EQ2330 Image and Video Processing
Project 2

Chenting Zhang
chzha@kth.se

Lukas Rapp
lrapp@kth.com

Shaotian Wu
shaotian@kth.se

December 7, 2022

Summary

The goal of this task is to implement two kinds of image transformation technology, DCT(Discrete Cosine Transform) and DWT(Discrete Wavelet transform). Then uniform quantizer with different step size is applied to two sets of transform coefficients respectively to investigate the relationship between bit-rate and peak signal-to-noise ratio and thus measure the quality of reconstructed image.

1 Introduction

Image compression is a method through which we can reduce the storage space of images, videos which will be helpful to increase storage and transmission process's performance. In image compression, we do not only concentrate on reducing size but also concentrate on doing it without losing quality and information of image. In this project, two types of image compression techniques are implemented. The first technique is based on Discrete Cosine Transform (DCT) and the second one is based on Discrete Wavelet Transform (DWT). The performance of both transformation is measured through the distortion and the bit rate-PSNR curve.

2 System Description

2.1 DCT-based Image Compression

2.1.1 DCT

DCT is the short of Discrete Cosine Transformation. It is the basic transformation used in JPEG 2000 standards. There are four types of DCT and one of the most widely used one is DCT-II, which is a orthonormal transformation and a unitary transformation. DCT transform of a size M×M block can be implemented by the following method:

$$y = AxA^T \tag{1}$$

where A is a $M \times M$ matrix whose elements is

$$a_{ij} = \alpha_i \cos(\frac{(2k+1)i\pi}{2M}) \tag{2}$$

with

$$\alpha_i = \begin{cases} \sqrt{\frac{1}{M}}, i = 0 \\ \sqrt{\frac{2}{M}}, \forall i > 0 \end{cases} \tag{3}$$

2.1.2 IDCT

IDCT is the inverse transform of DCT. It can be implemented by the following method:

$$x = A^T y A \tag{4}$$

The implementation of the DCT ind IDCT can be found in section A.2.1.

2.2 FWT-based Image Compression

2.2.1 The Two-Band Filter Bank

For the FWT implementation, We use the lifting scheme to design the filter bank. Lifting implementation of the 5/3 filter bank is illustrated in figure 1:

The forwarding lifting scheme of 5/3 filter bank consists of the following steps:

1. The splitting step, where the signal vector is separated into even and odd samples
2. The prediction step, associated with the predict operator $P_1(z) = -\frac{1+z}{2}$
3. The update step, associated with the update operator $U_1(z) = \frac{1+z^{-1}}{4}$
4. The scaling step, indicated by the scaling factors $\sqrt{2}$ and $\frac{1}{\sqrt{2}}$

For each pair of input samples, $2n$ and $2n+1$, the lifting equations of the analysis filter bank are written as follows:

$$\mathrm{HP}[n] = x[2n+1] - \frac{x[2n] + x[2n+2]}{2} \tag{5}$$

$$\mathrm{LP}[n] = \sqrt{2}(x[2n] + \frac{\sqrt{2}(\mathrm{HP}[n] + \mathrm{HP}[n-1])}{4}) \tag{6}$$

where x are the signal samples, HP denotes the high-frequency output coefficient, and LP denotes the low-frequency output coefficient. Note that we use a periodic extension of the signal to obtain signal values whose indices are less than 0 or larger than the signal length. In the MATLAB implemenation, this extension is achieved by $x[\mathrm{mod}(k, l)]$, where l is the length of x and mod denotes the modulo operation.

The corresponding synthesis step is the inverse transform of the analysis step. The input signal sets are the low frequency coefficients and high frequency coefficients. After the synthesis step, we obtain the reconstructed even samples and odd samples. Finally, by concatenating them, we obtain the reconstructed image. The liftings equations of the synthesis filter bank are written as follows:

$$\hat{x}[2n] = \frac{1}{\sqrt{2}}\mathrm{LP}[n] - \frac{\sqrt{2}\mathrm{HP}[n] + \sqrt{2}\mathrm{HP}[n-1]}{4} \tag{7}$$

$$\hat{x}[2n+1] = \sqrt{2}\mathrm{HP}[n] + \frac{\hat{x}[2n] + \hat{x}[2n+2]}{2} \tag{8}$$

The implementation of the FWT and IFWT can found in section 3.1[3].

2.2.2 The FWT

In this section, we use the Row-Column(RC) computation schedule to implement the 2D DWT. Firstly, we implement 1D FWT through x coordinate (to every column), thus obtaining the low band and high band. Then we implement 1D FWT through y coordinate (to every row) to these two bands respectively, thus obtaining four subbands which represent the approximation coefficients, horizontal coefficients, vertical coefficients and diagonal coefficients respectively. Lastly, we concatenate all the coefficients in a single image. This is the process of 1D-FWT.

