Computer Vision ENGN4528

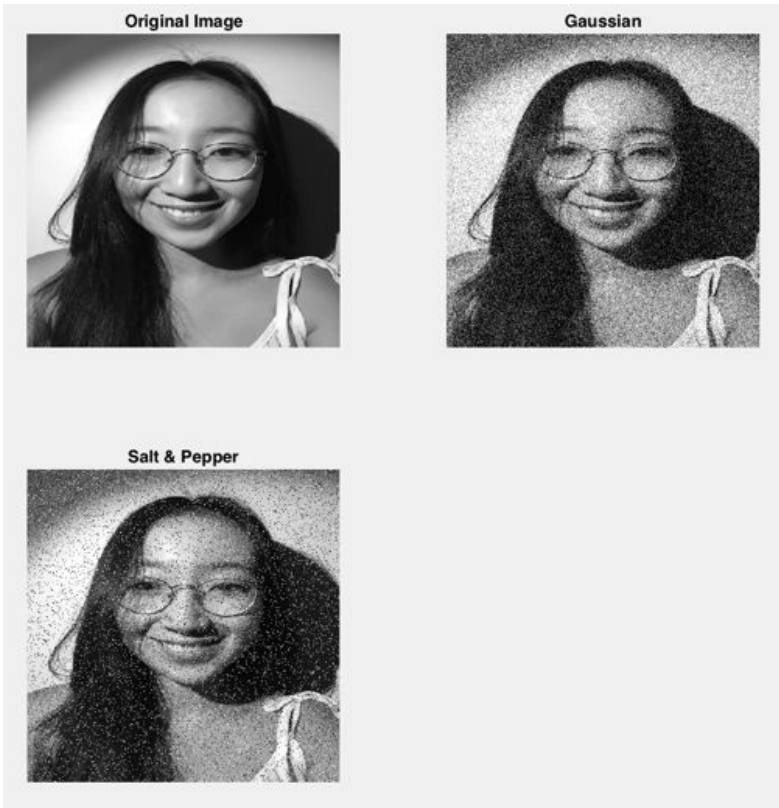
CLAB 01

Yuge Shi
u5634555

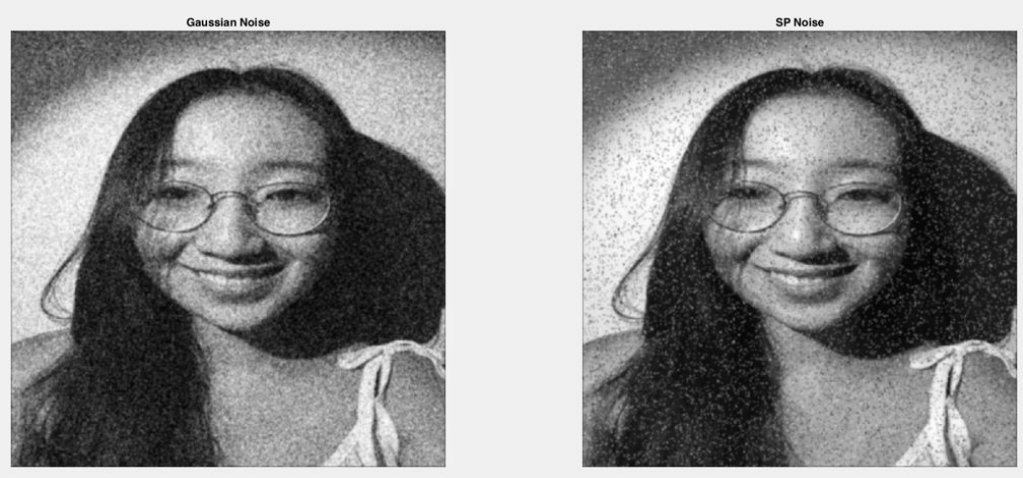
Task 1: Image denoise and filtering:

Step 1: Read in one of your colour face images. Resize it to 512 x 512.

Step 2: Add noise to the image (using MATLAB function “imnoise”). Use the following two different types of noise, in order to generate two different versions of the image.



Step 3: Implement your own Matlab function that performs a 9x9 Gaussian filtering.



Above is the result of de-noise with Gaussian filter with zero mean and standard variation of 1 (in each pixel). **It can be clearly seen that Gaussian filter has better noise-removal effect on Gaussian noise, although still is not ideal.**

To calculate the noise-removal effect on both noise with Gaussian filter, SNRs of image with Gaussian noise, Salt and Pepper noise and their de-noise images are calculated as follow:

SNRgaussian_noise = 22.7293;

SNRsp_noise = 21.2977;

SNRgaussian_gaussian_filtered = 42.6112;

SNRsp_gaussian_filtered = 41.7072;

Step 4: Implement your own 3x3 median filter in Matlab.

Above is the result of de-noise with Median filter. Top two images are filtered by self-implemented Median filter and the bottom two are filtered by built-in Median filter. **It is clear from the result that Median filter is more suitable for filtering SP noise and has a decent effect on it.**

To calculate the noise-removal effect on both noise with Median filter, SNRs of image with Gaussian noise, Salt and Pepper noise and their de-noise images are calculated as follow:

SNRgaussian_noise = 22.7293;

SNRsp_noise = 21.2977;

SNRgaussian_self_median_filtered = 43.9065;

SNRsp_self_median_filtered = 59.4166;

SNRgaussian_builtin_median_filtered = 37.9725;

SNRsp_self_median_filtered = 63.3292;

Q&A: Which filter is better for removing Salt & Pepper noise? Why?

