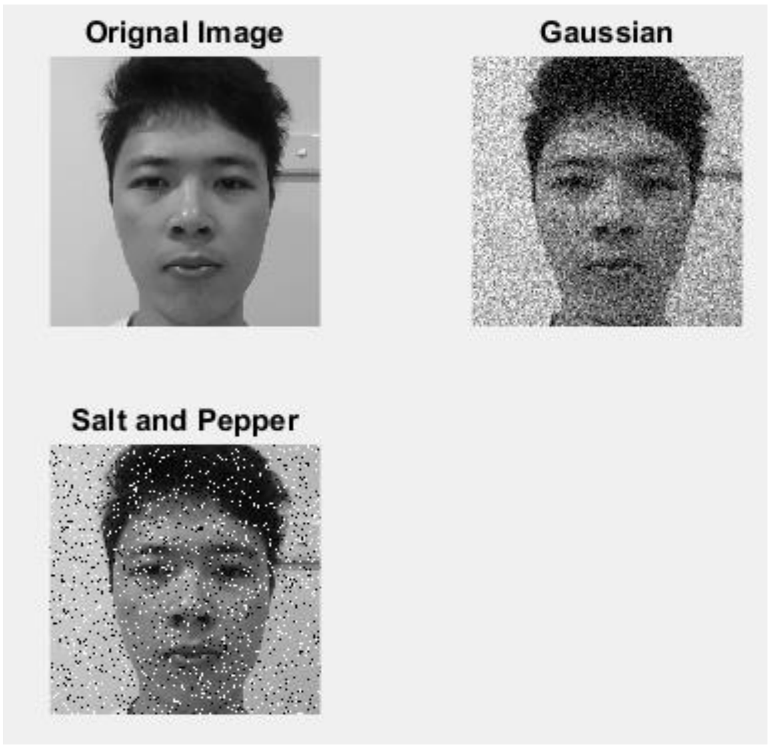
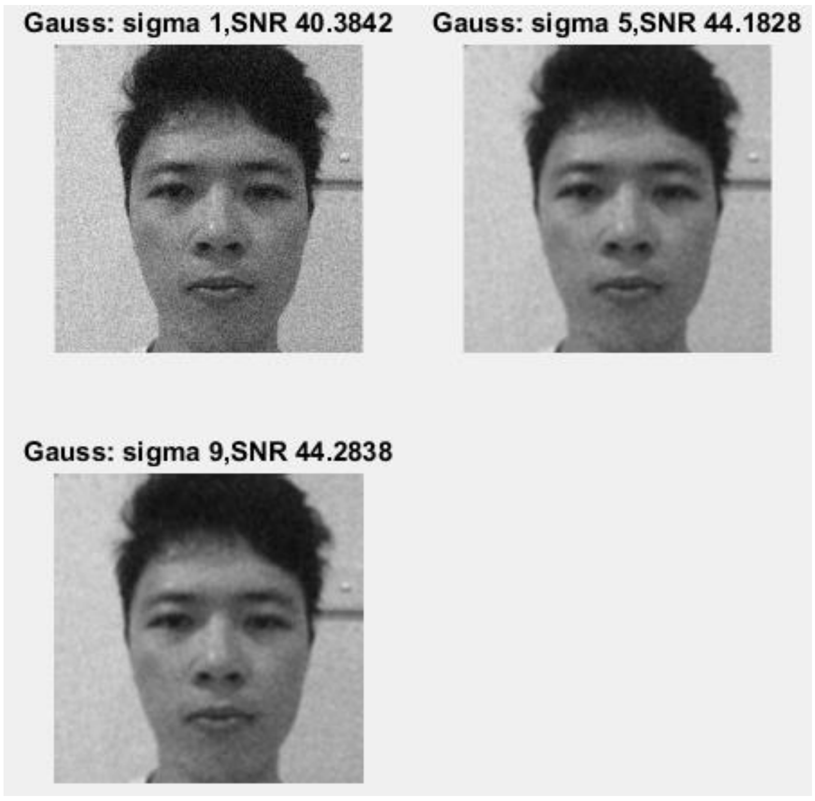
C-Lab-1 Report

Task-1: Image denoise and filtering:

The face image with class of uint8 is converted to greyscale then resized to 512 x512 resolution. Original image, image with Gaussian noise (zero means & 10 standard variation) and salt and pepper noise (density 0.1) are shown. Noise is added by using Matlab built-in function imnoise. Notice since the image is in uint8 class and imnoise function converts uint8 values to doubles before applying the variance, so the standard deviation of Gaussian need to be scaled down by 255.

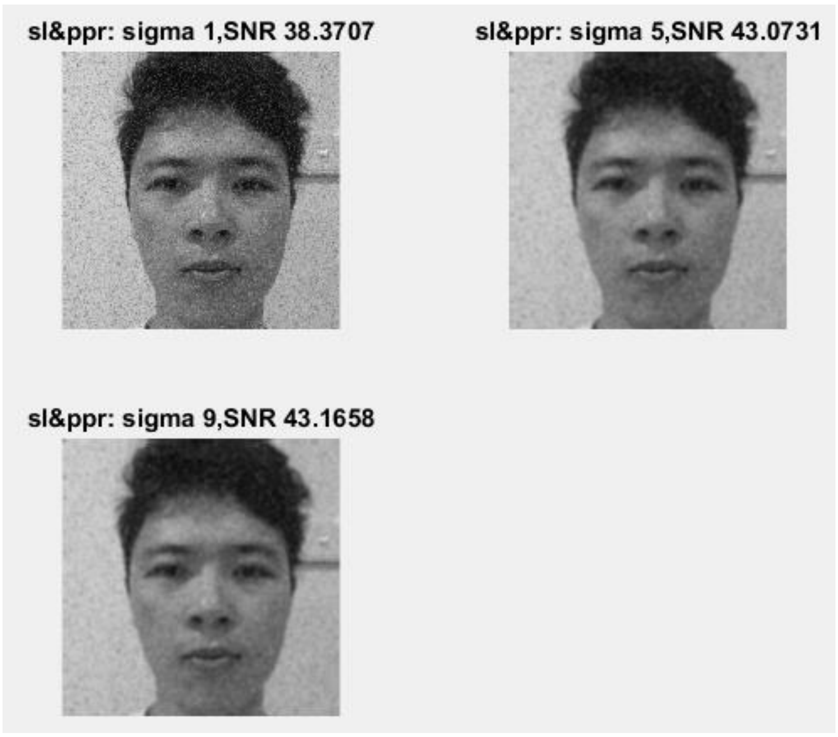



Gauss: sigma 1,SNR 40.3842
Gauss: sigma 5,SNR 44.1828
Gauss: sigma 9,SNR 44.2838

After that, 9 x 9 Gaussian filter is implemented. To create consistent image after convolution, the borders of original image is mirror-reflected to extend the image. Then a 9 x 9 kernel of Gaussian filter is created using fspecial function. Finally a function called mg_Gauss_filter is created to perform the convolution. The resulting image under sigma 1, 5, and 9 are compared in figure on the left hand side. The signal-to-noise-ratio (SNR) increased from 40.3 to 44.1 when sigma change from 1 to 5, after that increasing sigma has little effect on the SNR.

