Lab 1 - Image Processing EEE412

Author: Zhipeng Ye Student ID: 1926908

Faculty: MSc Multimedia Telecommunications Date: 28th September 2019

Task 1

Download from ICE the image lenna512color.bmp and save it as a file on your PC as lenna512Color.bmp. Use the functions imread to load the image Lenna512Color.bmp into Matlab.

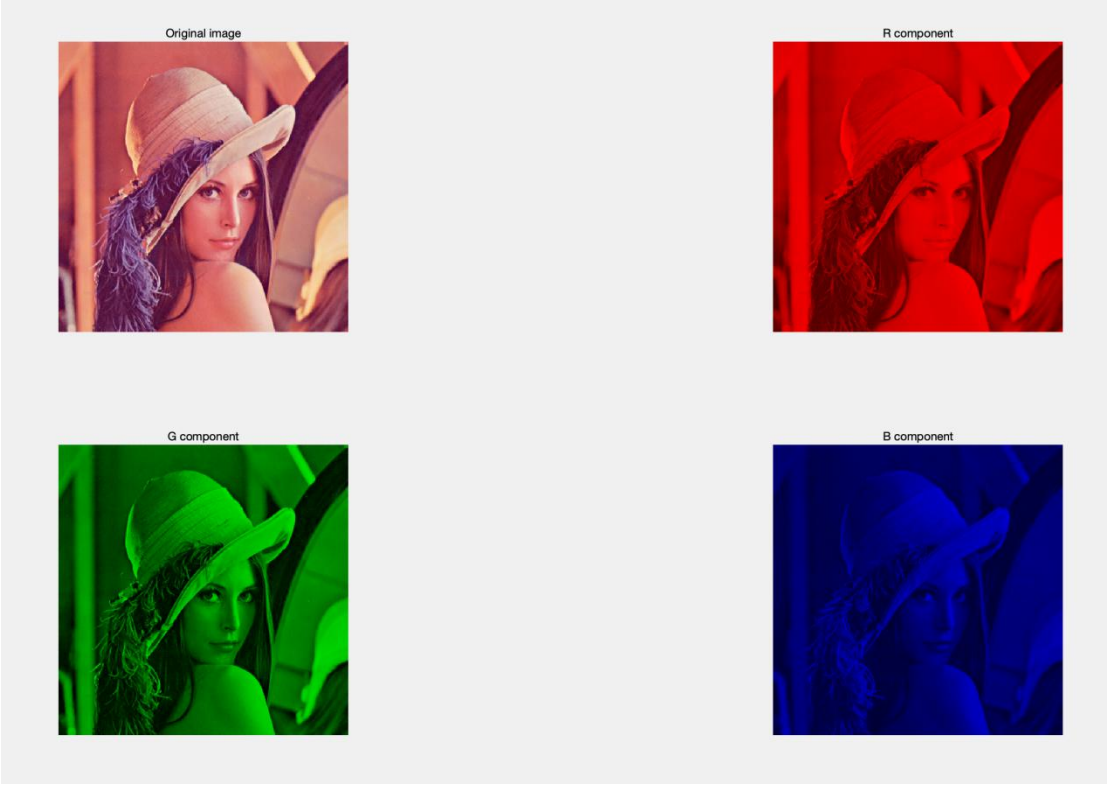
(1) Display the image with the function image, and shows three images of RGB components.

Code Block

```
% task 1
% load image
image = imread('lenna512color.bmp');

% create 3 copies of orginal image
RGB_R_component=image;
RGB_G_component=image;
RGB_B_component=image;

% remove 2 component respectively
RGB_R_component(:,:,[2 3])=0;
RGB_G_component(:,:,[1 3])=0;
RGB_B_component(:,:,[1 2])=0;

% show images respectively
subplot(2,2,1)
imshow(image)
title('Original image')

subplot(2,2,2)
imshow(RGB_R_component)
title('R component')

subplot(2,2,3)
imshow(RGB_G_component)
title('G component')
```

```
subplot(2,2,4)
imshow(RGB_B_component)
title('B component')
```

Result



Describe and analysis

As we can see from the result, RGB image can be divided into 3 components(Red, Green, Blue). Besides, we can found RGB image is a N*M*3 matrix.

(2) Change the color space into HSI, and show three images of HSI components.