After that, we do the 2D-FWT four times. Since most energy is concentrated in the approximation coefficients. In each step, we extract the approximation part as the input signal of the following 2D-FWT procedure. The whole procedure is illustrated in figure 2 [2].

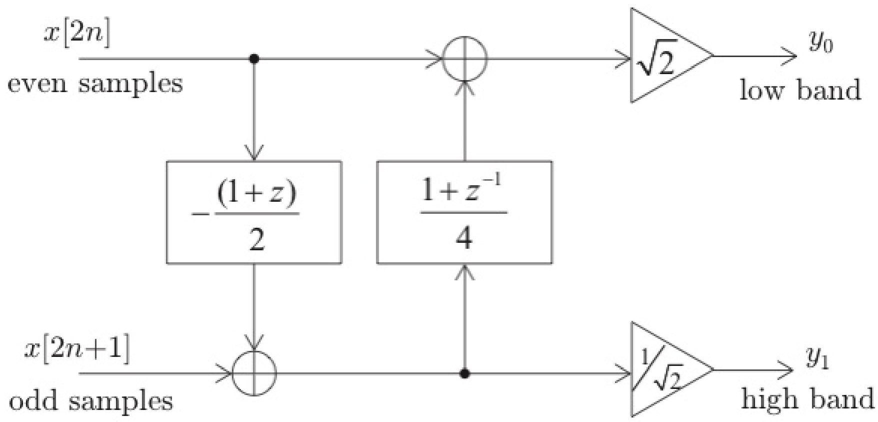

Figure 1: Lifting implementation of the 5/3 filter bank (Image source: [3])

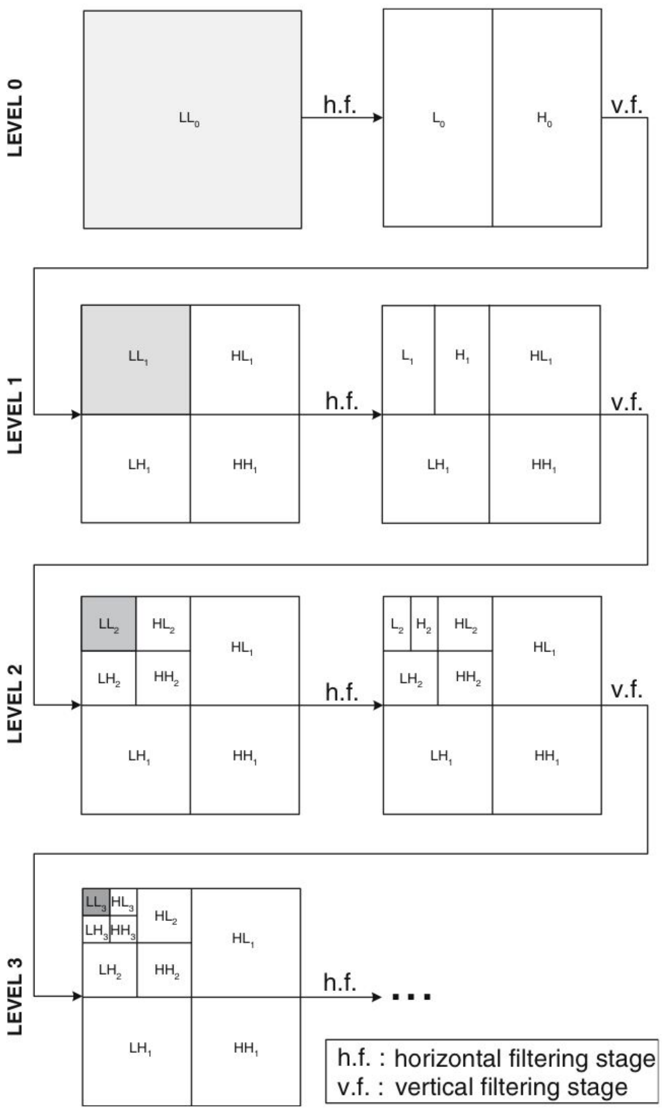

Figure 2: 2D FWT(Image source: [2])

2.3 Quantization and Performance analysis

2.3.1 Uniform Quantizer

Rounding a real number x to the nearest integer value forms a uniform quantizer. A mid-tread uniform quantizer has the following mapping format.

$$Q(x) = \Delta \cdot \left\lfloor \frac{x}{\Delta} + \frac{1}{2} \right\rfloor \tag{9}$$

where $\lfloor \cdot \rfloor$ denotes the floor operator, Δ denotes the quantization step size.

$Q(x)$ is the equalized value of real number x. By implementing this quantizer, all coefficient values are mapped to the midpoint of the range, and all equalization steps are equally spaced. Figure 3 shows the quantizer function for a step size $\Delta = 4$.