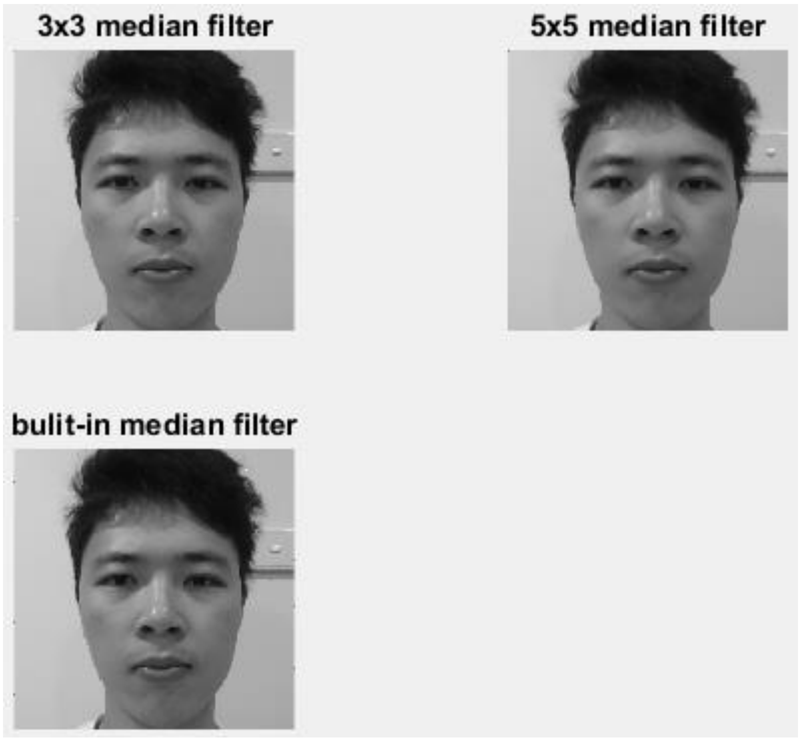
C-Lab-1 Report

The diagram below is the filtered images with Gaussian filter against salt and pepper noise.


sl&ppr: sigma 1,SNR 38.3707
sl&ppr: sigma 5,SNR 43.0731
sl&ppr: sigma 9,SNR 43.1658

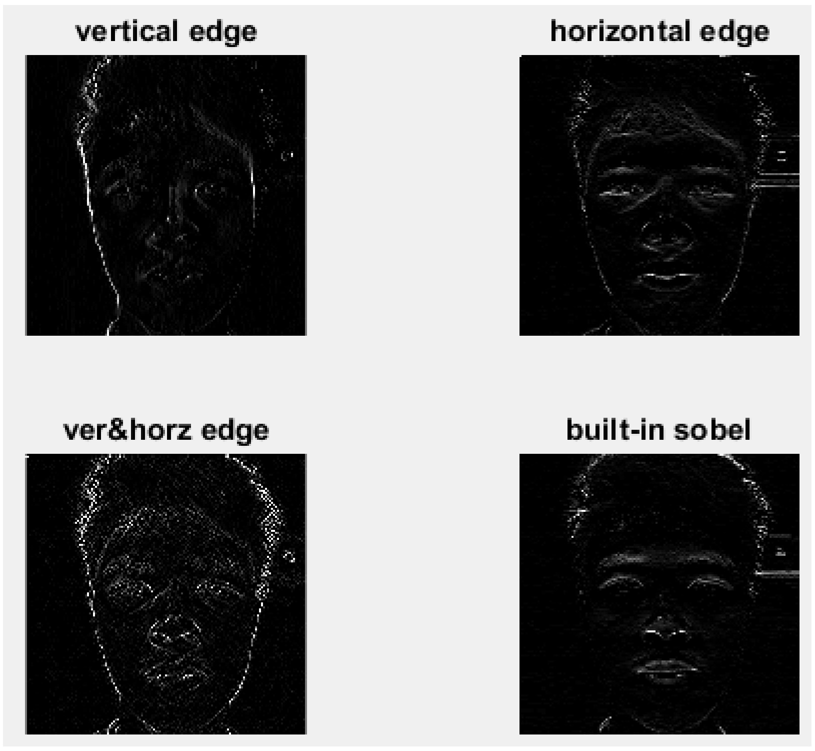
By comparing the signal-to-noise ratio the Gaussian filter is less effective in removing salt and pepper noise. Since salt and pepper noise consists of zeros and maximum valued pixel randomly on image. Gaussian filter can smooth the noise and spread it to surrounding pixels to so extent. But resident noise is still obvious in resulting image. Since Gaussian noise has far less sudden change, thus Gaussian filter is effective on removing Gaussian noise compared to salt and pepper noise.

The following diagram shows the implementation of median filter. First of all a 3 x 3 median filter is implemented. Median filter is implemented by replacing the pixel with the median value of the sorted array constructed by surrounding 3 x 3 pixels. In the resulting image under 3 x 3 median filter, there are few spikes that haven't been removed. This is due the extreme condition that the spikes in the noisy image is to many so that the spikes ended up of being the median value. This can be eliminate by implementing a median filter with larger kernel size. In this case 5 x 5 median filter is implemented and the resulting image has a better result. Compared the median function with Matlab built-in median filter function, the effect is mostly the same except under Matlab built-in filter has some black spikes leak in the image. This is due to the padding array around the image. In self programed code before applying the filter, image border is mirror-reflected, so there



is no black pixel pollution. In the case of removing salt and pepper noise, the median filter is better. Because median filter actually replace the spikes with the median value of surrounding pixels, which can eliminate the black and white extreme value , but Gaussian filter only doing the smoothing, which spreads spikes in surrounding pixels. Hence median filter is better in removing salt and pepper noise.

The result under Sobel filter is shown on the right hand side. The self-programed Sobel filter is h = [-1 0 1; -2 0 2; -1 0 1;]; and h = [1 2 1; 0 0 0; -1 -2 -1;]; the first one extract the vertical edges, and the second one extract the horizontal edges. By combining them gives the whole edge image. The built-in Sobel operation is default as h = [1 2 1; 0 0 0; -1 -2 -1;], which extracts the horizontal edges.