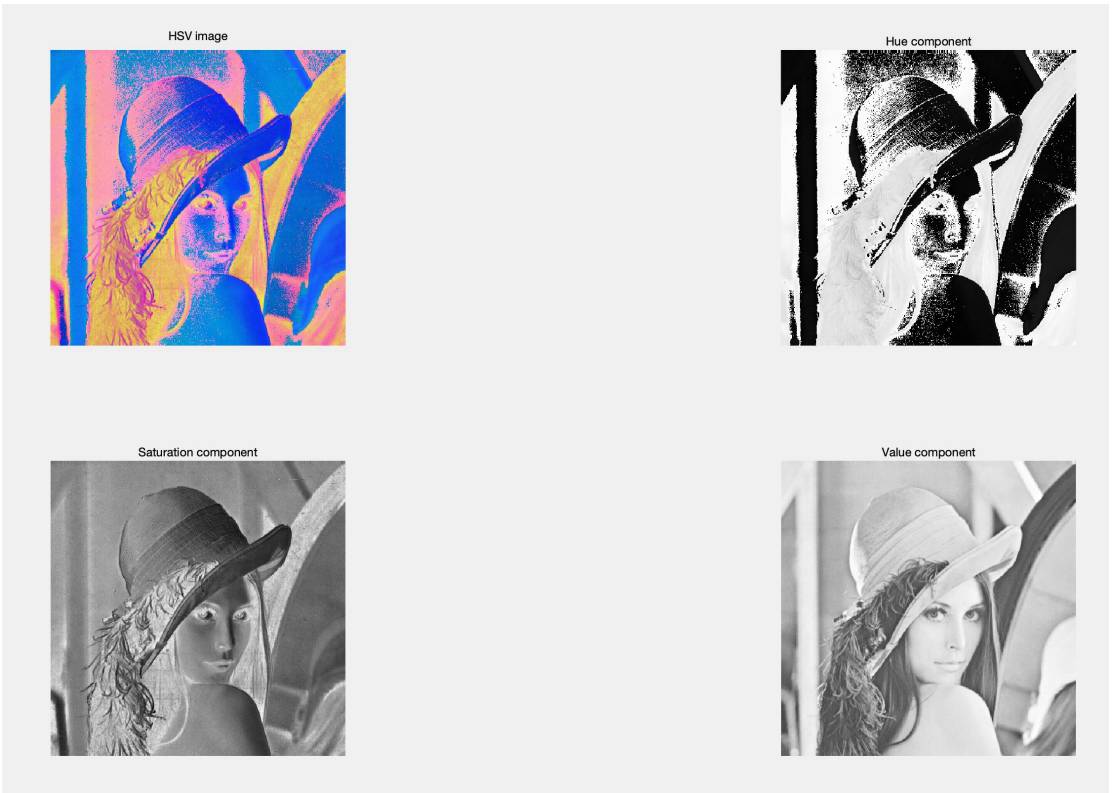
RGB to HSV

Code Block

```
% Firstly, I can't find the tool for transformation of rgb to hsi, But I find the tool for transformation of rgb to hsv.
% Therefore, I code about transformation of rgb to hsv

image=imread('lenna512color.bmp');
image_hsv=rgb2hsv(image);
```

```
subplot(2,2,1);
imshow(image_hsv);
title('HSV_ image');

% h(Hue) component
subplot(2,2,2);
imshow(image_hsv(:,:,1));
title('Hue component');

% s(Saturation) component
subplot(2,2,3);
imshow(image_hsv(:,:,2));
title('Saturation component');

% v(Value) component
subplot(2,2,4);
imshow(image_hsv(:,:,3));
title('Value component');
```

Result



Describe and analysis

In HSV space, H means Hue, S represents Saturation, V means Value. But when I am search on the internet. I found there are some difference between HSV and HSI space. HSL and HSV are quite similar color spaces. The difference is that in HSV space to get white color you should set Saturation to "0". But in HSL space at L=1 you get white regardless the saturation value. So I code a new vision of this question following by this equations.

$$I = \frac{1}{3}(R + G + B)$$

$$S = 1 - \frac{3[\min(R, G, B)]}{(R + G + B)}$$

$$H = \begin{cases} \theta, & B \leq G \\ 360 - \theta, & B > G \end{cases}$$

$$\theta = arccos\{\frac{\frac{1}{2}[(R - G) + (R - B)]}{[(R - G)^2 + (R - B)(G - B)]^{\frac{1}{2}}}\}$$

RGB to HSI

Code Block

```
% Firstly, I can't find the tool for transformation of rgb to hsi, But I find the tool for transformation of rgb to hsv.
% Therefore, I code about transformation of rgb to hsv.
% After learning about the theory of HSI, I create a new vision of this question.

image=imread('lenna512color.bmp');

% convert RGB to HSV
% image_hsv=rgb2hsv(image);

% convert RGB to HSI
R_component=im2double( image(:,:,1));
```