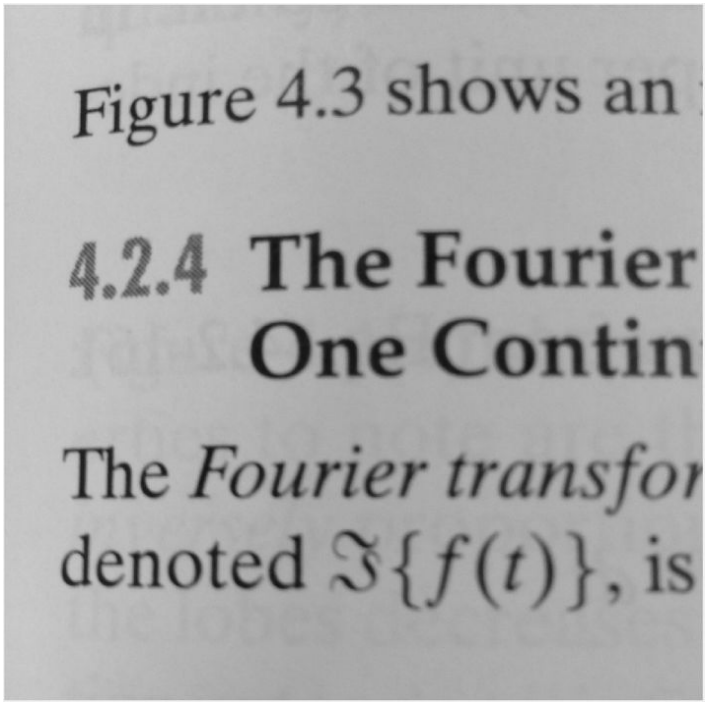
As mentioned above, Median filter has a better for removing SP noise. This can also be concluded by the SNR value of both results. Instead of blurring pixels like Gaussian filter does, Median filter can get rid of the pixels that are too different in its neighbourhood, thus more suitable for removing Salt & Pepper noise.

Step 5: Implement your own Sobel edge detector, compare the result with buit-in edge detector.

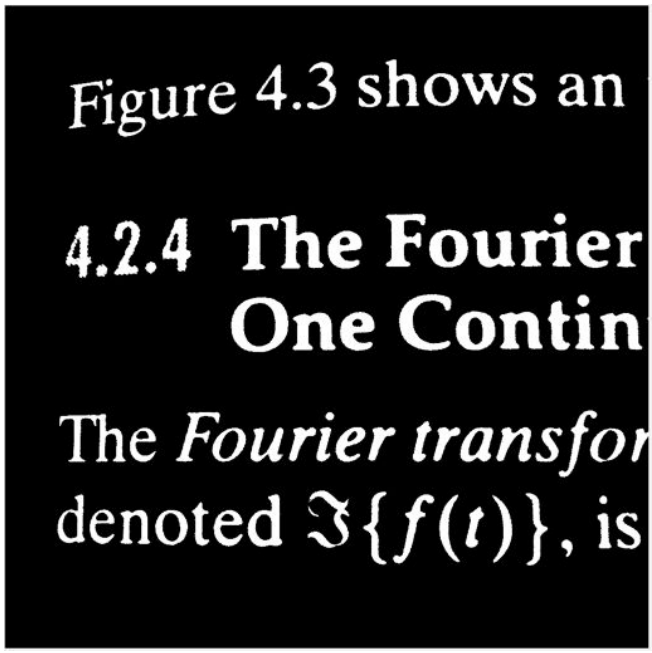
It can be seen from above image that the built-in edge detector, though suffered from some discontinuity on the edges, only highlights the edge of the image. Thus, it can be concluded that the built-in Sobel edge detector has a better effect on detecting the edge of an image.

Task 2: Morphology:

Step 1: Convert the image and resize to 1024x1024；



Step 2: Threshold the histogram to obtain binary image；



Threshold of the binary is set as 0.5.

Step 3: Count the number of white pixels. Count the number of black pixels. Sum up the two numbers above. Is this image a one mega-pixel image?

Number of White Pixels = 107404;

Number of Black Pixels = 941172;

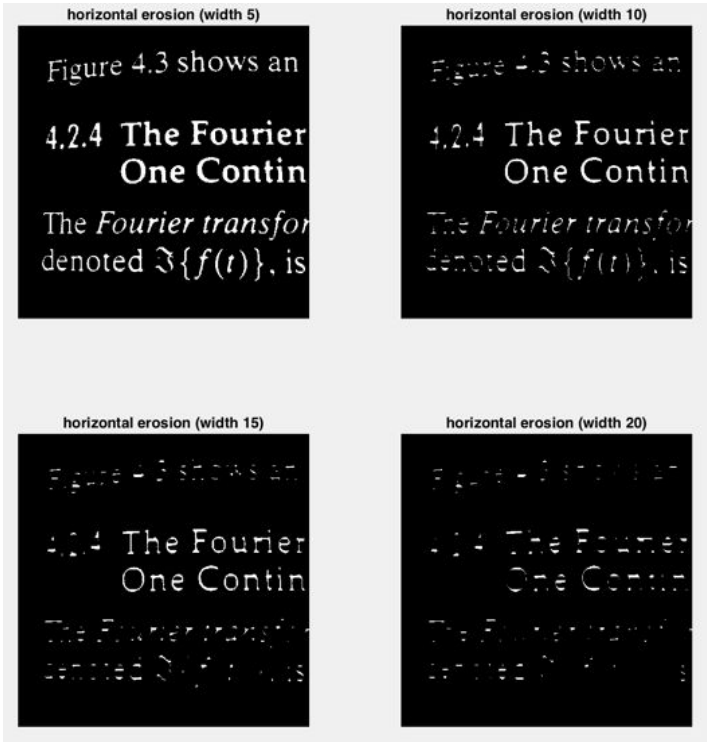
Sum = 1048576>1000000;

As a result, the image is a one megapixel image.

Step 4: Test the effect of erosion, dilation, closing and opening.