Code of task 1:

```
clc,clear;
%%%%%%%%%%%%%%%%  Task 1 %%%%%%%%%%%%%%%%%%%
%------------------------------------------%
%step 1:read img,resize to 512 x 512,convert to greyscale,[0, 255]
%%{
img = imread('face.jpg');            %read image from folder
angleDeg = 90;                       %set a rotation angle of 90 degrees
img=imrotate(img,angleDeg);          %rotate the image by 90 degrees
img = imresize(img, 0.4);            %resize the image to 0.4 of the original
img = img(200:711, 70:581,:);        %slice the image to meet resolution
img = rgb2gray(img);                 %convert image to grayscale
imwrite(img,'face_w.jpg');
%}
img = imread('face_w.jpg');
%------------------------------------------%
%step 2: add gaussian and salt&pepper noise use imnoise
imgN1 = imnoise(img,'gaussian',0, 5/255);
                                     %gaussian noise mean 0 & var 5
imgN2 = imnoise(img,'salt & pepper',0.1);
                                     %salt&pepper density 0.1
%figure;
subplot(2,2,1);imshow(img);title('Orignal Image');
                                     %display original image
subplot(2,2,2);imshow(imgN1);title('Gaussian');
                                     %display gaussian noised image
subplot(2,2,3);imshow(imgN2);title('Salt and Pepper');
                                     %display salt&pepper noised image
%------------------------------------------%
%step 3:implement 9X9 gaussian filter
%%{
figure;%for gaussian noisy image
for i = 1:3
```

```
    hsize = 9;                          %define gaussian filter size
    sigma = 1+4*(i-1);                          %define gaussian filter variance
    h = fspecial('gaussian',hsize,sigma);%create a gaussian filter kernel
    result = my_Gauss_filter(imgN1,h);  %call the filtering function
    img = im2double(img);               %convert the original image to double
    SNR = 20*log ( norm(img,'fro') /norm(img - result, 'fro' ));
                                    %calculate the signal to noise ratio
    subplot(2,2,i);imshow(result);title(['Gauss: sigma ', num2str(sigma) ',SNR
',num2str(SNR)]);
end
figure;%for salt&pepper noisy image
for i = 1:3
    hsize = 9;                          %define gaussian filter size
    sigma = 1+4*(i-1);                          %define gaussian filter variance
    h = fspecial('gaussian',hsize,sigma);%create a gaussian filter kernel
    result = my_Gauss_filter(imgN2,h);  %call the filtering function
    img = im2double(img);               %convert the original image to double
    SNR = 20*log ( norm(img,'fro') /norm(img - result, 'fro' ));
                                    %calculate the signal to noise ratio
    subplot(2,2,i);imshow(result);title(['sl&ppr: sigma ', num2str(sigma) ',SNR
',num2str(SNR)]);
end
%}
%-------------------------------------------%
%%{
%step 4:implement 3x3 median filter
figure;
h = zeros(3,3);
result = my_Median_filter(imgN2,h);
subplot(2,2,1);imshow(result);title('3x3 median filter'); %display result image
h = zeros(5,5);
result = my_Median_filter(imgN2,h);
subplot(2,2,2);imshow(result);title('5x5 median filter'); %display result image
result = medfilt2(imgN2);
subplot(2,2,3);imshow(result);title('bulit-in median filter'); %display result
image
%}
%-------------------------------------------%
%step 5:implement 3x3 sobel edge detector
%%{
figure;
h = [-1 0 1; -2 0 2; -1 0 1;];
result = my_Sobel_filter(img,h);
subplot(2,2,1);imshow(result);title('vertical edge'); %display result imag
h = [1 2 1; 0 0 0; -1 -2 -1;];
result1 = my_Sobel_filter(img,h);
subplot(2,2,2);imshow(result1);title('horizontal edge'); %display result imag
result = my_Sobel_filter(result,h);
subplot(2,2,3);imshow(result);title('ver&horz edge'); %display result imag
h = fspecial('sobel');
result = imfilter(img,h);
subplot(2,2,4);imshow(result);title('built-in sobel'); %display result imag
%}
%-------------------------------------------%
```

```
function output_image = my_Gauss_filter(img, h)
lenHrow =size(h,1);                                     %get the row of filter
lenHcol =size(h,2);                                     %get the col of filter
lenIrow =size(img,1);                                   %get the row of image
lenIcol =size(img,2);                                   %get the col of image
%creatre new image with to avoid black edges around the image after convolution
temp = flipud(img(1:lenHrow,:));
imgNew = cat(1,temp,img);
temp = flipud(img(lenIrow-lenHrow+1:lenIrow,:));
imgNew = cat(1,imgNew,temp);
temp = zeros(lenIrow+2*lenHrow,lenHcol);
temp(:,:) = mean(mean(img));
imgNew = cat(2,temp,imgNew);
imgNew = cat(2,imgNew,temp);
temp = fliplr(img(:,1:lenHcol));
imgNew(lenHrow+1:lenHrow+lenIrow,1:lenHcol)=temp;
temp = fliplr(img(:,lenIcol-lenHcol+1:lenIcol));
imgNew(lenHrow+1:lenIrow+lenHrow,lenIcol+lenHcol+1:lenIcol+2*lenHcol)=temp;
%finished creating the new image
lenINrow = size(imgNew,1);                      %get the row of new image
lenINcol = size(imgNew,2);                      %get the col of new image
h = fliplr(h);                                  %flip kernel left right
h = flipud(h);                                  %flip kernel up down
pix = 0;
imgNew = im2double(imgNew);                     %convert image to double
result = zeros(lenINrow,lenINcol);              %create matrix to hold result
hSum = sum(sum(h));                             %for nomalization
for k = 1:lenINrow-lenHrow+1                    %pixel operation for image
    for q = 1:lenINcol-lenHcol+1                %pixel opeartion for image
        temp = imgNew(k:k+lenHrow-1,q:q+lenHcol-1);%create a patch same size as
the image
        temp = temp.*h;                         %element-by-element
multiplication with the kernel
        pix = sum(sum(temp))/hSum;                   %sum all the pix of the
patch
        result(k+round(lenHrow)-1,q+round(lenHcol-1)) = pix;%allocate the pix in
the result image
    end
end
output_image = result(lenHrow:lenINrow-lenHrow-1,lenHcol:lenINcol-lenHcol-
1);%output of the function
```

```
function output_image = my_Median_filter(img, h)
lenHrow =size(h,1);                                 %get the row of filter
lenHcol =size(h,2);                                 %get the col of filter
lenIrow =size(img,1);                               %get the row of image
lenIcol =size(img,2);                               %get the col of image
%creatre new image with to avoid black edges around the image after convolution
temp = flipud(img(1:lenHrow,:));
imgNew = cat(1,temp,img);
temp = flipud(img(lenIrow-lenHrow+1:lenIrow,:));
imgNew = cat(1,imgNew,temp);
temp = zeros(lenIrow+2*lenHrow,lenHcol);
temp(:,:) = mean(mean(img));
imgNew = cat(2,temp,imgNew);
imgNew = cat(2,imgNew,temp);
temp = fliplr(img(:,1:lenHcol));
imgNew(lenHrow+1:lenHrow+lenIrow,1:lenHcol)=temp;
temp = fliplr(img(:,lenIcol-lenHcol+1:lenIcol));
imgNew(lenHrow+1:lenIrow+lenHrow,lenIcol+lenHcol+1:lenIcol+2*lenHcol)=temp;
%finished creating the new image
lenINrow = size(imgNew,1);                      %get the row of new image
lenINcol = size(imgNew,2);                      %get the col of new image
h = fliplr(h);                                  %flip kernel left right
h = flipud(h);                                  %flip kernel up down
pix = 0;
imgNew = im2double(imgNew);                     %convert image to double
result = zeros(lenINrow,lenINcol);              %create matrix to hold result
for k = 1:lenINrow-lenHrow+1                    %pixel operation for image
    for q = 1:lenINcol-lenHcol+1                %pixel opeartion for image
        temp = imgNew(k:k+lenHrow-1,q:q+lenHcol-1);%create a patch same size as
the image
        if lenHrow == 5                         %to compare with 3x3 median
filter create a 5x5 one
            temp =
cat(2,temp(1,1:lenHcol),temp(2,1:lenHcol),temp(3,1:lenHcol),temp(4,1:lenHcol),tem
p(5,1:lenHcol));
        end
        if lenHrow == 3                         %3X3 median filter
            temp = cat(2,temp(1,1:lenHcol),temp(2,1:lenHcol),temp(3,1:lenHcol));
        end
        temp = sort(temp);                      %sort the array
        pix = median(temp);                     %find the median value of the
array
        result(k+round(lenHrow)-1,q+round(lenHcol-1)) = pix;%allocate the pix in
the result image
    end
end
output_image = result(lenHrow:lenINrow-lenHrow-1,lenHcol:lenINcol-lenHcol-
1);%output of the function
```

```
function output_image = my_Sobel_filter(img, h)
lenHrow =size(h,1);                                 %get the row of filter
lenHcol =size(h,2);                                 %get the col of filter
lenIrow =size(img,1);                               %get the row of image
lenIcol =size(img,2);                               %get the col of image
%creatre new image with to avoid black edges around the image after convolution
temp = flipud(img(1:lenHrow,:));
imgNew = cat(1,temp,img);
temp = flipud(img(lenIrow-lenHrow+1:lenIrow,:));
imgNew = cat(1,imgNew,temp);
temp = zeros(lenIrow+2*lenHrow,lenHcol);
temp(:,:) = mean(mean(img));
imgNew = cat(2,temp,imgNew);
imgNew = cat(2,imgNew,temp);
temp = fliplr(img(:,1:lenHcol));
imgNew(lenHrow+1:lenHrow+lenIrow,1:lenHcol)=temp;
temp = fliplr(img(:,lenIcol-lenHcol+1:lenIcol));
imgNew(lenHrow+1:lenIrow+lenHrow,lenIcol+lenHcol+1:lenIcol+2*lenHcol)=temp;
%finished creating the new image
lenINrow = size(imgNew,1);                      %get the row of new image
lenINcol = size(imgNew,2);                      %get the col of new image
h = fliplr(h);                                  %flip kernel left right
h = flipud(h);                                  %flip kernel up down
pix = 0;
imgNew = im2double(imgNew);                     %convert image to double
result = zeros(lenINrow,lenINcol);              %create matrix to hold result
for k = 1:lenINrow-lenHrow+1                    %pixel operation for image
    for q = 1:lenINcol-lenHcol+1                %pixel opeartion for image
        temp = imgNew(k:k+lenHrow-1,q:q+lenHcol-1);%create a patch same size as
the image
        temp = temp.*h;                         %element-by-element
multiplication with the kernel
        pix = sum(sum(temp));                   %sum all the pix of the patch
        result(k+round(lenHrow)-1,q+round(lenHcol-1)) = pix;%allocate the pix in
the result image
    end
end
output_image = result(lenHrow:lenINrow-lenHrow-1,lenHcol:lenINcol-lenHcol-
1);%output of the function
```