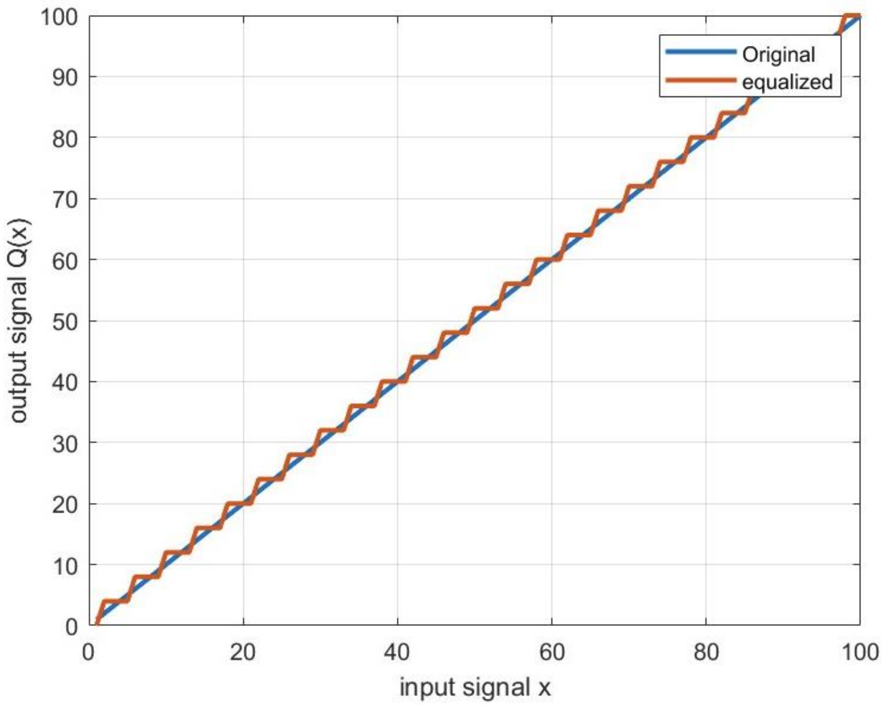

Figure 3: Mid-tread uniform equalizer(step size $\Delta = 4$)

2.3.2 Distortion and Bit-Rate Estimation

Given an image $\mathbf{I}$ with size $m \times n$, and the reconstructed output image $\mathbf{Q}$ after the equalization to the coefficients, the MSE (mean square error) is defined as follows:

$$\text{MSE} = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [\mathbf{I}(i,j) - \mathbf{Q}(i,j)]^2 \tag{10}$$

The PSNR(Peak Signal-to-Noise Ratio) is defined as:

$$\text{PSNR} = 10 \cdot \log_{10} \left(\frac{\text{MAX}_I^2}{\text{MSE}} \right) \tag{11}$$

Where MAX_I denotes the maximum pixel value of the image, in this case, $\text{MAX}_I = 255$.

For the DCT coefficients, different VLCs (Variable-length code), are uses depending on the position of the coefficient in the 8×8 blocks. That means all coefficients which are at position $(1,1)$ in their respective block use the same VLC and all coefficients at position $(2,1)$ use a different one and so on.

For the FWT coefficients, We uses a special code for each of the four subbands (approximation, horizontal, vertical, diagonal coefficients). The four subbands are encoded individually according to different types of VLC.

For the distortion analysis, we vary the quantizer step-size over the range $2^0, 2^1, 2^2, \cdots, 2^9$. For different step-sizes, the coefficient values are mapped to different representative levels. Average distortion is a measurement of the reconstruction quality, which is the mean square error between the original and the reconstructed image. Intuitively, if the step-size is smaller, there will be more representative levels, thus the average distortion is smaller which will lead to better reconstruction quality. The quality of the quantization can be predicted by the 6 dB rule. According to this rule of thumb, the SNR of a quantized signal increases with 6 dB per bit.

Then, to calculate the bit-rate, due to the fact that the shannon entropy is the lower bound of any variable-length code, we assume that the shannon entropy is a reasonable approximation of the ideal code word length of a VLC. Since different VLCs are used for different coefficient types, we calculated the entropy for each coefficient type individually. The bit-rate is the weighted average of these entropies.

3 Results

3.1 DCT

We are able to derive the A matrix of the blockwise 8×8 DCT:

$$A = \begin{bmatrix} 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 \\ 0.4904 & 0.4157 & 0.2778 & 0.0975 & -0.0975 & -0.2778 & -0.4157 & -0.4904 \\ 0.4619 & 0.1913 & -0.1913 & -0.4619 & -0.4619 & -0.1913 & 0.1913 & 0.4619 \\ 0.4157 & -0.0975 & -0.4904 & -0.2778 & 0.2778 & 0.4904 & 0.0975 & -0.4157 \\ 0.3536 & -0.3536 & -0.3536 & 0.3536 & 0.3536 & -0.3536 & -0.3536 & 0.3536 \\ 0.2778 & -0.4904 & 0.0975 & 0.4157 & -0.4157 & -0.0975 & 0.4904 & -0.2778 \\ 0.1913 & -0.4619 & 0.4619 & -0.1913 & -0.1913 & 0.4619 & -0.4619 & 0.1913 \\ 0.0975 & -0.2778 & 0.4157 & -0.4904 & 0.4904 & -0.4157 & 0.2778 & -0.0975 \end{bmatrix} \tag{12}$$

3.2 FWT

By implementing the FWT and the inverse FWT of scale 6, we could get the reconstructed image. By comparing the reconstructed image with its original version in figure 4, we could conclude that the lifting scheme allows for perfect reconstruction with arbitary scale since the MSE between the reconstructed image and original image equals zero.

