DIP Assignment Report

yelantingfeng
December 31, 2017

0 INTRODUCTION

Interest in digital image processing methods stems from two principal application areas: improvement of pictorial information for human interpretation; and processing of image data for storage, transmission, and representation for autonomous machine perception[1].

This report is for the 10 assignments of digital image processing(CS386) course in SJTU, instructed by Prof. Hongtao Lu. Files submitted together with this report are organized in this way, where two file folders ***images***, ***outputs*** contain all used input and output images for all 10 assignments respectively and the folder ***codes*** is for all source codes of these 10 assignments. Each assignment is finished by implementing some *.m* function with no use of the built-in image processing function in Matlab. To see good results which are similar to those presented in textbook, you can directly run the ***testdemo.m*** script in every single subfolder in folder ***codes***. These scripts will invoke those *.m* functions with proper parameters and present the results in Matlab's figure windows. If you think those images cannot be clearly browsed, you can also see the output images in ***outputs*** where the result images are saved as *.png* or *.pdf* files in advance.

This report will introduce the the whole 10 assignments sequentially. For each assignment, this essay will discuss its principle, implementation and analyze its outputs. We are trying to focus on the part which is critical or not mentioned in textbook. To fully understand the ideas of digital image processing, you are still required to consult the textbook.

1 HISTOGRAM EQUALIZATION

1.1 PRINCIPLE

Histogram equalization is a kind of intensity transformation which can make the PDF of an image appears more uniformly. With its intensity uniformly distributed, an image gets higher contrast and can be seen more clearly. The transformation is

$$s_k = T(r_k) = \frac{L-1}{MN}\sum_{j=0}^{k} n_j, \quad k = 0,1,2,\ldots,L-1 \tag{1.1}$$

where MN is the total number of pixels in the image, n_k is the number of pixels with intensity r_k, and L is the gray level of this image. If we apply this equation to an image, and then use s_k to replace r_k, we can then get our result image.

1.2 IMPLEMENTATION

To finish this assignment, 3 functions are defined. Function **a_histogram** is used to compute the histogram of an image. It has the general syntax

[frequency, intensity] = a_histogram(image, bitnum, plotflag)

It will compute the frequency of each intensity in **image** with $2^{\textbf{bitnum}}$ as its gray level. If **plotflag** is set true, it will plot the bar chart with **frequency** and **intensity** as its Y-axis and X-axis respectively. You can also invoke another function **plotbar** manually to plot the chart with command

plotbar(intensity,frequency,bitnum)

The third function is **b_equalization** and its syntax is

[resImg, transY, transX] = b_equalization(rawImg, bitnum, plotflag)

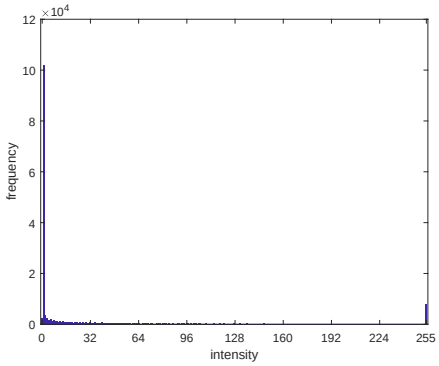
This function will invoke **a_histogram** to calculate the histogram of **rawImg** and then do the transformation. **transY, transX** will be used to plot the transformation function. By specifying a parameter **bitnum**, these functions can be applied to images with different gray level.

1.3 RESULT ANALYSIS

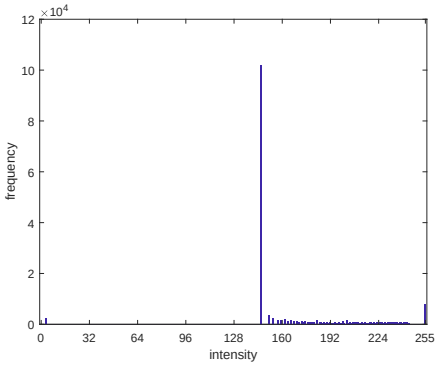
After running the script **testdemo.m**, you will get following results.

original image

enhanced image



histogram of original image



histogram of enhanced image

Figure 1.1: Result of **fig1.jpg**

As we can see in Figure 1.1, for the first test image **fig1.jpg**, this method doesn't work well enough. Though more details can be seen in enhanced image, it looks overexposed since there is a large white area