Task-2: Image denoise and filtering:

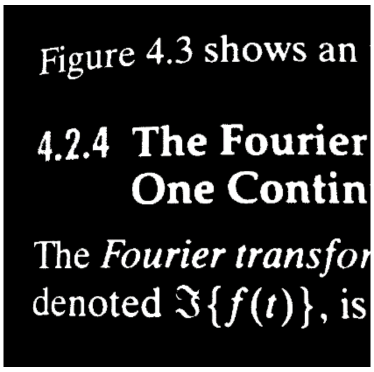


Figure 4.3 shows an

4.2.4 The Fourier
One Contin

The *Fourier transfor*
denoted $\Im\{f(t)\}$, is

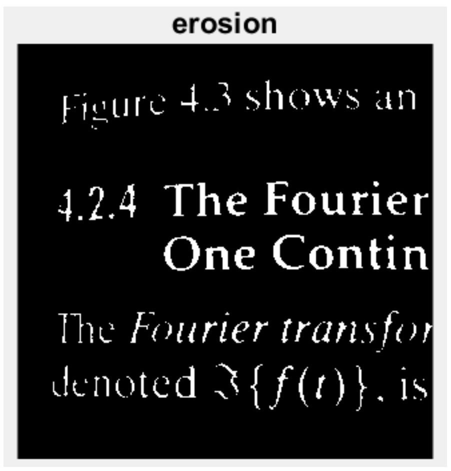
The image is resized, converted to greyscale, threshold to binary image and shown on the right hand side. Using im2bw built-in function with 0.5 parameter and imcomplement function, the image has 939364 black and 109212 white pixels, summing to 1048576, which equals to 1024*1024. The image is one mega-pixel image.

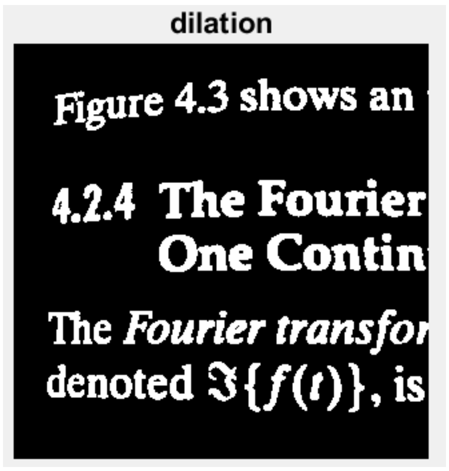
Image Morphology: structure element square with size 5 by 5 pixels and value 1.

C-Lab-1 Report

Erosion: performs AND operation. If pixel block AND with the structure element is true then gives one. Since text contents have value of one, this operation shrinks the text and enlarge the black portion. Visually the text lines appear thinner, any possible noise in the black ground less than the size of the structure element is eliminated by the AND operation.

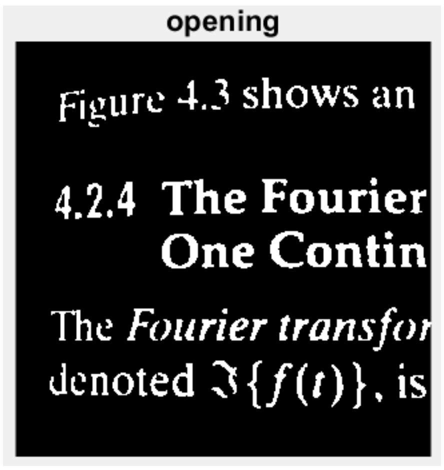
Dilation: performs OR operation. If pixel block OR with the structure element is true then gives one. Since text contents have value of one, the operation enlarge the text and shrinks the black portion. Visually the text lines appear bolder and discontinuity between each word can be removed.