Figure 4: original(left) versus reconstructed(right)

By applying the 2D-FWT four times to the image "`harbor`", the wavelet coefficients for scale 2 and 4 of the image `harbor` are illustrated as figure 5. We could conclude that the main energy is concentrated in the approximation coefficients (upper left).

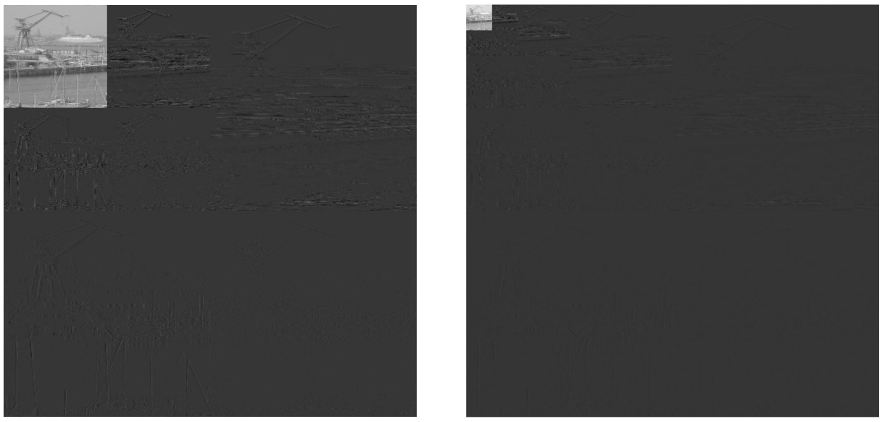

Figure 5: wavelet coefficients of harbor(scale 2 and 4)

3.3 Quantization and Performance analysis

3.3.1 DCT

First, we compared the mean squared error of the reconstructed image with the mean squared of the quantized coefficients. We observed that both distortions are always equal. To explain this behaviour, let $C \in \mathbf{R}^{512\times512}$ be the coefficients, $C_\mathrm{q} = C + E$ the quantized coefficients with additional quantization noise E and I and I_r the original and reconstructed image, respectively. Then, we have

$$I_\mathrm{r} = \mathrm{IDCT}(C) = \mathrm{IDCT}(C_\mathrm{q} + E) = \mathrm{IDCT}(C_\mathrm{q}) + \mathrm{IDCT}(E) = I + \mathrm{IDCT}(E), \tag{13}$$

because the IDCT is linear. Therefore, the distorition in the reconstructed image is the quantization noise that was transformed by the IDCT. Since the IDCT is unitary, the energy of this distortion is equal to the energy of E. Hence, the mean square errors are also equal.

The result of the distortion analysis for the DCT transformation are shown. Figure 6 shows the PSNR of the quantized image with respect to the bit-rate that is necessary to encode the quantized image. The PSNR increases with the bit rate. For bit rates above 1 bit per pixel, the slope of the curve is approximately 6 dB per bit. This behaviour can be motivated by the 6 dB per bit rule . Since the DCT is an orthonormal transformation, the quantization noise stays the same after back transformation and hence, the 6 dB per bit rule also holds for the PSNR of the reconstructed image.

3.3.2 FWT

It could be observed that the entropy of the approximation coefficients is always higher than that of the horizontal, vertical, diagonal coefficients since the low frequency part carries more information. For example, when step size equals 1. The entropy of approximation coefficients is 6.9455, while the entropy of horizontal, vertical, diagonal coefficients are 5.7850, 5.2386, 4.4330 respectively.

The result of the distortion analysis for the FWT transformation are shown in table 1 and 2. Figure 6 shows the PSNR of the quantized image with respect to the bit-rate that is necessary to encode the quantized image. The PSNR increases with the bit rate. For bit rates above 1 bit per pixel, the slope of the curve is approximately 6 dB per bit. The rate distortion performance of the compression with the FWT compression is similar to the DCT compression.