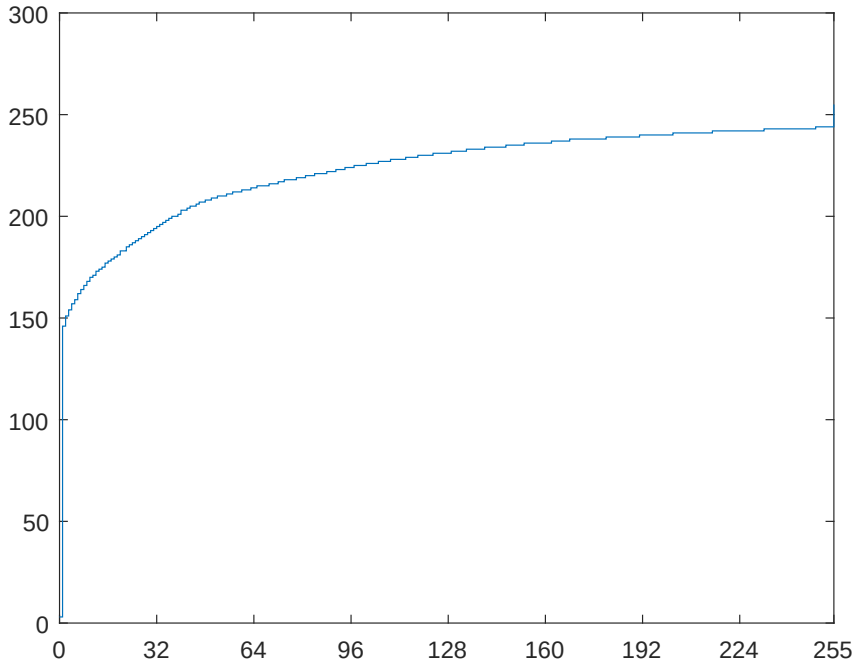
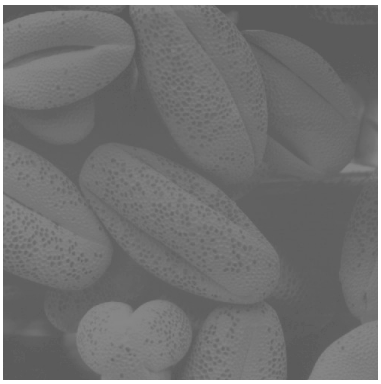
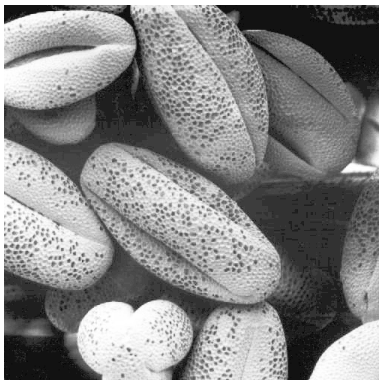


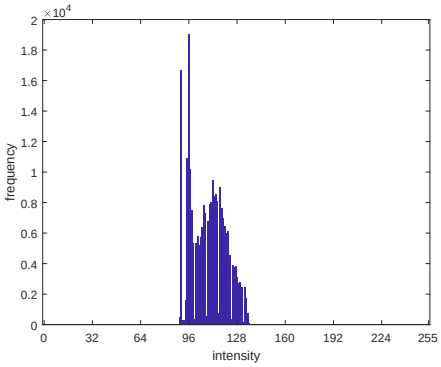
Figure 1.2: Transformation function of **fig1.jpg**



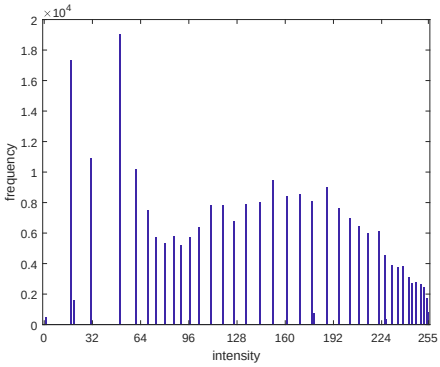
original image



enhanced image



histogram of original image



histogram of enhanced image

Figure 1.3: Result of **fig2.jpg**

in it. This is because we are dealing with discrete values. You can also find this problem correspondingly by observe the histogram plot of enhance image. Figure 1.2 shows the transformation function.