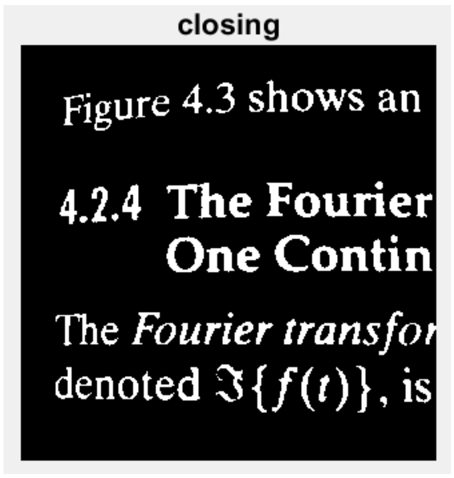




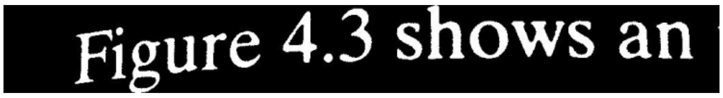
Opening: erosion next to dilation using the same structuring element. First shrinks the text with the shape of square structure element, this makes the text squarish, and may disconnect some words if the white parts is smaller than the structure element. Also eliminate any possible white noise in the background. Then enlarge white to shrink the text. Some of the discontinuity remains.

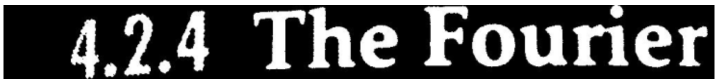
Closing: dilation next erosion using the same structuring element. First enlarge white, the text become bolder, any possible noise in the background will be more obvious. But this connect the disconnected word. The following erosion will make text thinner, but will not destroy the connectivity in each word.

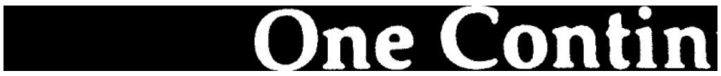


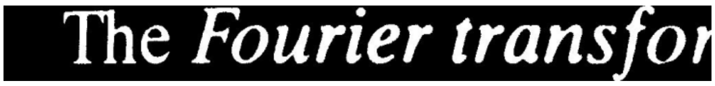


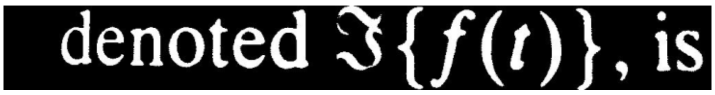
To separate the text, a rectangular structure (30 rows 100 columns) element is utilized to connect all the white parts in one line by image closing operation. Then bwlabel function is used to label the connected lines. Each label serves as the text location information to slice iamge.

Line1 
Figure 4.3 shows an

Line2 
4.2.4 The Fourier

Line3 
One Contin

Line4 
The *Fourier transfor*

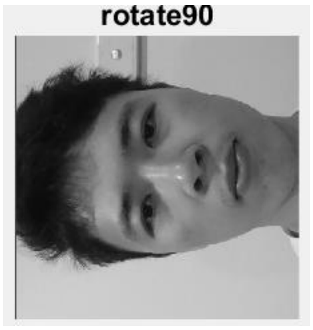
Line5 
denoted $\Im\{f(t)\}$, is

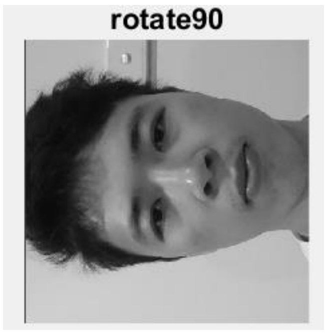
```
clc,clear;
%%%%%%%%%%%%%%%%  Task 2 %%%%%%%%%%%%%%%%%%%
%------------------------------------------%
%step 1:read img,resize to 1024 x 1024,convert to greyscale,[0, 255]
%%{
img = imread('text.png');                   %read image from folder
img = imresize(img, 0.604);                 %resize the image to 0.4 of the
original
img = img(1:1024,:,:);                      %slice the image to meet resolution
img = rgb2gray(img);                        %convert image to grayscale
%step 2: threshold the histogram to binary image
img = im2bw(img, 0.5);                      %bulit-in function to convert to
binary
imwrite(img,'text_w.jpg');
img = imcomplement(img);                    %complement the iamge
figure;
imshow(img);                                %show image
[counts,binLocations] = imhist(img);        %use histogram to counts
blackPix = counts(1,1);                     %number of black pix
whitePix = counts(2,1);                     %number of white pix
totalPix = blackPix + whitePix;             %total pix of the binary image
SE = strel('square',5)                      %define a structure element
imgErode = imerode(img,SE,'same');                 %perform image erosion
imgDilate = imdilate(img,SE);               %perform image dilation
imgOpen = imopen(img,SE);                   %perform image opening
imgClose = imclose(img,SE);                 %perform image closing
figure;                                     %display the above four image
subplot(2,2,1);imshow(imgErode);title('erosion');
subplot(2,2,2);imshow(imgDilate);title('dilation');
subplot(2,2,3);imshow(imgOpen);title('opening');
subplot(2,2,4);imshow(imgClose);title('closing');
lenIrow = size(img,1);                      %get the row of image
lenIcol = size(img,2);                      %get the col of image
temp = ones(lenIrow,1);                     %create a temp array to hold
variables
rowCollector = [];                          %define empty array to collect rows

SE = strel('rectangle',[30 100]);
imgClose = imclose(img,SE);
figure,imshow(imgClose);
[L,num] = bwlabel(imgClose);
i = 1;
for i = 1:num
[r, c] = find(L == i);
rMax = max(r);
rMin = min(r);
temp = img(rMin:rMax,:);
```

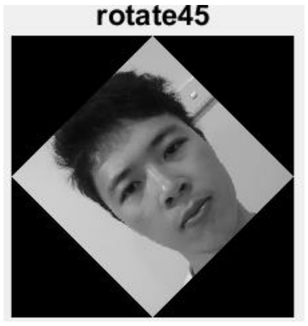
```
str = num2str(num+1-i);
imwrite(temp,strcat('line',str,'.jpg'));
end
```

Task-3 Geometric transformation:

The diagram shows the self-programed rotation degree of 90, 45, -45 and -90 degrees using forward warping method.