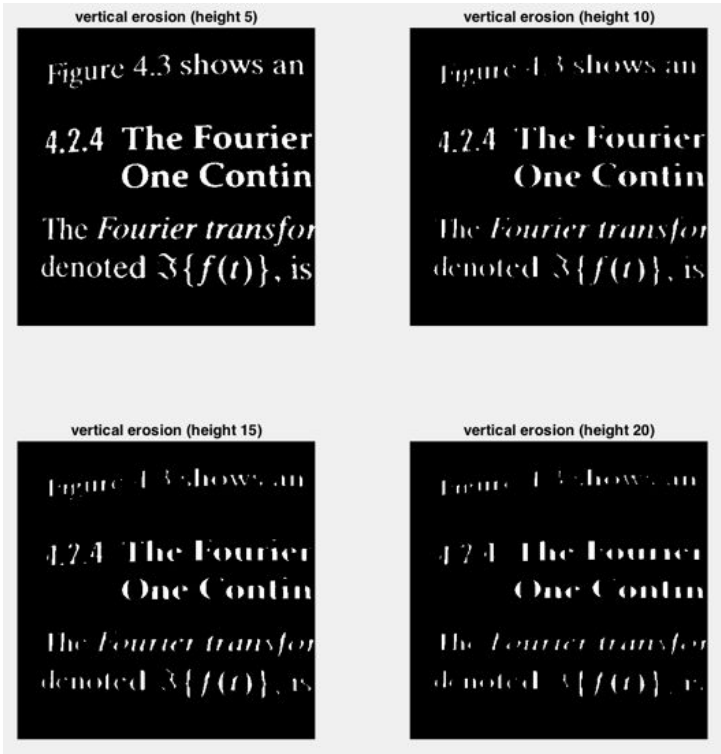
4.1. Test the effect of erosion

4.1.1. Horizontal erosion with different size of vector



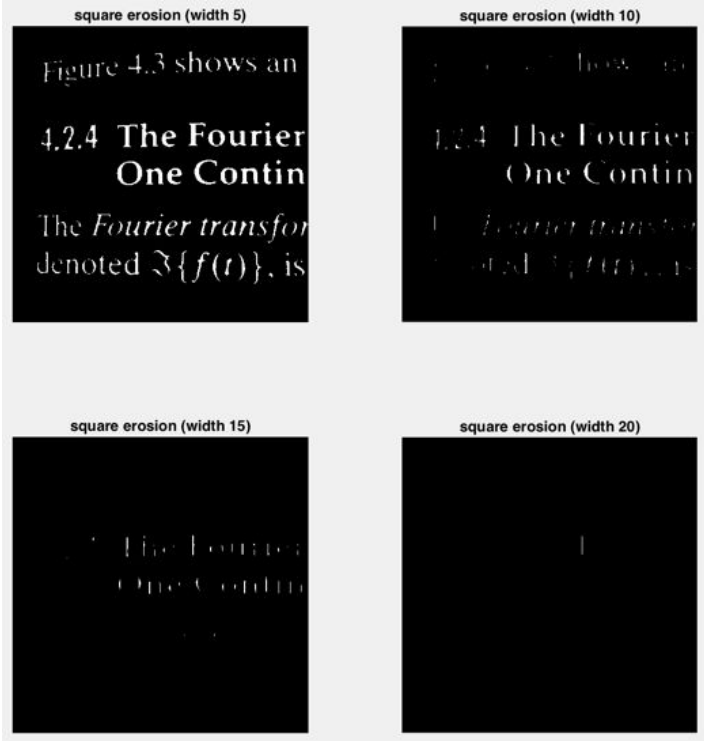
Effect: erasing pixels horizontally. It can be seen from the above image that with the increasing of vector size, the erosion effect gets stronger. The effect on vertical direction is more visible.

4.1.2. Vertical erosion with different size of vector



Effect: erasing pixels vertically. It can be seen from the above image that with the increasing of vector size, the erosion effect on horizontal pixels gets stronger. The effect on horizontal direction is more visible.

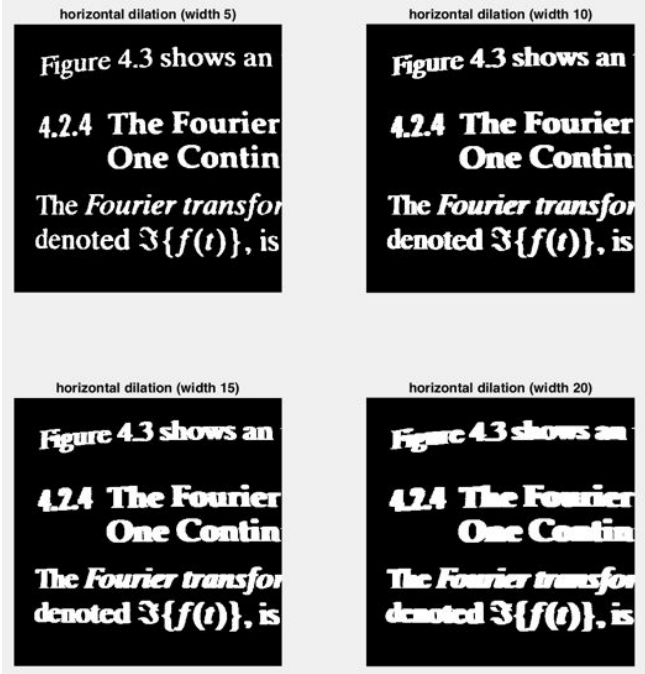
4.1.3. Square erosion with different size of vector



Effect: erasing pixels in the shape of a square. It can be seen from the above image that with the increasing of matrix size, the erosion effect gets stronger.