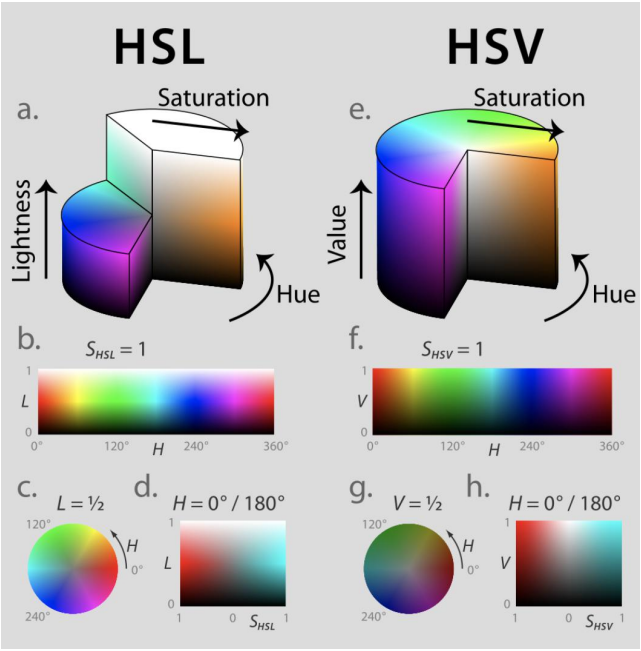
```
G_component=im2double(image(:,:,2));
B_component=im2double(image(:,:,3));

[m, n, q] = size(image);
H=zeros(m, n);
S=H;
for i=1:m
    for j=1:n
        numerator = 0.5*(R_component(i,j)-G_component(i,j)+R_component(i,j)-B_component(i,j));
        denominator = sqrt( ((R_component(i, j) - G_component(i, j))^2 + (R_component(i, j) - B_component(i, j))
* ( G_component(i,j)- B_component(i,j) )));
        theta = acos(numerator/denominator)*180/pi;
        if( B_component(i,j)<=G_component(i,j))
            H(i,j)=theta;
        else
            H(i,j)= 360-theta;
        end
        min_1 = min(G_component(i,j),G_component(i,j));
        min_1 = min(min_1, B_component(i,j));
        S(i,j) = 1 - 3/(R_component(i,j) + G_component(i, j) + B_component(i, j))*min_1;

    end
end

I = (R_component+G_component+B_component)/3;

image_hsi=cat(3, im2uint8(H),im2uint8(S),im2uint8(I));

subplot(2,2,1);
imshow(image_hsi);
title('HSI_ image');

% h(Hue) component
subplot(2,2,2);
imshow(image_hsi(:,:,1));
title('Hue component');

% s(Saturation) component
subplot(2,2,3);
imshow(image_hsi(:,:,2));
title('Saturation component');

% I(Intensity) component
subplot(2,2,4);
```

```
imshow(image_hsi(:,:,3));
title('Intensity component');
```

Result

Describe and analysis

As we can see from two results of HSV, HSI, there are some differences between them. Here is a picture describing the distinction.



(3) Change this image into gray level, and show the gray image.

Code Block

```
image = imread('lenna512color.bmp');

% I found a library to convert rgb into gray space.
Image_gray= rgb2gray(image);
imshow(Image_gray)

% Here is a implement without library following by this formula
% Gray = R*0.299 + G*0.587 + B*0.114

R=image(:,:,1);
G=image(:,:,2);
B=image(:,:,3);
Gray = R*0.299 + G*0.587 + B*0.114;

imshow(Gray);
```

Result

Describe and analyze

It is found that we can use this formula (Gray = R*0.299 + G*0.587 + B*0.114) to calculate the gray space.

(4) Change this image into binary level, and show the binary image.

Code Block

```
% convert rgb into binary space

image=imread('lenna512.bmp');

subplot(2,2,1);
imshow(image);
title('original picture');

% use library to implement
Image_binary=im2bw(image)
subplot(2,2,2);
imshow(Image_binary);
title('lib to implement')

% use 256/2-1=127 as a threshold to convert
Image_binary = image>127;
subplot(2,2,3);
imshow(Image_binary);
title('127 threshold');

% use mean as a threshold to convert
[m,n]=size(image);
Image_binary = im2double(image) > sum(sum(im2double(image)))/(m*n);
subplot(2,2,4)
imshow(Image_binary)
title('mean threshold');
```

Result

Describe and analyze

As we can see, when we are converting images to binary pictures, we must determine a specific threshold. We will get different images by using different threshold. In this experiment, I use library, 127 and mean to calculate respectively.

(5) Describing your founding from the above tasks by comparing different shown images?

I have written the answer of this question in each questions (1-4) . You can find detailed information in previous questions.