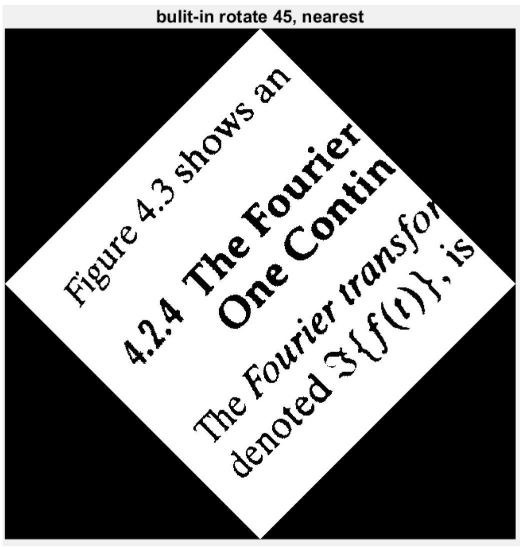


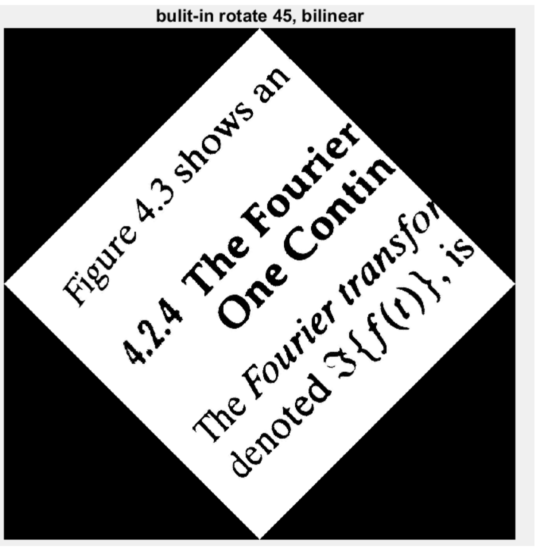




First a matrix that can hold the maximum size of image after rotation is created. Then a rotation matrix and two translation parameter are created. Inverse warping method is used to locate every pixel in the new image back to the original image so that each pixel in new image has corresponding value. For the case of some pixel may not be mapped to integer value in the original image. Bilinear interpolation is used. First get 4 nearest pixel location around the non-integer pixel. Then the intensity of the 4 pixels used to perform bilinear interpolation.

The following diagram shows the built-in rotation result with inverse warping method.

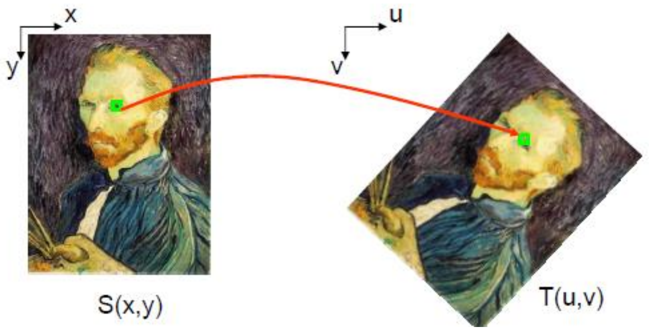




Forward warping:

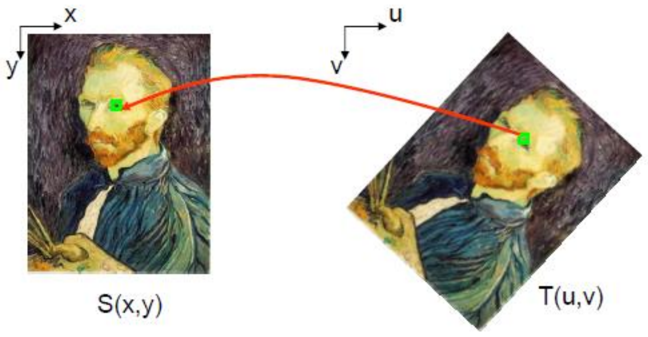
C-Lab-1 Report

Forward warping iteratively transfer pixels from original image to the transformed image. Generally pixels after transform would not fall in the exact pixels in the transformed image, this leads to the distribution colour among neighbouring pixels.



Inverse Warping:

Inverse warping is a more universal method to remove holes. This method maps pixels in transformed image to the original image, so that each pixel has their corresponding mapping. By using different interpolation method such bilinear and nearest neighbour method.

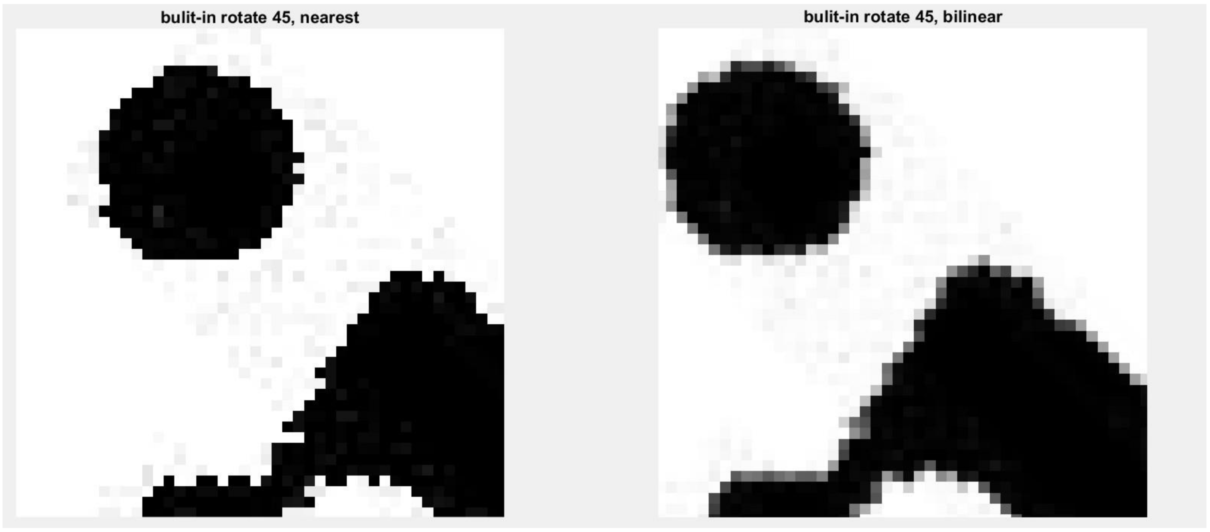


Nearest Neighbour Interpolation:

For nearest neighbour interpolation, the block uses the value of nearby translated pixel values for the output pixel values.

Bilinear interpolation:

For bilinear interpolation, the block uses the weighted average of two translated pixel values for each output pixel value. The following diagram shows the zoom-in figure after rotation.



By doing nearest interpolation the interpolated pixel value is same as the nearest neighbour. As the nearest neighbour is either black or white, so the interpolated pixel is either black or white. By doing bilinear interpolation the interpolated value is calculated by using the linear interpolation. So the interpolated pixel is the average of black and white which gives blur grey pixel value after being enlarged. Same effect happen in the image border with sawtooth-shape.

```
clc,clear;
img = imread('face_w.jpg');%rotate square gray image
img45 = my_Rotation(img,pi/4);
img90 = my_Rotation(img,pi/2);
img45M = imrotate(img,45);
subplot(2,2,1);imshow(img);title('original');
subplot(2,2,2);imshow(img45);title('rotate45');
subplot(2,2,3);imshow(img90);title('rotate90');
subplot(2,2,4);imshow(img45M);title('builtInRotate45');
```