The second test image **fig2.jpg** performs better, as you can see in Figure 1.3. After histogram equalization, the histogram of the image becomes much more uniform. The enhanced image looks accordingly better than that of **fig1.jpg**. The result of this test image can be found as **FIGURE 3.20** in the textbook. Our result is nearly the same with that. The transformation function is displayed in Figure 1.4.

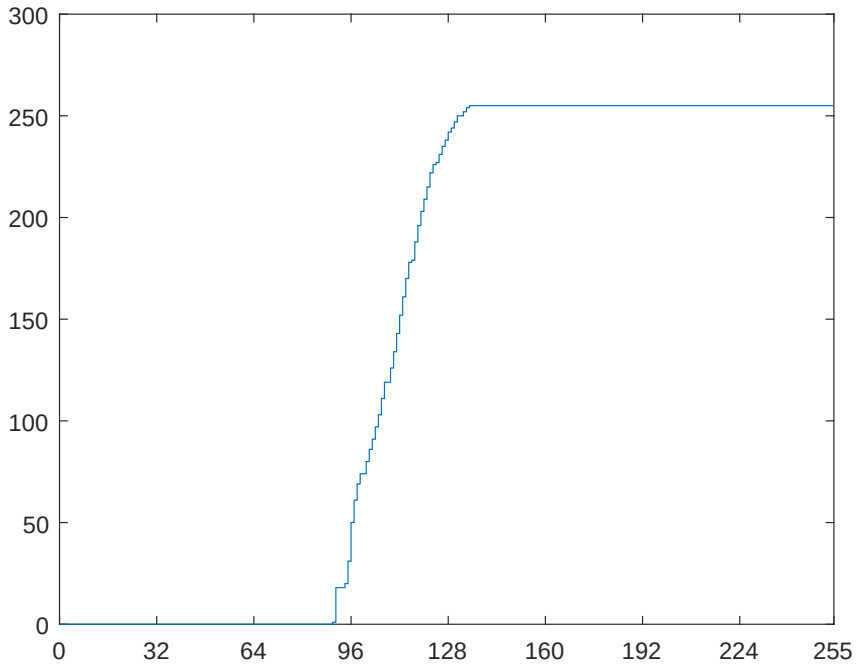


Figure 1.4: Transformation function of **fig2.jpg**

2 Combining Spatial Enhancement Methods

2.1 Principle

In this assignment, we are required to enhance an image by using diffrent kinds of enhancement methods sequentially. These methods include Laplacian, Sobel gradient, averaging filter, power-law transformation and simple array operation(addtion and multiplication).

Laplacian and gradient are two types of sharpening filters which are based on first-order and second-order derivatives respectively. Laplacian of two variables is

$$\nabla^2 f(x,y) = f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y) \tag{2.1}$$

The gradient of $f(x,y)$ is a vector which defined as

$$\nabla f \equiv grad(f) \equiv \begin{bmatrix} g_x \\ g_y \end{bmatrix} \equiv \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \tag{2.2}$$

Sobel gradient calculate g_x and g_y in this way where

$$g_x = \frac{\partial f}{\partial x} = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3) \tag{2.3}$$

and

$$g_y = \frac{\partial f}{\partial y} = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7) \tag{2.4}$$

$z_k(k = 1,2,\ldots,4,6,\ldots,9)$ here denote eight neighbors of $f(x,y) = z_5$. Indices are given from left to right and top to bottomn. Its magnitude can be calculated following

$$M(x,y) = |g_x| + |g_y| \tag{2.5}$$

Averaging filtering is a linear filter where value of each pixel will be the average of its neighbors.
Power-law transformation is a kind of intensity transform with the form

$$s = cr^{\gamma} \tag{2.6}$$

where r is the raw intensity value, and s is the transformation of r.

2.2 Implementation

For power-law transformation and simple array operation, the implementation is quite easy. The critical part of this assignment is to implement a linear filter which can be specified with different masks. Once done, the Laplacian, Sobel gradient and averaging filter can be easily computed.

Linear filter implemented here is a function called **dipLinearFilter**. You can invoke it with the syntax

[resImg] = dipLinearFilter(rawImg, mask)

where **rawImg** is the target image data, and **mask** is a filter mask stored in a 2-D matrix. Output **resImg** is the result of filtering contains values in double type. This function is implemented with zero padded and don not use the built-in function **conv2**. Thus, the speed is a little slower but still tolerable. For masks used for different methods, you can see them in Figure 2.1.

