end
function decode = decoding(code,dict,array_width)
    decode = huffmandeco(code,dict);
    decode = reshape(decode,[],array_width);
```

(c) Draw the PDF diagram of the coefficients (7,8) (high-frequency) and (3,2) (low-frequency) for both DCT and Haar transformation (test image: cameraman.pgm), and compare the distributions.

As shown in the figure 14, for the low frequency coefficient(3,2), both the Haar transformation coefficient and DCT coefficient of image are around -10 to 10. But the DCT transform coefficient has less number of coefficient that is zero. On the other hand, as shown in figure 15, for the high frequency coefficient(7,8), both the Haar transformation coefficient and DCT coefficient of image are around -10 to 10. But the DCT transform coefficient has less number of coefficient that is zero. Compare to the coefficient in high and low frequency, there are more zero coefficient in high frequency, and less zero coefficient in low frequency, that means the image are combined mostly by low frequency basis for both Haar transform and DCT transform.

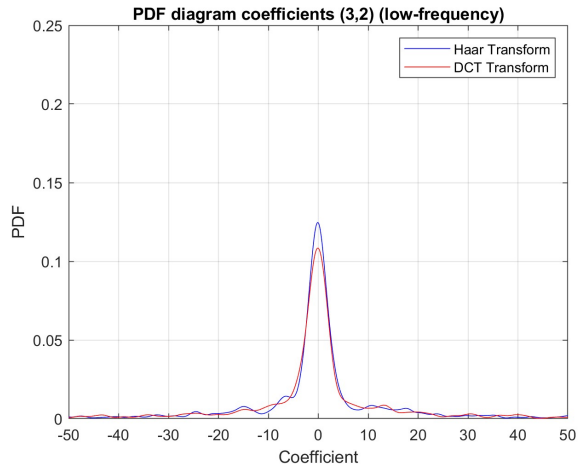


Figure 14: PDF diagram of coefficient (3,2)

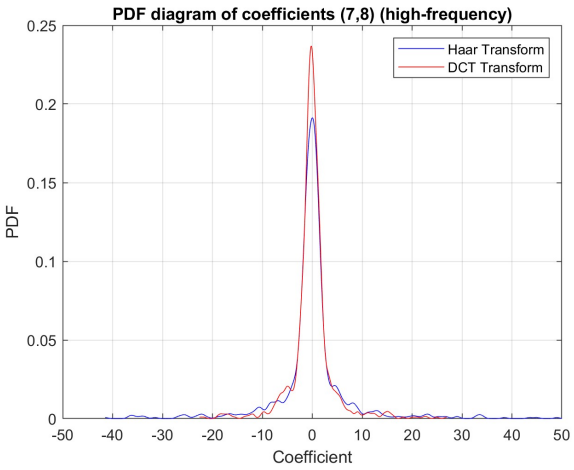


Figure 15: PDF diagram of coefficient (7,8)

(d) Draw a diagram showing the relationship between the SNR and the bit rate for both DCT and Haar transformation (test images: cameraman.pgm and butterfly.pgm).

The realationship between SNR and the bit rate for DCT and Haar transformation on the "cameraman.pgm" and "butterfly.pgm" are shown in the figure 16 and figure 17 respectively. For the "cameraman.pgm", when the bit rate is low (1 bit to 3 bits), the DCT transform is better than the Haar transform, when increasing the bit rate (4 bits to 8 bits), the value of SNR that Haar transform perform is better than the DCT tranform. For the "butterfly.pgm", the Haar transform and DCT transform have similar performance.