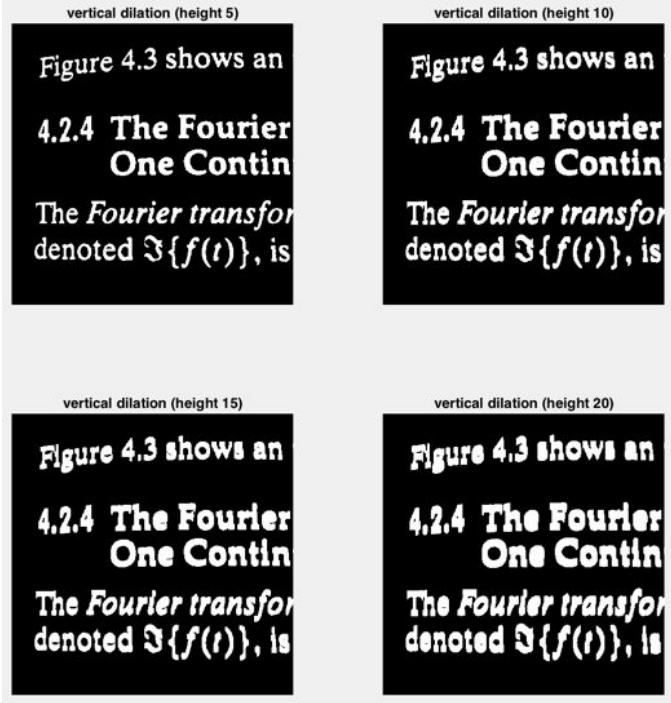
4.2. Test the effect of dilation

4.1.1. Horizontal dilation with different size of vector



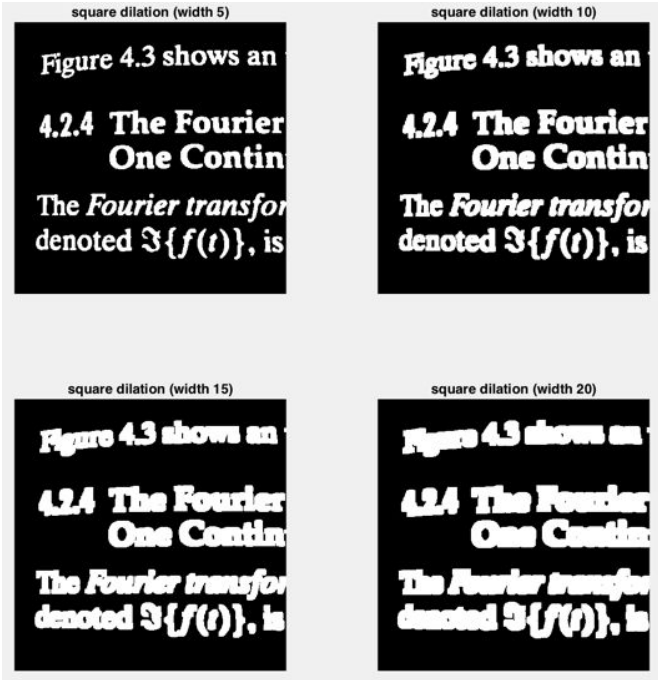
Effect: dilate pixels horizontally. It can be seen from the above image that with the increasing of vector size, the dilation effect on vertical pixels gets stronger. The effect on vertical direction is more visible.

4.1.2. Vertical dilation with different size of vector



Effect: dilating pixels vertically. It can be seen from the above image that with the increasing of vector size, the dilation effect on horizontal pixels gets stronger. The effect on horizontal direction is more visible.

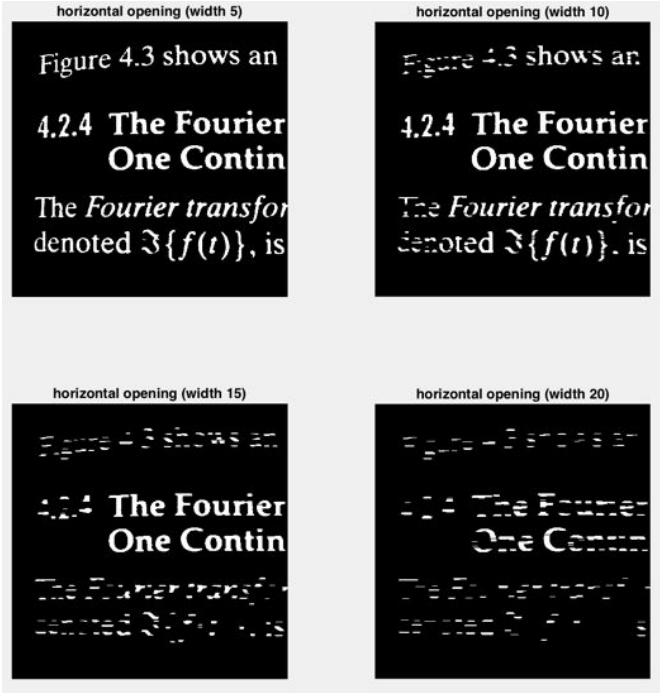
4.1.3. Square dilation with different size of vector



Effect: dilating pixels in the shape of a square. It can be seen from the above image that with the increasing of matrix size, the dilation effect gets stronger.

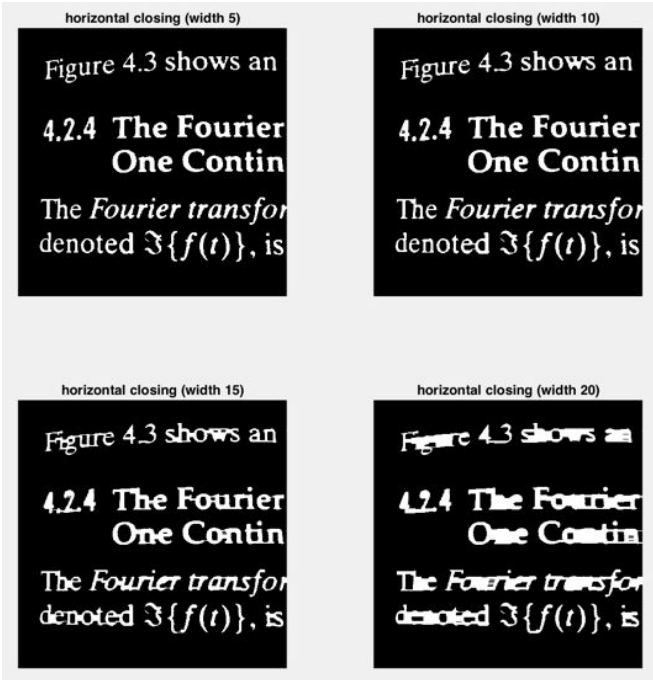
4.3. Test the effect of opening

As introduced in the lecture, opening is an operation includes erosion and dilation. With dilation implemented after erosion, the effect of opening would be erasing pixels. The results of horizontal opening with different size of vector can be seen as follow:



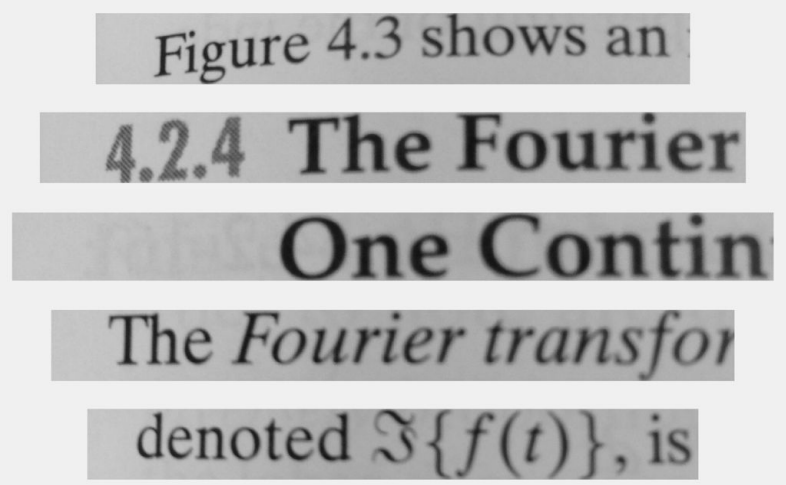
4.4. Test the effect of closing

Similarly to opening, closing is an operation includes erosion and dilation. The only difference between closing and opening is the sequence of erosion and dilation. As a result, the effect of opening is equivalent to that of dilation. The results of horizontal closing with different size of vector can be seen as follow:



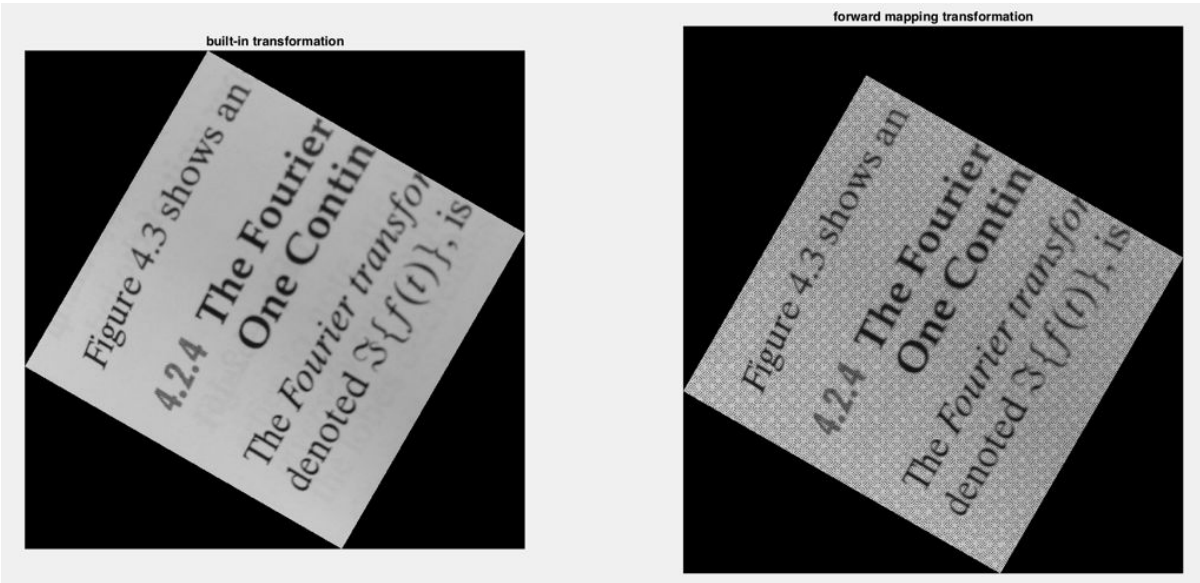
Step 5: Design and program a morphology-based algorithm that can automatically segment each of the text lines from the image. In other words, input the particular given image "text.png", your algorithm will automatically extract five sub-images containing the five text lines.

The result of this task is as follow:



Task 3: Geometric Transformation:

Forward mapping was first implemented in the function, the result of 60° translation and the translation with MATLAB built-in function is as follow:



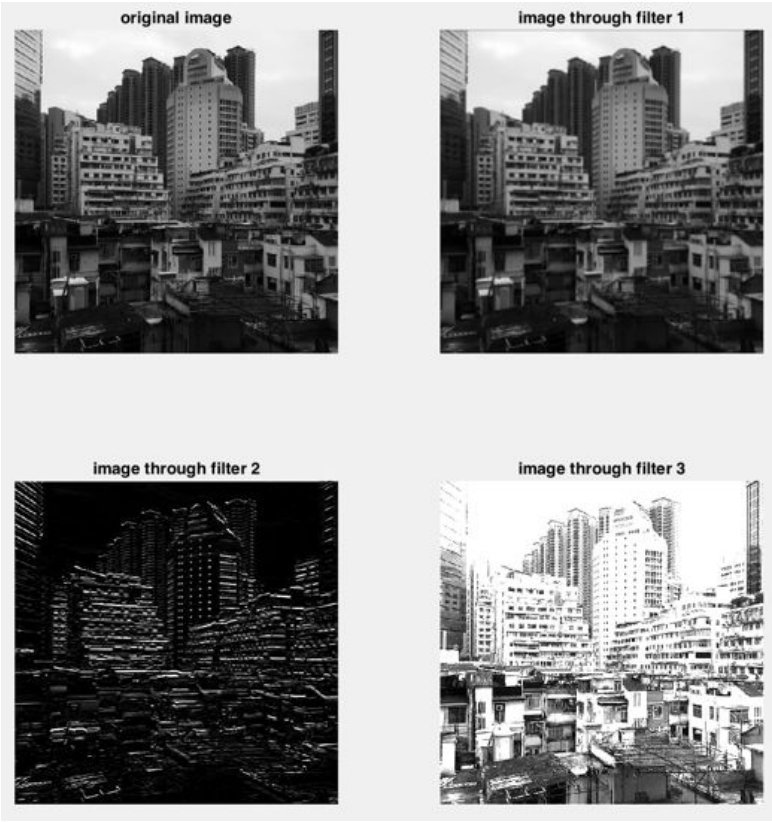
It can be observed from the above that there are quite a few pixels that didn't get mapped to in the forward mapping translation. In order to acquire more high-quality results, inverse mapping was then implemented in the function. The comparison between the result of inverse mapping and built-in translation is as follow:

As shown above, the result of inverse-mapping translation is almost as good as the built-in translation.

The reason that inverse-mapping is much prefered than forward-mapping is that with forward-mapping, the input coordinates are projected to output, whilst in inverse-mapping the situation is the other way around. Mapping output pixels onto input ensures that each pixel in the output gets mapped to and thus better quality of result can be acquired.