Task 2

Write a function to measure the Peak Signal to Noise Ratio (PSNR) between two gray images in dB. For the peak value use 255.

$$PSNR(db) = 10log_{10}(\frac{255^2}{mse})$$

Where mse is the mean square error, and it is evaluated as:

$$mse = \frac{1}{N}\sum_{ri}\sum_{ci}(im(ri,ci) - im_2(ri,ci))^2$$

Code Block

```
% calculate the PSNR of gray image , 2 dimension matrix
function PSNR = CalculatePSNR(image_1, image_2)
    [m, n] = size(image_1);
    N = m * n;
    mse = sum( sum( (image_1–image_2).^2) )/N;
    PSNR=10 * log10( 255^2 / mse);
end
```

Result

Because the question doesn't ask us to calculate specific problems, so there is no result about this question. Task 3 need this small function to calculate PSNR.

Describe and analyze

Firstly, we can see PSNR is defined as a ratio and the outcome be taken by log10 to control the range. Furthermore, When MSE is small, the PSNR will be larger.

Task 3

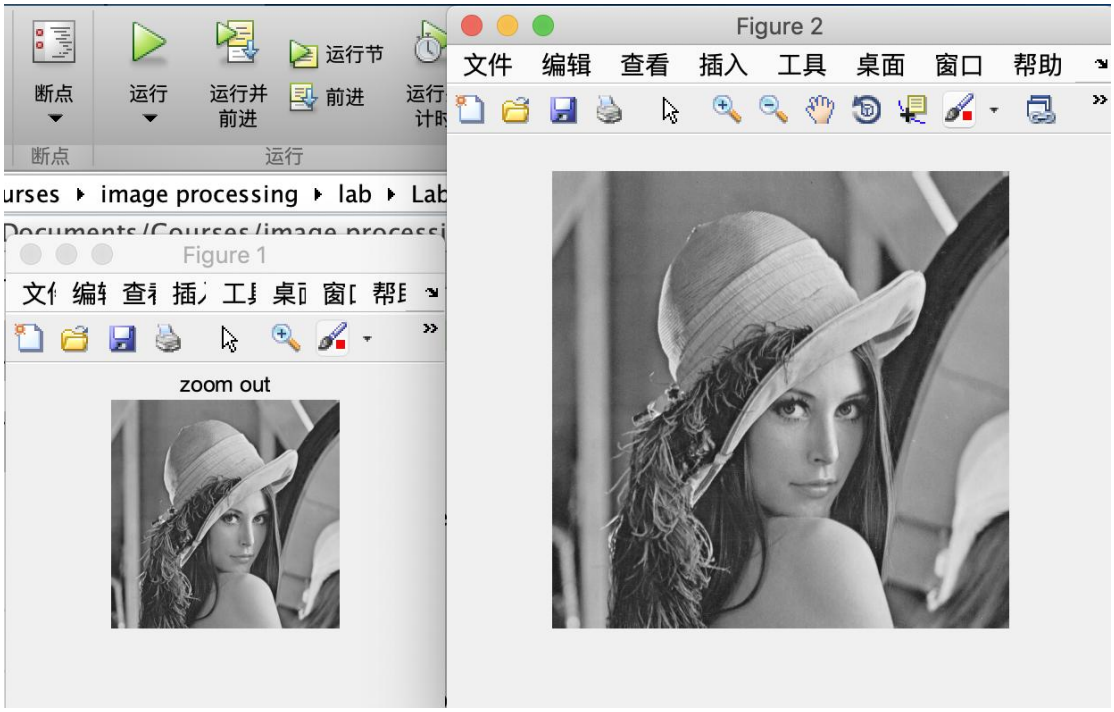
In this task, we use the monochrome image Lenna (i.e., lenna512.bmp in ICE) to do the following sub tasks, and let's call the original image Lenna as I0.

(1) I0 -> down-sampling to I1 with 1/2 size of I0 (both horizontally and vertically) using mean value **(programing it by yourself)**. Display it and explain your founding in the report;

Code Block

```
% The vision of my matlab is R2017, I can't use the mean function. So I
% implement an algorithm.

function narrow_matrix=narrow_down(matrix)
   [m,n]=size(matrix);
   matrix_double=im2double(matrix);
   narrow_matrix=zeros(m/2,n/2);
   for i=1:2:m
       for j=1:2:n
%               calculate the mean of neighbor
           mean_value=(matrix_double(i,j)+matrix_double(i,j+1)+matrix_double(i+1,j)+matrix_double(i+1,j+1))/4;
           narrow_matrix((i+1)/2,(j+1)/2)=mean_value;
       end
   end
end

% mean zoom out
image_gray=imread('lenna512.bmp');
image_narrow=narrow_down(image_gray);
figure(1)
imshow(image_narrow)
title('zoom out')
```