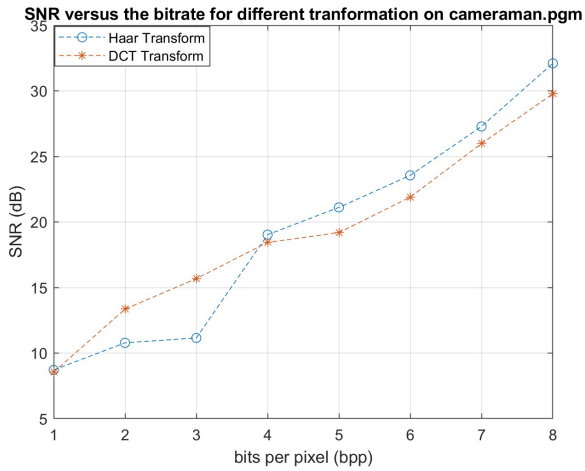


Figure 16: SNR and the bit rate for DCT and Haar transformation on cameraman.pgm

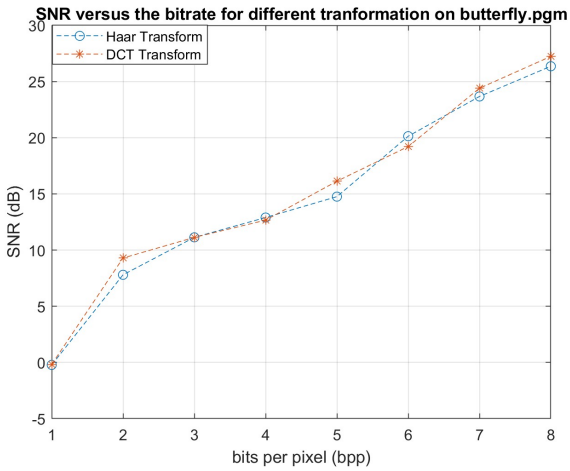


Figure 17: SNR and the bit rate for DCT and Haar transformation on butterfly.pgm

(e) Indicate the bit rates at which you think the Haar and DCT transformation give a 'good' visual quality. ('good' = difficult to see distortion) Motivate your answer.

The figure 18 and 19 shows the DCT transform and Haar transform of the camera.pgm. The figure 20 and 21 shows the DCT transform and Haar transform of the butterfly.pmg. For all the images mentioned above, I think a "good" visual quality image should be larger than 7 bit (SNR larger than 25 dB).

Figure 18: DCT transform with 1 to 8 bit per pixel on cameraman.pgm

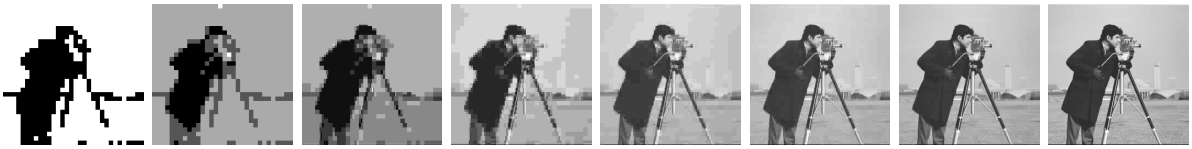


Figure 19: Haar transform with 1 to 8 bit per pixel on cameraman.pgm

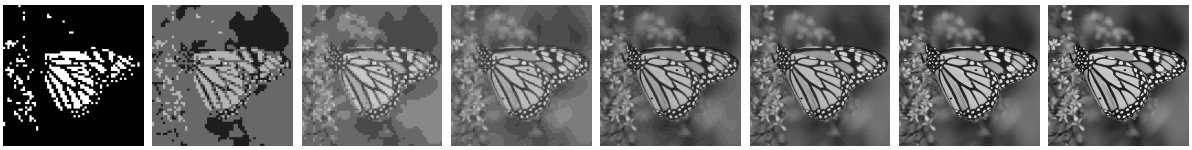


Figure 20: DCT transform with 1 to 8 bit per pixel on butterfly.pgm

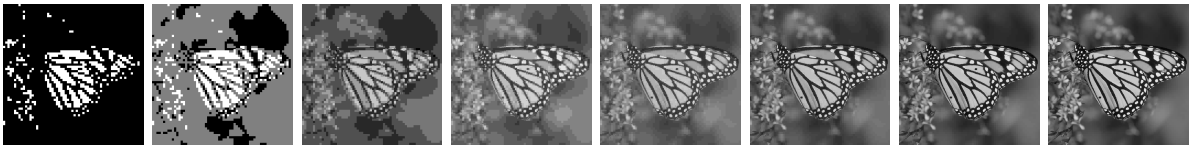


Figure 21: Haar transform with 1 to 8 bit per pixel on butterfly.pgm