Table 1: Average distortion and MSE for the quantized images (Part 1)

Step size / MSE	2^0	2^1	2^2	2^3	2^4
Average distortion	0.1036	0.3956	1.4114	4.9249	16.3830
MSE(coefficients)	0.0842	0.3260	1.1874	4.1820	14.3805

Table 2: Average distortion and MSE for the quantized images (Part 2)

Step size / MSE	2^5	2^6	2^7	2^8	2^9
Average distortion	48.9955	128.5811	277.9681	447.3449	589.7516
MSE(coefficients)	45.2450	126.6955	297.3768	533.9498	780.9233

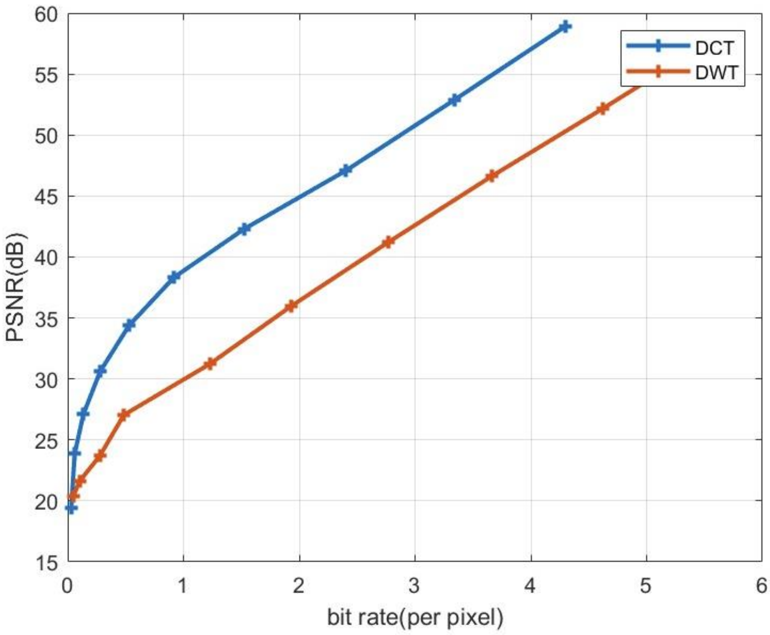

Figure 6: When using DCT and FWT transformation, PSNR of the quantized image plotted over the bit rate per pixel needed to encode the quantized image with a variable length code.

4 Conclusions

By implementing the algorithms of DCT and FWT respectively, we find that DCT and FWT transformation both allow for perfect reconstruction. When doing the performance analysis, it could be observed that the rate-PSNR curves of two transforms both adhere to the 6 dB per bit rule. By comparing the two rate-PSNR curves, it could be observed that the DCT outperforms the DWT due to its higher PSNR with respect to the same bit rate. However, in practice, FWT technique is more efficient than DCT technique in quality and efficiency wise.

A Appendix

A.1 Who Did What

Chenting Zhang: FWT implementation and performance analysis

Lukas Rapp: FWT implementation and performance analysis collaboratively with Chenting Zhang and part of DCT performance analysis

Shaotian Wu: DCT part, including DCT implement, DCT uniform quantizer and part of DCT performance analysis

A.2 MatLab code

A.2.1 Implementation DCT and IDCT using matrix multiplications

```
im=imread("images/boats512x512.tif");
im=double(im);
```

```
im_size=length(im);
block_size=8;

%Calculate A matrix in DCT
A=DCT_Mat(block_size);

%Implement of DCT to the image
s=im_size/block_size;
for i=0:s-1
    for k=0:s-1
        C=im(block_size*i+1:block_size*(i+1),block_size*k+1:block_size*(k+1));
        D=A*C*A';
        im_conv(block_size*i+1:block_size*(i+1),block_size*k+1:block_size*(k+1))=D;
    end
end

%Implement of IDCT to the image
for i=0:s-1
    for k=0:s-1
        C=im_conv(block_size*i+1:block_size*(i+1),block_size*k+1:block_size*(k+1));
        D=A'*C*A;
        im_rec(block_size*i+1:block_size*(i+1),block_size*k+1:block_size*(k+1))=D;
    end
end

function A=DCT_Mat(block_size)
for i=0:block_size-1
    for k=0:block_size-1
        s=i+1;
        t=k+1;
        A(s,t)=cos(i*(2*k+1)*pi/16);
        if i==0
            A(s,t)=A(s,t)*sqrt(1/8);
        else
            A(s,t)=A(s,t)*sqrt(2/8);
        end
    end
end
end
```

A.2.2 Uniform Quantizer and plot

```
step_size_list=[2^0,2^1,2^2,2^3,2^4,2^5,2^6,2^7,2^8,2^9];

xhead=biggest(im_conv,im_size);
for digit=0:9
    %Implement Uniform Quantizer
    stepsize=step_size_list(digit+1);
    for i=1:512
        for k=1:512
            im_conv_qua(i,k)=sign(im_conv(i,k))*stepsize*floor(abs(im_conv(i,k))/stepsize+0.5);
        end
    end

    %Plot the uniform quantizer
```

```
    x=-xhead:0.01:xhead;
    y=x;
    for i=1:length(x)
        y(i)=sign(x(i))*stepsize*floor(abs(x(i))/stepsize+0.5);
    end
    figure();
    plot(x,y);
    title('Quantizer function');
    xlabel('x');
    ylabel('(Q(x)');
end

function xhead=biggest(Mat,size)
xhead=0;
if length(Mat)~=size
    error("Wrong matrix size");
end
for i=1:size
    for k=1:size
        if xhead<Mat(i,k)
            xhead=Mat(i,k);
        end
    end
end
end
```