Result



Describe and analyze

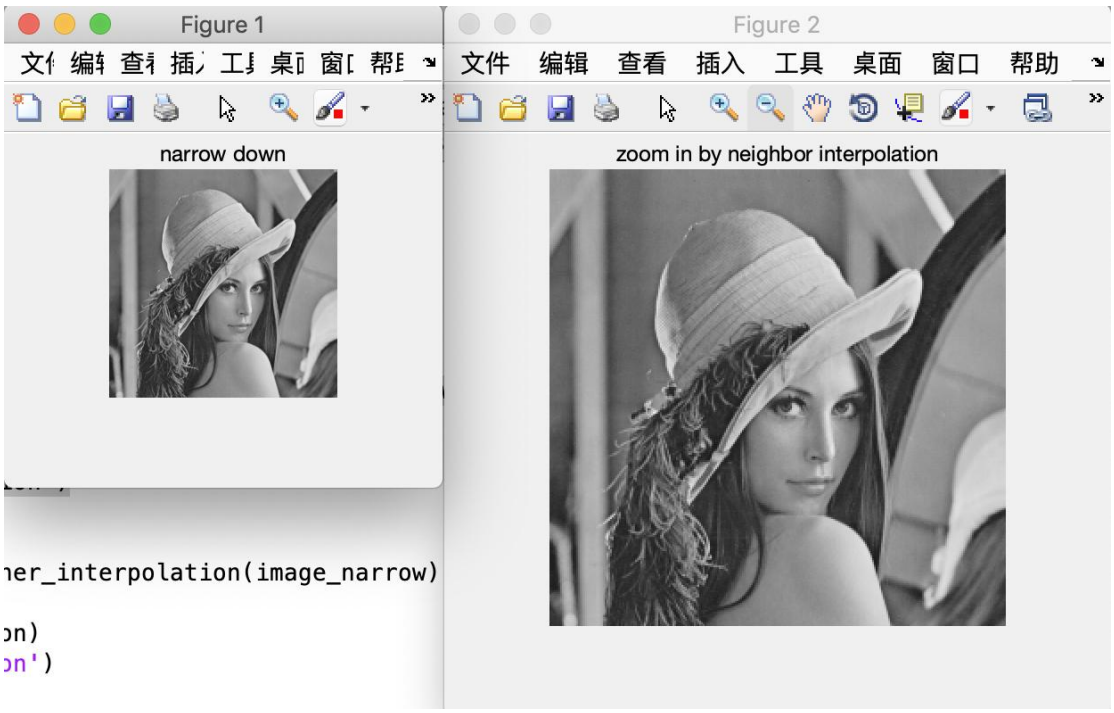
I found that when we down-sampling to 1/2 size, we must calculate the average of neighbor pixels and consider the situation of edge when assign the value (index = (i +1)/2).

(2) I1-> up-sampling to I1' with the same size of I0 using nearest neighbor interpolation **(programing it by yourself)**. Display it and compare to the original image. Explain your founding in the report.

Code Block

```
% neighbor interpolation
function zoom_out_image=neighbor_interpolation(image)
    [m,n]=size(image);
    zoom_out_image=zeros(2*m,2*n);
    for i=1:2:2*m
        for j=1:2:2*n
            zoom_out_image(i:i+1,j:j+1)=image((i+1)/2,(j+1)/2);
        end
    end
end

image_zoom_neighbor_interpolation=neighbor_interpolation(image_narrow);
figure(2)
imshow(image_zoom_neighbor_interpolation)
title('zoom in by neighbor interpolation')
figure(1)
imshow(image_narrow)
title('narrow down')
```

Result



Describe and analyze

It is simple to implement this algorithm, we can use 1 pixel to fill out 4 pixels. In addition, we can use this technique, zoom_out_image(i:i+1,j:j+1)=image((i+1)/2,(j+1)/2), instead of assign 4 s respectively to simplify this process.

(3) Repeat the (b) with bilinear interpolation and bicubic interpolation (you can use Matlab function directly), respectively.