Task 4: 2D FFT and IFFT of image:

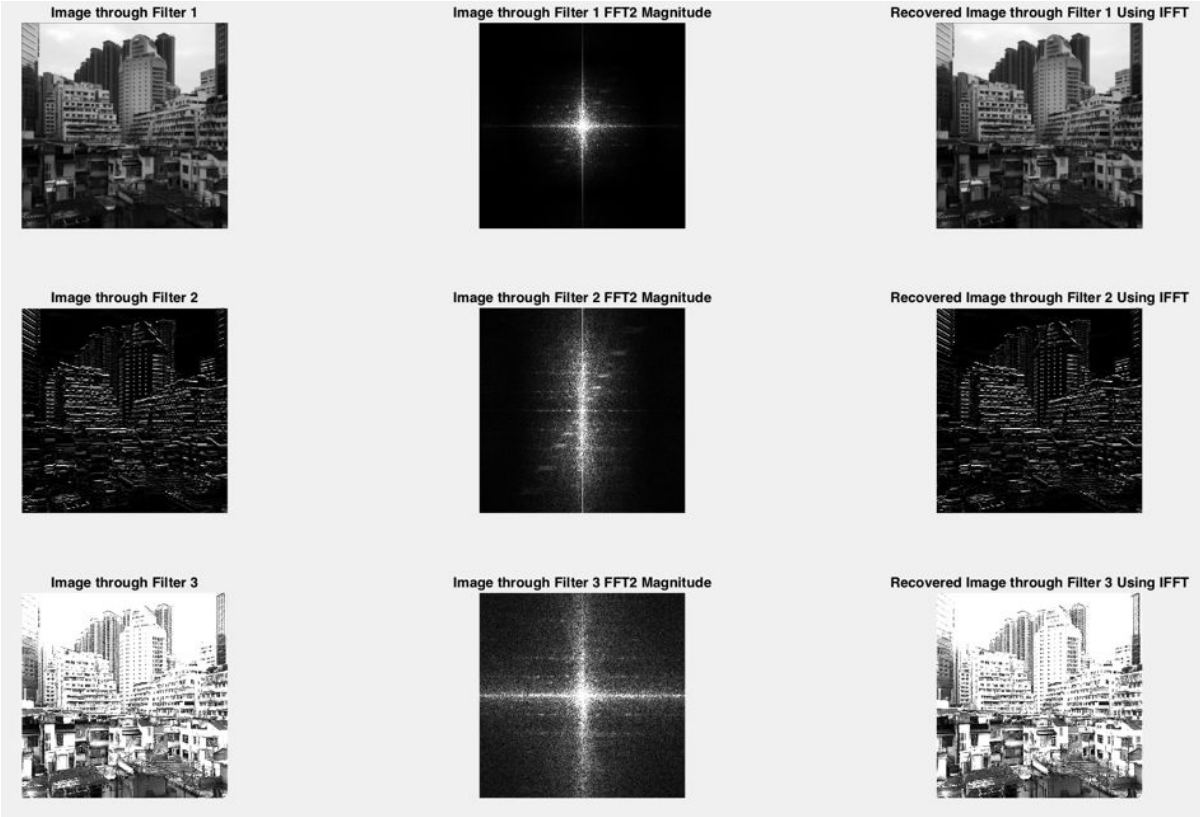
Step 1: Apply three filters to an image and explain the effect of the filters.



Filter 1 (Low Pass Filter): Blur the image;

Filter 2 (Band Pass Filter): Find the edge of the image;
Filter 3 (High Pass FIlter): Sharpen the image.

Step 2&3: Apply FFT and IFFT to the filtered images.
The result is as seen as follow:



Task 5: Image Deblur

Result of image deblur is as shown below (NSR = 0.01):

Comparison between the self-implemented deblur function and built-in Wiener function is shown below (NSR = 0.01):

It can be seen from the graph above that the self-implemented Wiener filter has almost the same effect with the built-in filter.

Appendix A: Code for Task 1

Main function:

```
close all
clear all
%% Step1: convert to gray scale & resize
im = imread('3.jpg');
im = imresize(im,[512,512]);
im = rgb2gray(im);
imshow(im);

%% Step2: add noise & display
img = imnoise(im,'gaussian',0,10/255);
imsp = imnoise(im,'salt & pepper',0.1);
subplot(2,2,1),imshow(im),title('Original Image')
subplot(2,2,2),imshow(img),title('Gaussian')
subplot(2,2,3),imshow(imsp),title('Salt & Pepper')

%% gauss filter
figure()
gaussk = fspecial('gaussian', [9 9],1);
gauss_outputg = my_Gauss_filter(img,gaussk);
gauss_outputsp = my_Gauss_filter(imsp,gaussk);2
subplot(1,2,1),imshow(gauss_outputg),title('Gaussian Noise')
subplot(1,2,2),imshow(gauss_outputsp),title('SP Noise')
Gauss_SNRg1 = 20*log ( norm(double(im),'fro') /norm(double(im) - double(img), 'fro' ))
Gauss_SNRg2 = 20*log ( norm(double(im),'fro') /norm(double(im) - double(gauss_outputg),
'fro' ))
Gauss_SNRsp1 = 20*log ( norm(double(im),'fro') /norm(double(im) - double(imsp), 'fro' ))
Gauss_SNRsp2 = 20*log ( norm(double(im),'fro') /norm(double(im) -
double(gauss_outputsp), 'fro' ))

%% median filter
figure()
median_outputg = my_Median_filter(img);
median_outputsp = my_Median_filter(imsp);
subplot(2,2,1),imshow(median_outputg),title('Gaussian Noise with My Median Filter')
subplot(2,2,2),imshow(median_outputsp),title('SP Noise with My Median Filter')
builtin_outputg = medfilt2(img,[3 3]);
builtin_outputsp = medfilt2(imsp, [3 3]);
subplot(2,2,3),imshow(builtin_outputg) ,title('Gaussian Noise with built-in filter')
subplot(2,2,4),imshow(builtin_outputsp) ,title('SP Noise with built-in filter')
Median_SNRg = 20*log ( norm(double(im),'fro') /norm(double(im) - double(median_outputg),
'fro' ))
Median_SNRsp = 20*log ( norm(double(im),'fro') /norm(double(im) -
double(median_outputsp), 'fro' ))
```

```
Builtin_SNRg = 20*log ( norm(double(im),'fro') /norm(double(im) - double(builtin_outputg),
'fro' ))
Builtin_SNRsp = 20*log ( norm(double(im),'fro') /norm(double(im) - double(builtin_outputsp),
'fro' ))

%% edge detector
figure()
output = Sobel_edgeDetector(im);
subplot(1,2,1),imshow(output),title('Result with my edge detector');
subplot(1,2,2),imshow(edge(im,'Sobel')),title('Result with built-in edge detector');
```