A.2.3 Performance Analysis

```

close all;
clear;

im=imread("images/boats512x512.tif");
im=double(im);

im_size=length(im);

block_size=8;
step_size_list=[2^0,2^1,2^2,2^3,2^4,2^5,2^6,2^7,2^8,2^9];

im_conv=DCT(im,im_size,block_size);

for index_step_size=1:length(step_size_list)
    disp(index_step_size);
    stepsize=step_size_list(index_step_size);
    im_conv_qua_int = sign(im_conv).*floor(abs(im_conv)/stepsize+0.5);
    im_conv_qua = stepsize * im_conv_qua_int;

    im_rec=ones(512);
    for i=0:63
        for k=0:63
            C=im_conv_qua(8*i+1:8*(i+1),8*k+1:8*(k+1));
            D=idct2(C,[8 8]);
            im_rec(8*i+1:8*(i+1),8*k+1:8*(k+1))=D;
        end
    end

    dist_fig=sum((im-im_rec).^2, 'all')/512^2;
    disp(dist_fig);

```

```
    PSNR=10*log10(255^2/dist_fig);
    PSNR_list(index_step_size)=PSNR;

    entropy_im_conv_qua = 0;
    for m = 1:8
        for n = 1:8
            % Extracts all coefficients which are at position m, n in each
            % block
            coefficients_m_n = im_conv_qua(m:8:end, n:8:end);

            % Get probabilities for coefficients
            [numbers, values] = groupcounts(reshape(coefficients_m_n, [], 1));
            probs = numbers / sum(numbers);

            % Calculate entropy of all coefficients at position m, n in each
            entropy_coefficients_m_n = sum(-log2(probs) .* numbers, 'all');
            entropy_im_conv_qua = entropy_im_conv_qua + entropy_coefficients_m_n;
        end
    end
    bitr = entropy_im_conv_qua / (size(im_conv_qua,1) * size(im_conv_qua, 2));

    bitr_list(index_step_size)=bitr;
end

figure();
plot(bitr_list,PSNR_list,'+-','LineWidth',2);


% correct
function A=DCT_Mat(block_size)
for i=0:block_size-1
    for k=0:block_size-1
        s=i+1;
        t=k+1;
        A(s,t)=cos(i*(2*k+1)*pi/16);
        if i==0
            A(s,t)=A(s,t)*sqrt(1/8);
        else
            A(s,t)=A(s,t)*sqrt(2/8);
        end
    end
end
end

% correct
function im=DCT(im,im_size,block_size)
if mod(im_size,block_size)
    error("Wrong block size");
end
s=im_size/block_size;
for i=0:s-1
    for k=0:s-1
        C=im(block_size*i+1:block_size*(i+1),block_size*k+1:block_size*(k+1));
        D=dct2(C,[block_size block_size]);
        im(block_size*i+1:block_size*(i+1),block_size*k+1:block_size*(k+1))=D;
    end
```

```
end
end
```

A.2.4 FWT

```

function DWT=FWT2(im, layer, max_layer)
% layer:always start with 0
% max_layer: the number of transformations that are applied
    disp(size(im));          % display in the command line
    if layer >= max_layer
        DWT = im;
    else
        [a,d_hor, d_ver, d_diag] = FWT2_step(im); % one-step 2-D FWT

        a = FWT2(a, layer + 1, max_layer);      % recursive, using the previou approximation as input

        DWT = cat(1,[a d_hor],[d_ver d_diag]);  % concatenate coefficients in a single image

%          approximation | horizontal detail
%       ------------------------------------
%         vertical detail | diagonal detail
    end
end


function [a,d_hor, d_ver, d_diag] = FWT2_step(im)
    for n = 1:size(im,2)   %apply to single columns
        [a_col(:,n), d_col(:,n)] = analysis_step(im(:,n));
    end

    for m = 1:size(a_col,1)
        [a(m,:), d_ver(m,:)] = analysis_step(a_col(m,:));   %get LP and vertical details
    end

    for m = 1:size(d_col,1)
        [d_hor(m,:), d_diag(m,:)] = analysis_step(d_col(m,:));  %get horizontal and diagonal details
    end
end


function [low_band, high_band] = analysis_step(signal)
% 5/3 lifting scheme
    even_samples = signal(2:2:end);
    odd_samples = signal(1:2:end);

    odd_samples_tmp = zeros(size(odd_samples));
    for time=1:length(odd_samples)
        odd_samples_tmp(time) = odd_samples(time) - 1/2 * (even_samples(time) + even_samples(mod_time(time + 1, length(even_samples))));
    end

    low_band = zeros(size(even_samples));
    for time=1:length(even_samples)
        low_band(time) = sqrt(2) * (even_samples(time) + 1/4 * (odd_samples_tmp(time) + odd_samples_tmp(mod_time(time - 1,
```

```
        length(odd_samples_tmp)))));
    end

    high_band = 1/sqrt(2) * odd_samples_tmp;
end


function new_time = mod_time(time, L)
    new_time = mod(time - 1, L) + 1;
end
```