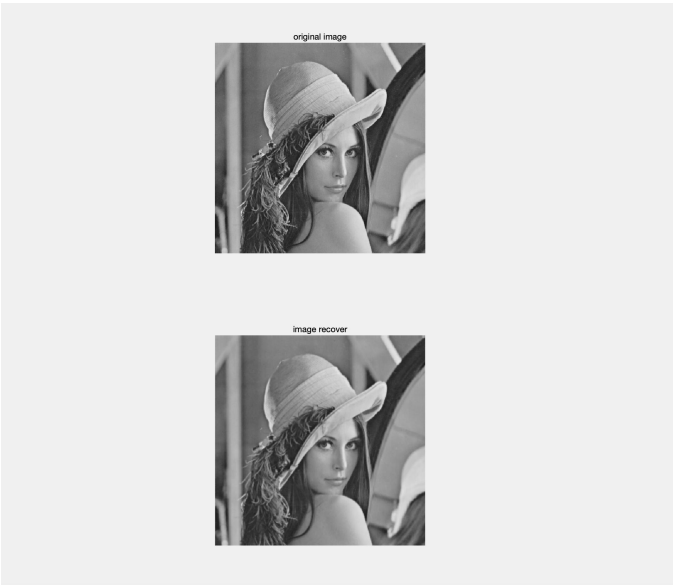
Bilinear interpolation

Code Block

```
function worked_matrix = biliner_interpolation_1(image, scale)

% create return matrix
worked_matrix = zeros(size(image)*scale);
worked_matrix = im2double(worked_matrix);

padding_matrix = zeros(size(image) + 2);
padding_matrix = im2double(padding_matrix);
padding_matrix(2 : end-1, 2 : end-1) = image;

% padding processing
padding_matrix([1 end],:) = padding_matrix([2 end-1],:);
padding_matrix(:,[1 end]) = padding_matrix(:, [2 end-1]);

[m, n] = size(worked_matrix);

for i = 1: m
    for j = 1: n

        x = i/scale;
        y = j/scale;

%           compute this original index
        i_original = floor(x);
        j_original = floor(y);

%           computer the coefficient
        u=x-i_original;
        v=y-j_original;

        i_original=i_original+1;
```

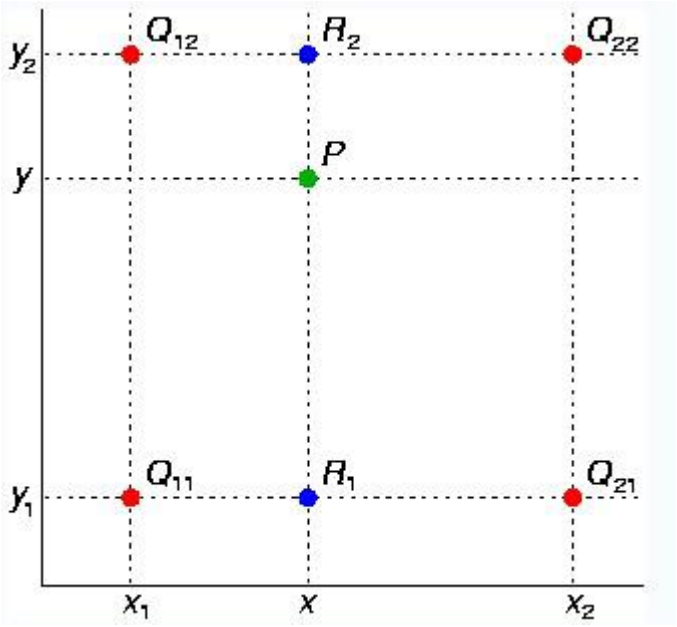
```
        j_original=j_original+1;

%           follow by this formula
        worked_matrix(i, j) = (1-u)*(1-v)*padding_matrix(i_original,j_original) +...
            (1-u)*v*padding_matrix(i_original,j_original+1) +...
            u*(1-v)*padding_matrix(i_original+1,j_original) +...
            u*v*padding_matrix(i_original+1,j_original+1);

    end
end

% convert double into uin8 type
worked_matrix = uint8(worked_matrix);

image = imread('lenna512.bmp');
image_narrow = narrow_down(image);

image_recover = biliner_interpolation_1(im2uint8(image_narrow), 2);

subplot(2,1,1)
imshow(image)
title('original image')

subplot(2,1,2)
imshow(image_recover)
title('image recover')
```

Result



Describe and analyze

It is a little difficult to implement bilinear interpolation. There are some several points that we must to understand. The first one is that we must demonstrate the formula by ourselves. Here is the proof demonstrated by myself.



$$\frac{f(R_2)-f(Q_{12})}{x-x_1} = \frac{f(Q_{22})-f(Q_{12})}{x_2-x_1}$$
$$\frac{f(R_1)-f(Q_{11})}{x-x_1} = \frac{f(Q_{21})-f(Q_{11})}{x_2-x_1}$$

Because in image processing, x_2 equals x_1. So we can simplify these equations.

$$f(R_1) = (x - x_1)Q_{21} + (1 - x + x_1)Q_{11}$$

$$f(R_2) = (x - x_1)Q_{22} + (1 - x + x_1)Q_{12}$$

$$\frac{f(P)-f(R_1)}{y-y_1} = \frac{f(R_2)-f(R_1)}{y_2-y_1}$$

Let f(R_1) and f(R_2) be replaced.