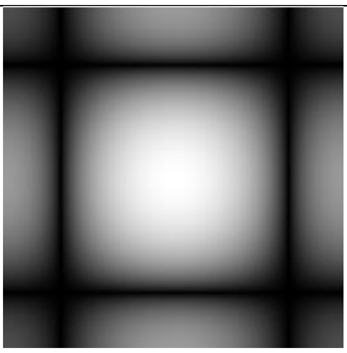
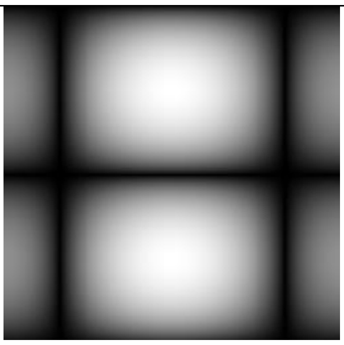
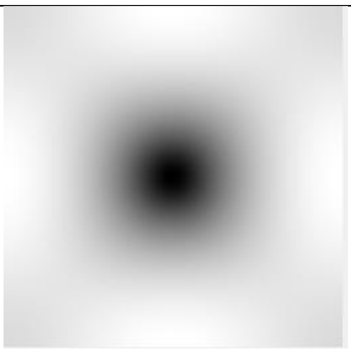
```
function result = my_Rotation( img,angle )           %function definition
img=double(img);
lenIrow=size(img,1);                                 %get length of row
lenIcol=size(img,2);                                 %get length of col
lenNrow=lenIcol*sin(angle)+lenIrow*cos(angle);       %get row of rotated img
lenNcol=lenIcol*cos(angle)+lenIrow*sin(angle);       %get col of rotated img
lenNrow=ceil(lenNrow);                               %round to integer
lenNcol=ceil(lenNcol);                               %round to integer
T=lenIcol*sin(angle);                                %translation parameter
R=[cos(angle),sin(angle);-sin(angle),cos(angle)];    %rotation matrix
%inverse mapping method
for u=1:lenNrow                                      %for each row in new img
    for v=1:lenNcol                                  %for each col in new img
        temp=R*([u-T;v]);                            %translation and rotation
        x=temp(1);                                   %get x from new img
        y=temp(2);                                   %get y from new img
        if x>=1 & x<=lenIrow & y>=1 & y<=lenIcol     %if xy falls on the range
            x1 = floor(x);                           %floor the x
            y1 = floor(y);                           %floor the y
            y2 = ceil(y);                            %ceil the y
            x2 = ceil(x);                            %ceil the x
            if (x-x1) <= (x2-x)                      %get 4 surrounding pix
                  x = x1;                            %then get the nearest pix
location
            else
                x = x2;
            end
            if (y-y1) <= (y2-y)
                y = y1;
            else
                y = y2;
            end
            %bilinear interpolation
            p1 = img(x1,y1);                         %assign pix value
            p2 = img(x2,y1);                         %assign pix value
            p3 = img(x1,y1);                         %assign pix value
            p4 = img(x2,y2);                         %assign pix value
            s = x-x1;
            t = y-y1;
            imgN(u,v) = (1-s)*(1-t)*p1+(1-s)*t*p3+(1-
t)*s*p2+s*t*p4;%imterpolation
        end
    end
   result= uint8(imgN);
end
```

Task-4 2D FFT and IFFT of image:

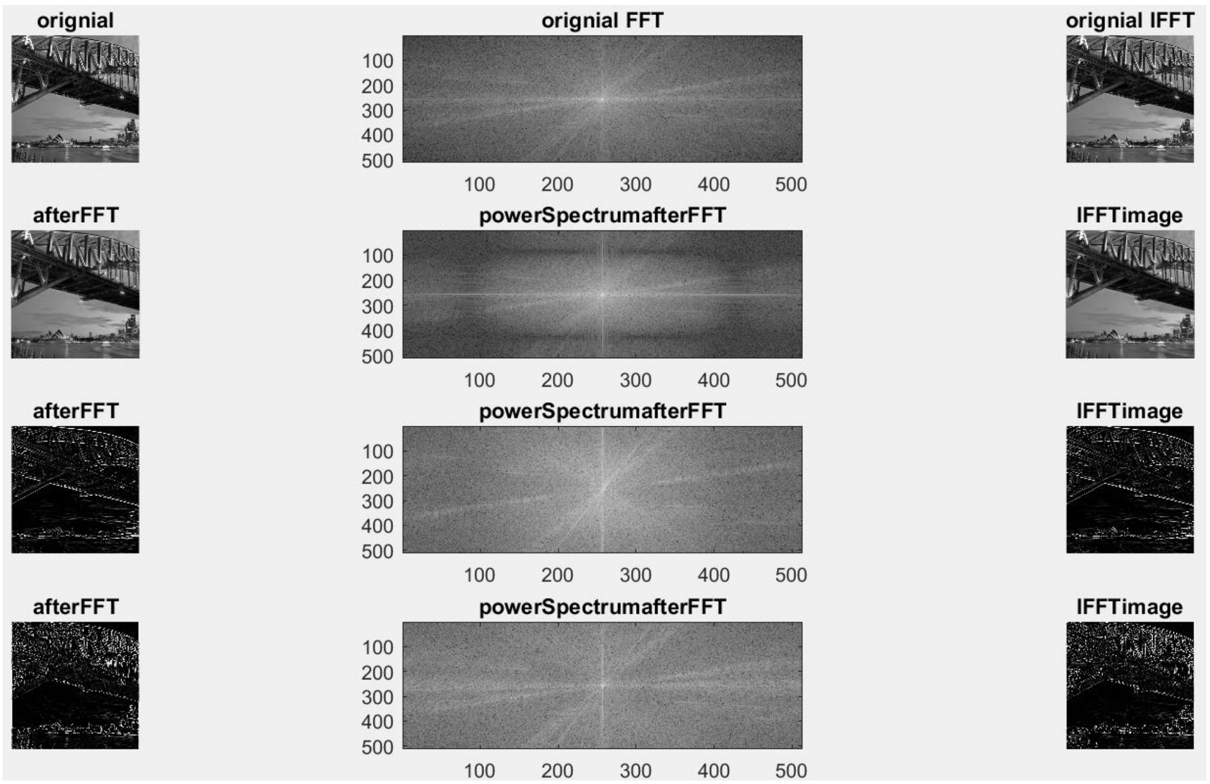
Effect of filter [1, 1, 1; 1, 1, 1; 1, 1, 1]: low pass filter. The power spectrum after filtering is darker in the outer ring compared to the original image power spectrum.

Effect of filter [1, 1, 1; 0, 0, 0; -1, -1, -1]: band pass filter. The power spectrum after filtering is brighter in the middle ring compared to the original image power spectrum. The filter is gradient filter with one in left and -1 in the right. This can extract the change in the image intensity.

Effect of filter [-1, -1, -1; -1, 8, -1; -1, -1, -1]: high pas filter. This artefact is due to the result of a bright image minus by a blur version of image which gives a sharpened image. High frequency content such as change in image is preserved but low frequency content with smooth surface is filtered.