Gaussian Filter:

```
function output_image = my_Gauss_filter(input_image,my_99_gausskernel)
        % have the matrix ready for convolution
        gauss = rot90(my_99_gausskernel,2);

        % create edge of 4 pixels around the original image
        im = uint8(zeros(520,520));
        for i = 5:516
           for j = 5:516
              im(i,j) = input_image(i-4,j-4);
           end
        end

        % apply convolution
        for i = 5:516
           for j = 5:516
              matrix99 = im((i-4):(i+4),(j-4):(j+4));
              pixel = im2double(matrix99)*255.*gauss;
              output_image(i-4,j-4) = uint8(sum(sum(pixel)));
           end
        end
end
```

Median Filter:

```
function  output_image = my_Median_filter(input_image)
   for i = 2:511
      for j = 2:511
         a = [];
         % put all 9 pixel in the range in one vector
         for i_ker = -1:1
            for j_ker = -1:1
               a = [a,input_image(i+i_ker,j+j_ker)];
            end
```

```
            end
            % sort the vector and find the mid-value pixel
            a = sort(a);
            input_image(i,j) = a(5);
        end
    end
    output_image = input_image;
end
```

Sobel Edge Detector:

```
function output_image = Sobel_edgeDetector(input_image)
    % create edge of 1 pixel around the original image
    im = uint8(zeros(514,514));
    for i = 2:513
        for j = 2:513
            im(i,j) = input_image(i-1,j-1);
        end
    end
    sobel = [-1,0,1;-2,0,2;-1,0,1];

    % get the Sobel matrix ready for convolution
    % on both direction (horizontal & vertical)
    sobel_conv_h = rot90(sobel,2);
    sobel_conv_v = rot90(sobel,3);

    % apply convolution on both direction
    for i = 2:513
        for j = 2:513
            matrix33 = im((i-1):(i+1),(j-1):(j+1));
            horizontal = im2double(matrix33)*255.*sobel_conv_h;
            vertical = im2double(matrix33)*255.*sobel_conv_v;
            % combine the results of horizontal edge detection
            % and vertical detection to acquire the final result
            pixel = sqrt(sum(sum(horizontal))^2+sum(sum(vertical))^2);
            output_image(i-1,j-1) = uint8(pixel);
        end
    end
end
```

Appendix B: Code for Task 2

Read image and convert to binary:

```
%% read image, resize, convert to greyscale
im = imread('text.png');
im = imresize(im,[1024,1024]);
im = rgb2gray(im);
imshow(im)
figure()
%% convert to binary image
b_im = im2bw(im,0.5);
b_im = ~b_im;
imshow(b_im)
figure()
%% count white & black pixel respectively
count_w = size(find(b_im == 1))
count_b = size(find(b_im == 0))
```

Morphology:

```
%% erosion
% horizontal erosion with different matrix width
se_h1 = strel('line',5,0);
erodedb_im_h1 = imerode(b_im,se_h1);
se_h2 = strel('line',10,0);
erodedb_im_h2 = imerode(b_im,se_h2);
se_h3 = strel('line',15,0);
erodedb_im_h3 = imerode(b_im,se_h3);
se_h4 = strel('line',20,0);
erodedb_im_h4 = imerode(b_im,se_h4);
subplot(2,2,1),imshow(erodedb_im_h1),title('horizontal erosion (width 5)')
subplot(2,2,2),imshow(erodedb_im_h2),title('horizontal erosion (width 10)')
subplot(2,2,3),imshow(erodedb_im_h3),title('horizontal erosion (width 15)')
subplot(2,2,4),imshow(erodedb_im_h4),title('horizontal erosion (width 20)')
figure()
```

Code for vertical erosion, square erosion, horizontal & vertical & square dilation, opening and closing are similar to the above chunk and will not be list in here.

Segment text lines:

```
% segment text lines
% read image, resize, convert to binary
im = imread('text.png');
im = imresize(im,[1024,1024]);
im = rgb2gray(im);
b_im = im2bw(im,0.5);
b_im = ~b_im; %make foreground 1 and background 0
```

```
% dilate the image with a vector of width 3000
% so that the letters on the same line will connect
% to each other
se_h = strel('line',3000,0);
dilated_b_im= imdilate(b_im,se_h);
% mark each area in the image
L = bwlabel(dilated_b_im,8);
s = size(L);

% crop the image into 5 pieces using different marks
[x1f,y1f] = ind2sub(size(L),find(L==1,1,'first'));
[x1l,y1l] = ind2sub(size(L),find(L==1,1,'last'));
img1 = im(x1f:x1l,y1f:y1l);

[x2f,y2f] = ind2sub(size(L),find(L==2,1,'first'));
[x2l,y2l] = ind2sub(size(L),find(L==2,1,'last'));
img2 = im(x2f:x2l,y2f:y2l);

[x3f,y3f] = ind2sub(size(L),find(L==3,1,'first'));
[x3l,y3l] = ind2sub(size(L),find(L==3,1,'last'));
img3 = im(x3f:x3l,y3f:y3l);

[x4f,y4f] = ind2sub(size(L),find(L==4,1,'first'));
[x4l,y4l] = ind2sub(size(L),find(L==4,1,'last'));
img4 = im(x4f:x4l,y4f:y4l);

[x5f,y5f] = ind2sub(size(L),find(L==5,1,'first'));
[x5l,y5l] = ind2sub(size(L),find(L==5,1,'last'));
img5 = im(x5f:x5l,y5f:y5l);

subplot(5,1,1),imshow(img1);
subplot(5,1,2),imshow(img2);
subplot(5,1,3),imshow(img3);
subplot(5,1,4),imshow(img4);
subplot(5,1,5),imshow(img5);
```