$$P(x,y) = [1-x+x_1 \ x-x_1] \begin{bmatrix} Q_{11}Q_{12} \\ Q_{21}Q_{22} \end{bmatrix} [1-y+y_1 \ y-y_1]^{\mathrm{T}}$$

Let x-x_1 as u and y-y_1 as v.

$$P(x,y) = [1-u \ u] \begin{bmatrix} Q_{11}Q_{12} \\ Q_{21}Q_{22} \end{bmatrix} [1-v \ v]^{\mathrm{T}}$$

As we can see from proof, if we don't prove this formula, we can't understand the meaning of u and v. Furthermore, we must program by mapping between new image and original image. In addition, we also should consider the condition of margin by padding.

Bicubic interpolation

Code Block

```
[X,Y]=meshgrid(0:255)
[XQ,YQ]=meshgrid(0:0.5:255)

% padding block
image_recover=interp2(X,Y,image_narrow,XQ,YQ,'cubic')

subplot(2,1,1)
imshow(image)
title('original image')

subplot(2,1,2)
imshow(image_recover)
title('image recover')
```

Result

Describe and analyze

Because the question ask us to code with library, I didn't code without tools. But I search on internet and survey the principle of this method. It is found that this algorithm use 16 points to calculate value and multiple weight of each point [1] . This algorithm utilizes sinc as convolution function. Here is the formula.

$$F(i+v,j+u) = A * B * C$$

$$A = \begin{pmatrix} S(1+v) & S(v) & S(1-v) & S(2-v) \end{pmatrix}$$

$$B = \begin{pmatrix} f(i-1,j-1) & f(i-1,j) & f(i-1,j+1) & f(i-1,j+2) \\ f(i,j-1) & f(i,j) & f(i,j+1) & f(i,j+2) \\ f(i+1,j-1) & f(i+1,j) & f(i+1,j+1) & f(i+1,j+2) \\ f(i+2,j-1) & f(i+2,j) & f(i+2,j+1) & f(i+2,j+2) \end{pmatrix}$$

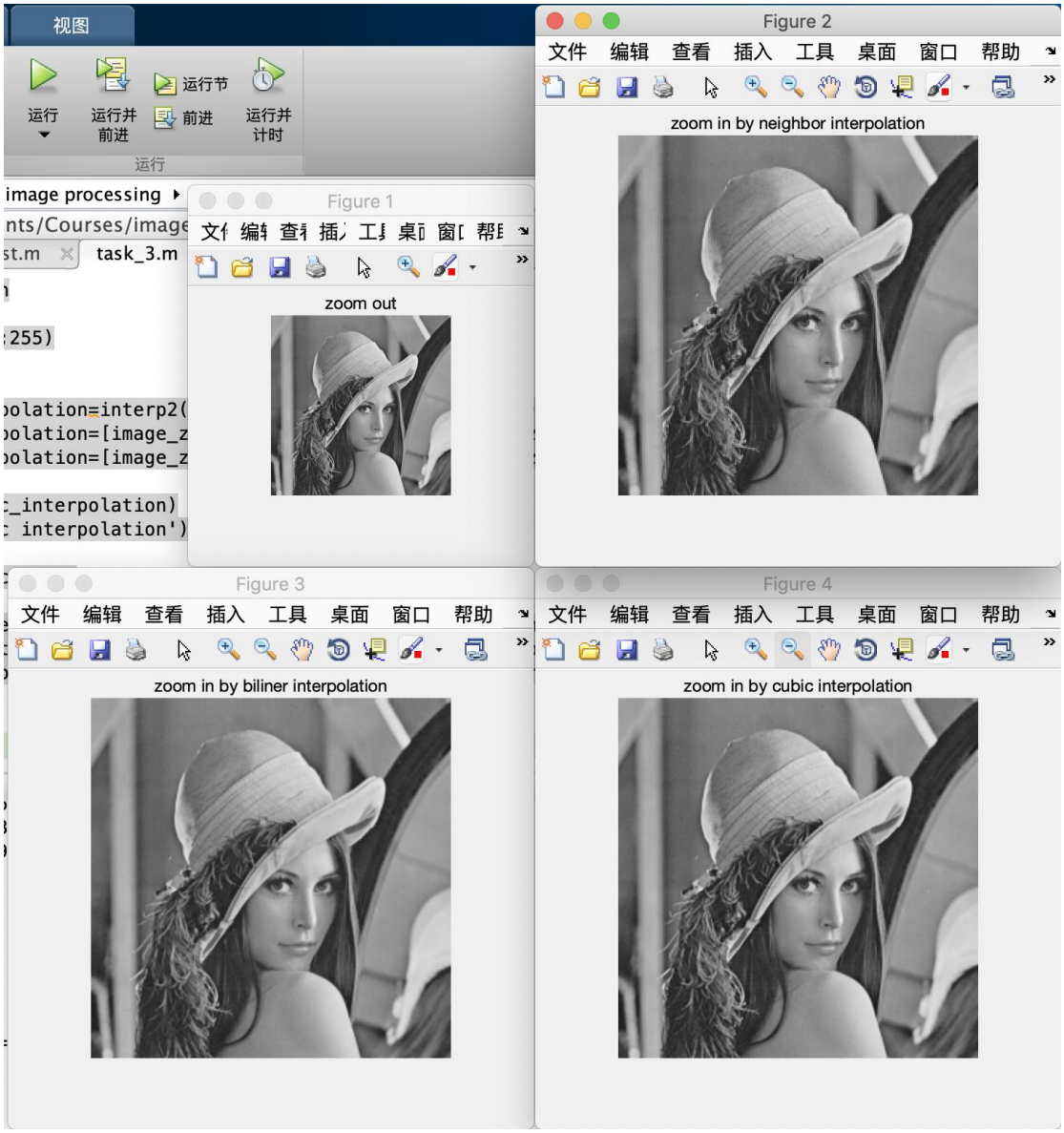
$$C = \begin{pmatrix} S(1+u) \\ S(u) \\ S(1-u) \\ S(2-u) \end{pmatrix}$$