Power spectrum of [1, 1, 1; 1, 1, 1; 1, 1, 1] Low pass, high freq is dark	Power spectrum of [1, 1, 1; 0, 0, 0; -1, -1, -1] Band pass	Power spectrum of [-1, -1, -1; -1, 8, -1; -1, -1, -1] High pass, middle is dark
		

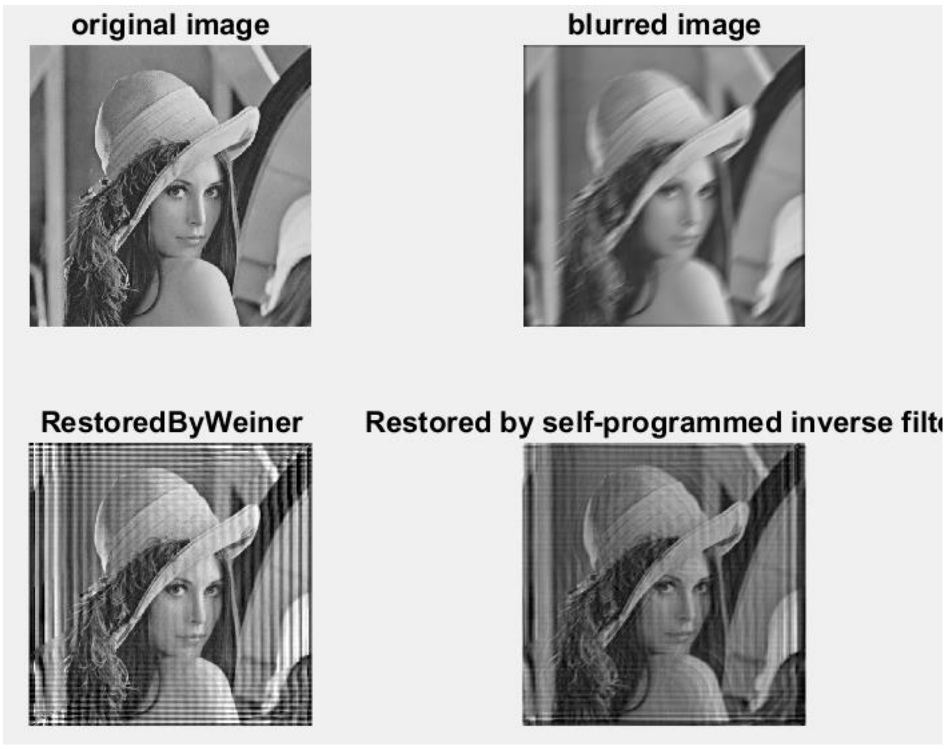
The following diagram is obtain by multiplication in frequency domain.



```
clc,clear;
%%%%%%%%%%%%%%%%  Task 4 %%%%%%%%%%%%%%%%%%%
```

```
%--------------------------------------------%
img = imread('sydney.jpg');                         %read iamge
img = imresize(img,0.6);                            %resize image
img = rgb2gray(img);                                %convert to gray
img = img(1:512,70:582);                            %slice image
figure;                                             %envolve a figure
F=fft2(double(img));                                %convert to double and do fft
A=fftshift(F);                                      %move zero-freq to the center
for visulization
A=log2(A);                                          %computes the base 2
logarithm
A=abs(A);                                           %returns the absolute value
IF =ifft2(F);                                       %do inverse fft
IF =uint8(IF);                                      %convert to unit8 for
visulization
subplot(4,3,1);imshow(img);title('orignial');       %display
subplot(4,3,2);imagesc(A);title('orignial FFT');
subplot(4,3,3);imshow(IF);title('orignial IFFT');
%--------------------------------------------%
h = [1 ,1,1; 1,1,1; 1,1,1];                         %define filter
imgF = imfilter(img,h/9,'conv');                      %convolve filter and image
F=fft2(double(imgF));
A=fftshift(F);
A=log2(A);
A=abs(A);
IF = ifft2(F);
IF =uint8(IF);
subplot(4,3,4);imshow(imgF);title('afterFFT');
subplot(4,3,5);imagesc(A);title('powerSpectrumafterFFT');
subplot(4,3,6);imshow(IF);title('IFFTimage');
%--------------------------------------------%
h = [ 1,1,1; 0,0,0 ; -1,-1,-1];
imgF = imfilter(img,h,'conv');
F=fft2(double(imgF));
A=fftshift(F);
A=log2(A);
A=abs(A);
IF = ifft2(F);
IF =uint8(IF);
subplot(4,3,7);imshow(imgF);title('afterFFT');
subplot(4,3,8);imagesc(A);title('powerSpectrumafterFFT');
subplot(4,3,9);imshow(IF);title('IFFTimage');
%--------------------------------------------%
h = [ -1,-1, -1; -1 , 8, -1; -1,-1,-1];
imgF = imfilter(img,h,'conv');
F=fft2(double(imgF));
A=fftshift(F);
A=log2(A);
A=abs(A);
IF = ifft2(F);
IF =uint8(IF);
subplot(4,3,10);imshow(imgF);title('afterFFT');
subplot(4,3,11);imagesc(A);title('powerSpectrumafterFFT');
subplot(4,3,12);imshow(IF);title('IFFTimage');
%subplot(4,3,3);imagesc(img-IF);title('img-IF');
%--------------------------------------------%
```

Task-5 Image deblur:



The above diagram shows the effect on motion blurred image under built-in Wiener filter and self-programmed inverse filter. The SNR in Weiner is 0.001, this gives ringing effect around the image. The self-programmed inverse filter takes the FFT for both inverse filter and image and perform multiplication in frequency domain. The inverse filter is generate by thresholding at 0.1. Any value below that thresholding is replaced by 1.

```
clc,clear;
%%%%%%%%%%%%%%%%  Task 5 %%%%%%%%%%%%%%%%%%%
%------------------------------------------%
%image blurring and deblurring using bulit-in weiner function
img = imread('Lenna.png');                                %read image
img = rgb2gray(img);                                      %convert image to
gray
figure;subplot(2,2,1);imshow(img);title('original image');  %show image
theta = 30; len = 20;                                     %define motion blur
parameters
h = imrotate(ones(1, len), theta, 'bilinear');            %define motion blur
filter
h = h / sum(h(:));                                        %normalizing filter
imgBlur = imfilter(img, h);                               %blur the image
subplot(2,2,2);imshow(imgBlur);title('blurred image');    %plot the blurred
image
imgWnr = deconvwnr(imgBlur, h, 0.001);                    %use built-in deconv
wnr filter
subplot(2,2,3);imshow(imgWnr);title('RestoredByWeiner');  %plot the deconv
result image
%------------------------------------------%
%self-programmed function
H = zeros(size(img,1),size(img,2));                       %degin filter same
size as image
H(1:11,1:18) = h;                                         %locate h values in
the new filter
Hf = fft2(H);                                             %inverse fft for the
new filter
d = 0.1;                                                  %set threshold for
the inverse
```

```
Hf(find(abs(Hf)<d))=1;                                      %replace value below
theshold
imgDeblur = ifft2(fft2(imgBlur)./Hf);                       %inverse filter
deblur method in freq domain
subplot(2,2,4);imshow(mat2gray(abs(imgDeblur)));title('Restored by self-
programmed inverse filter');
                                                            %plot the deconv
result image
```