A.2.5 IFWT

```
function im_rec=FWT2_inv(fwt, layer)
    for l = (layer-1):-1:0
        s = size(fwt, 1) / 2^l;
        fwt(1:s, 1:s) = FWT2_inv_step(fwt(1:s, 1:s));
    end
    im_rec = fwt;
end


function im_rec = FWT2_inv_step(fwt)
    s = size(fwt, 1) / 2;

    a = fwt(1:s, 1:s);
    d_hor = fwt(1:s, (s+1):end);
    d_ver = fwt((s+1):end, 1:s);
    d_diag = fwt((s+1):end, (s+1):end);

    d_col = zeros(size(a,1), 2*size(a,2));
    for m = 1:size(a, 1)   %apply to rows
        d_col(m,:) = synthesis_step(d_hor(m,:), d_diag(m,:));   %get LP and vertical details
    end

    a_col = zeros(size(a,1), 2*size(a,2));
    for m = 1:size(a,1)   %apply to rows
        a_col(m,:) = synthesis_step(a(m,:), d_ver(m,:));   %get LP and vertical details
    end

    im_rec = zeros(2*size(a, 1), 2*size(a, 2));
    for n = 1:size(fwt,2)   %apply to single columns
        im_rec(:, n) = synthesis_step(a_col(:, n), d_col(:, n));
    end
end


function signal_rec = synthesis_step(low_band, high_band)
    even_samples = zeros(length(low_band), 1);
    odd_samples = zeros(length(low_band), 1);

    low_band_sc = 1/sqrt(2) * low_band;
    high_band_sc = sqrt(2) * high_band;

    for time = 1:length(low_band)
        even_samples(time) = low_band_sc(time) - 1/4 * (high_band_sc(time) + high_band_sc(mod_time(time - 1, length(high_band_sc))));
    end
```

```
    for time = 1:length(high_band)
        odd_samples(time) = high_band_sc(time) + 1/2 * (even_samples(time) +
            ↪ even_samples(mod_time(time + 1, length(even_samples))));
    end

    signal_rec = zeros(2*length(low_band), 1);
    signal_rec(1:2:end) = odd_samples;
    signal_rec(2:2:end) = even_samples;
end
```