(4) Calculate the psnr between the original image I0 and the up-sampled images, i.e., nearest, bilinear, and bicubic, respectively. Compare the results of different interpolation methods. Explain your founding in the report.

Code Block

```
% mean zoom out
image_gray=imread('lenna512.bmp');
image_narrow=narrow_down(image_gray);
figure(1)
imshow(image_narrow)
title('zoom out')

image_zoom_neighbor_interpolation=neighbor_interpolation(image_narrow);
figure(2)
imshow(image_zoom_neighbor_interpolation)
title('zoom in by neighbor interpolation')

% biliner interpolation
image_zoom_biliner_interpolation=biliner_interpolation_1(im2uint8(image_narrow),2);
figure(3)
imshow(image_zoom_biliner_interpolation)
title('zoom in by biliner interpolation')


% bicubic interpolation
[X,Y]=meshgrid(0:255)
[XQ,YQ]=meshgrid(0:0.5:255)

% padding block
image_zoom_cubic_interpolation=interp2(X,Y,image_narrow,XQ,YQ,'cubic')
image_zoom_cubic_interpolation=[image_zoom_cubic_interpolation,image_zoom_cubic_interpolation(:,end)];
image_zoom_cubic_interpolation=[image_zoom_cubic_interpolation;image_zoom_cubic_interpolation(end,:)];
figure(4)
imshow(image_zoom_cubic_interpolation)
title('zoom in by cubic interpolation')

% Calculate PSNR respectively

PSNR_neighbor=CalculatePSNR(image_gray,im2uint8(image_zoom_neighbor_interpolation))
PSNR_biliner_interpolation=CalculatePSNR(image_gray, im2uint8(image_zoom_biliner_interpolation))
PSNR_cubic_interpolation=CalculatePSNR(image_gray, im2uint8(image_zoom_cubic_interpolation))
```

Result



PSNR_neighbor =

36.8532

PSNR_biliner_interpolation =

36.0962

PSNR_bicubic_interpolation =

36.1001

Describe and analyze

The consequence presents that there are some differences between theoretical value and practical value. The ranking of theory is bicubic, bilinear and neighbor, but actually neighbor interpolation works best in this picture. I think we should choose different algorithm in different situations.

Task 4

The original image of Lenna (i.e., lenna512.bmp) uses 8 bits to represent the intensity levels, so it has 256 gray levels. Write a script to reduce it to 16 values by quantization. Display the quantized image, and describe the effect of severe quantization on images.

Code Block

```
image=imread('lenna512.bmp')
% if we don't multiply 16 again, the image will be dark
image_8=(image./16+1)*16-1;
figure(1)
imshow(image)
title('Original image with 256 depth')

figure(2)
imshow(image_8)
title('Image with 8 depth')
```

Result

Describe and analyze

We can find that the gray image will create gaps and faults when 256 gray levels are reduced to 16 levels. So the more gray level, the details the picture will be.

Reference List

[1]. S. E. Reichenbach and F. Geng, "Two-dimensional cubic convolution," in IEEE Transactions on Image Processing, vol. 12, no. 8, pp. 857-865, Aug. 2003. Doi: 10.1109/TIP.2003.814248.
Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1217263&isnumber=27367