Appendix C: Code for Task 3

Forward-mapping translation:

```
close all
clear all
im = imread('text.png');
im = imresize(im,[1024,1024]);
im = rgb2gray(im);

theta = input('Input the rotation angle:');
% transform from degree to radian
theta = theta*pi/180;

% forward mapping
for i = 1:1024
   for j = 1:1024
      coor = [cos(theta),-sin(theta);sin(theta),cos(theta)]*[i;j];
      coor(1) = coor(1)+1025;
      im_rotate_forward(round(coor(1)),round(coor(2))) = im(i,j);
   end
end
imshow(im_rotate_forward)
im1 = imrotate(im,theta*180/pi);
```

Inverse-mapping translation:

```
%inverse mapping
coor = [];
b1 = 0;
b2 = 0;
b3 = 0;
b4 = 0;
rmax = 0;
rmin = 0;
cmax = 0;
cmin = 0;
% find the minimum and maximum value of position of row and column
% after transformation
for i = 1:1024
    for j = 1:1024
       b1 = rmax;
       b2 = rmin;
       b3 = cmax;
       b4 = cmin;
       a = [cos(theta),-sin(theta);sin(theta),cos(theta)]*[i;j];
       rmax = max(a(1),b1);
       rmin = min(a(1),b2);
```

```
        cmax = max(a(2),b3);
        cmin = min(a(2),b4);
    end
end
for i = round(rmin):round(rmax)
    for j = round(cmin):round(cmax)
        coor = [cos(theta),sin(theta);-sin(theta),cos(theta)]*[i;j];
        coor = round(coor);
        ij = [i-round(rmin)+1,j-round(cmin)+1];
        if 1 <= coor && coor<=1024
            im_rotate_inverse(ij(1),ij(2)) = im(coor(1),coor(2));
        else
            im_rotate_inverse(ij(1),ij(2)) = uint8(0);
        end
    end
end
```

Plotting:

```
subplot(1,2,1), imshow(im1), title('built-in transformation');
subplot(1,2,2), imshow(im_rotate_forward),title('forward mapping transformation');
figure()
subplot(1,2,1),imshow(im_rotate_inverse), title('inverse mapping translation');
subplot(1,2,2), imshow(im1),title('built-in translation');
```

Appendix D: Code for Task 4

Main function (filter part):

```
close all
clear all
im = imread('urban.jpg');
%% filter 1 (low pass filter)
kernel1 = [1 ,1,1; 1,1,1; 1,1,1]*1/9 ;
output1 = my_filter(im,kernel1);
%% filter 2 (band pass filter)
kernel2 =  [1,1,1; 0,0,0 ; -1,-1,-1]  ;
output2 = my_filter(im,kernel2);
%% filter 3 (high pass filter)
kernel3 =   [ 0,-1, 0; -1 , 8, -1; 0,-1,0]   ;
output3 = my_filter(im,kernel3);
%% display
subplot(3,3,1),imshow(output1),title('Image through Filter 1');
subplot(3,3,4),imshow(output2),title('Image through Filter 2');
subplot(3,3,7),imshow(output3),title('Image through Filter 3');
```

Filter function:

```
function output_image = my_filter(input_image,filter_kernel)
        % create edge of width 1 pixel around the image
        im = uint8(zeros(514,514));
        for i = 2:513
           for j = 2:513
              im(i,j) = input_image(i-1,j-1);
           end
        end
        kernel_conv = rot90(filter_kernel,2); % get the kernel ready for convolution
        % convolution
        for i = 2:513
           for j = 2:513
              matrix33 = im((i-1):(i+1),(j-1):(j+1));
              pixel = im2double(matrix33)*255.*kernel_conv;
              output_image(i-1,j-1) = uint8(sum(sum(pixel)));
           end
        end
end
```

FFT:

```
%% fft
fftA = fft2(double(output1));
fftB = fft2(double(output2));
fftC = fft2(double(output3));
```

```
subplot(3,3,2), imshow(abs(fftshift(fftA)),[0 100000])
title('Image through Filter 1 FFT2 Magnitude')
subplot(3,3,5), imshow(abs(fftshift(fftB)),[0 100000])
title('Image through Filter 2 FFT2 Magnitude')
subplot(3,3,8), imshow(abs(fftshift(fftC)),[0 100000])
title('Image through Filter 3 FFT2 Magnitude')

output1_fft = ifft2(fftA);
Amin = min(min(abs(output1_fft)));
Amax = max(max(abs(output1_fft)));
subplot(3,3,3), imshow(abs(output1_fft),[Amin Amax])
title('Recovered Image through Filter 1 Using IFFT')

output2_fft = ifft2(fftB);
Bmin = min(min(abs(output2_fft)));
Bmax = max(max(abs(output2_fft)));
subplot(3,3,6), imshow(abs(output2_fft),[Bmin Bmax])
title('Recovered Image through Filter 2 Using IFFT')

output3_fft = ifft2(fftC);
Cmin = min(min(abs(output3_fft)));
Cmax = max(max(abs(output3_fft)));
subplot(3,3,9), imshow(abs(output3_fft),[Cmin Cmax])
title('Recovered Image through Filter 3 Using IFFT')
```

Appendix E: Code for Task 5

```
clear all
close all

% blur the image
im = imread('urban.jpg');
theta = 30; len = 20;
fil = imrotate(ones(1, len), theta, 'bilinear');
fil = fil / sum(fil(:));
im2 = imfilter(im, fil);
subplot(1,2,1),imshow(im2),title('before deblur');

% deblur the image by implementing Wiener filtering
fftfil = fft2(double(fil),512,512); % transfer both image and blurring filter to FFT domain
fftim2 = fft2(double(im2));
ffth = (conj(fftfil))./(abs(fftfil).^2+0.01); % apply Wiener filter
fftout = fftim2.*ffth;
out = ifft2(fftout);
outmin = min(min(out));
outmax = max(max(out));
subplot(1,2,2),imshow(out,[0 outmax+outmin]), title('after deblur');

figure()
wnr1 = deconvwnr(im2, fil, 0.01); % built-in Wiener filter
subplot(1,2,1),imshow(wnr1),title('built-in Wiener deblur');
subplot(1,2,2),imshow(out,[0 outmax+outmin]),title('my Wiener deblur');
'
```