A.2.6 Equalization, performance analysis

```
clear;
close all;
clc;
im=imread ('harbour512x512.tif') ;
imshow(im);
im = double(im);

DWT = FWT2(im, 0, 2);   %coefficient
DWT_img = mat2gray(DWT);
imshow(DWT_img,[]);   %show scale 4 DWT coefficients

%% mid-tread quantizer
step_size = [2^0,2^1,2^2,2^3,2^4,2^5,2^6,2^7,2^8,2^9];

% 3-D matrix(2-D coefficients for different step_size)
DWT_q = zeros(size(DWT,1),size(DWT,2),length(step_size));
for k = 1:length(step_size)
DWT_q(:,:,k) = mid_tread(DWT, step_size(k));
end

for k = 1:length(step_size)
Appro(:,:,k) = DWT_q(1:256,1:256,k);
Horizontal(:,:,k) = DWT_q(1:256,257:512,k);
Vertical(:,:,k) = DWT_q(257:512,1:256,k);
Diagonal(:,:,k) = DWT_q(257:512,257:512,k);
end

im_rec = zeros(size(im,1),size(im,2),length(step_size));
for i = 1:length(step_size)
    im_rec(:,:,i) = FWT2_inv(DWT_q(:,:,i), 4);
end
% test when k = 4
figure;
im_rec_temp = FWT2_inv(DWT_q(:,:,4),4);
imshow(uint8(im_rec_temp));

%% calculate the MSE
mse_coef = zeros(1,length(step_size));
mse_image = zeros(1,length(step_size));

for i = 1:length(step_size)
    mse_coef(i) = MSE(DWT,DWT_q(:,:,i));
end

for i = 1:length(step_size)
    mse_image(i) = MSE(im,im_rec(:,:,i));
end
```

```

%% bit-rate versus PSNR
PSNR_image = zeros(1,length(step_size));
for i = 1:length(step_size)
    PSNR_image(i) = 10.*log10(255^2./mse_image(i));
end

Appro_vect = reshape(Appro, 1,[],10);
Horizontal_vect = reshape(Horizontal,1,[],10);
vertical_vect = reshape(Vertical,1,[],10);
Diagonal_vect = reshape(Diagonal,1,[],10);

num_bins_appro = zeros(1,length(step_size));
num_bins_hori = zeros(1,length(step_size));
num_bins_vert = zeros(1,length(step_size));
num_bins_diag = zeros(1,length(step_size));

for i = 1:length(step_size)
    num_bins_appro(i) = ceil((max(Appro_vect(:,:,i))-min(Appro_vect(:,:,i)))./step_size(i));
    num_bins_hori(i) = ceil((max(Horizontal_vect(:,:,i))-min(Horizontal_vect(:,:,i)))./step_size(i));
    num_bins_vert(i) = ceil((max(vertical_vect(:,:,i))-min(vertical_vect(:,:,i)))./step_size(i));
    num_bins_diag(i) = ceil((max(Diagonal_vect(:,:,i))-min(Diagonal_vect(:,:,i)))./step_size(i));
end


% bit rate(ideal code word length of a VLC) entropy
entropy = cal_pro(Appro_vect,Horizontal_vect,vertical_vect,Diagonal_vect,step_size);
bitrate = entropy;
figure;
plot(bitrate, PSNR_image, '+-', 'LineWidth',2);
title('bit rate versus PSNR(dB)');
grid on;
xlabel('bit rate(per pixel)');
ylabel('PSNR(dB)');


function quantized = mid_tread(input_signal, step_size)
 quantized = input_signal;
     for i = 1 : size(quantized,1)
         for j = 1 : size(quantized,2)
             quantized(i,j) = step_size.*floor(quantized(i,j)./step_size+1/2);
         end
     end
end

function value = MSE(input, output)
temp = 0;
    for i = 1:size(input,1)
        for j = 1:size(input,2)
            temp = temp + (output(i,j)-input(i,j)).^2;
        end
    end
value = temp./(size(input,1).*size(input,2));
end

```

```matlab

function entropy = cal_pro(Appro_vect,Horizontal_vect,vertical_vect,Diagonal_vect,step_size)
num_bins_appro = zeros(1,length(step_size));
num_bins_hori = zeros(1,length(step_size));
num_bins_vert = zeros(1,length(step_size));
num_bins_diag = zeros(1,length(step_size));

for i = 1:length(step_size)
    num_bins_appro(i) = ceil((max(Appro_vect(:,:,i))-min(Appro_vect(:,:,i)))./step_size(i));
    num_bins_hori(i) = ceil((max(Horizontal_vect(:,:,i))-min(Horizontal_vect(:,:,i)))./step_size(i));
    num_bins_vert(i) = ceil((max(vertical_vect(:,:,i))-min(vertical_vect(:,:,i)))./step_size(i));
    num_bins_diag(i) = ceil((max(Diagonal_vect(:,:,i))-min(Diagonal_vect(:,:,i)))./step_size(i));
end


% calculate the probability
    pro_appro_1 = histcounts(Appro_vect(:,:,1),num_bins_appro(1),'Normalization','probability');
    pro_hori_1 = histcounts(Horizontal_vect(:,:,1),num_bins_hori(1),'Normalization','probability');
    pro_vert_1 = histcounts(vertical_vect(:,:,1),num_bins_vert(1),'Normalization','probability');
    pro_diag_1 = histcounts(Diagonal_vect(:,:,1),num_bins_diag(1),'Normalization','probability');
    entropy_appro_1 = -sum(pro_appro_1.*log2(pro_appro_1+eps));     % +eps cus if pro = 0, it will lead to NAN
    entropy_hori_1 = -sum(pro_hori_1.*log2(pro_hori_1+eps));
    entropy_vert_1 = -sum(pro_vert_1.*log2(pro_vert_1+eps));
    entropy_diag_1 = -sum(pro_diag_1.*log2(pro_diag_1+eps));
    entropy_1 = 1/4*(entropy_appro_1+entropy_hori_1+entropy_vert_1+entropy_diag_1);
... repeat 10 times ...
    entropy=[entropy_1,entropy_2,entropy_3,entropy_4,entropy_5,entropy_6,entropy_7,entropy_8,entropy_9,entropy_10];
```

References

[1] Rafael C. Gonzalez and Richard E. Woods, *Digital Image Processing*, Prentice Hall, 2nd ed., 2002

[2] Angelopoulou, M. E., Masselos, K., Cheung, P. Y. K., & Andreopoulos, Y. *Implementation and comparison of the 5/3 lifting 2D discrete wavelet transform computation schedules on FPGAs*, Journal of Signal Processing Systems, 51(1), 3–21. https://doi.org/10.1007/s11265-007-0139-5,

[3] EQ2330 Image and Video Processing, *Project 2: Image Compression*, 2022