$\frac{1}{25}$

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

5×5 averaging filter

0	-1	0
-1	4	-1
0	-1	0

Laplacian

-1	-2	-1
0	0	0
1	2	1

Sobel gradient(g_x)

-1	0	1
-2	0	2
-1	0	1

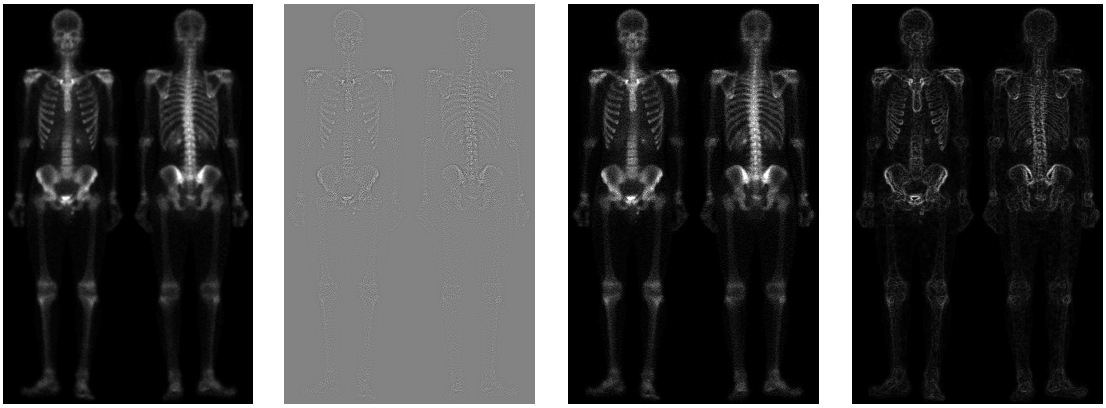
Sobel gradient(g_y)

Figure 2.1: Masks we used

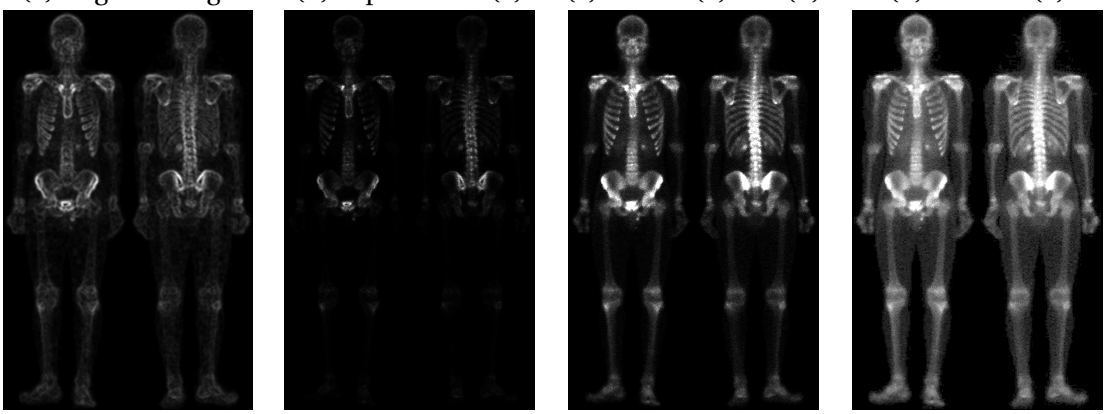
2.3 Resutl Analysis

After running the script **testdemo.m**, you will get following results.

In Figure 2.2, we can see that in the final result (h), we see that significant new detail is visible. The areas around the wrists, hands, ankles, and feet are good examples of this. The skeletal bone structure also is much more pronounced, including the arm and leg bones. Note also the faint definition of the outline of the body, and of body tissue. Bringing out detail of this nature by expanding the dynamic range of the intensity levels also enhanced noise, but it still represents a significant visual improvement over the original image **skeleton_orig.tif**. The result of this test image can be found as **FIGURE 3.43** in the textbook.



(a) original image (b) Laplacian of (a) (c) Sum of (a) and (b) (d) Sobel of (a)



(e) averaging of (d) (f) Product of (c),(e) (g) Sum of (a) and (f) (h) power-law of (g)

Figure 2.2: Enhancement of **skeleton_orig.tif**

3 Filtering in frequency domain

3.1 Principle

Filtering techniques in the frequency domain are based on modifying the Fourier transform to achieve a specific objective and then computing the inverse DFT to get us back to the image domain.

2-D discrete Fourier transform(DFT) is

$$F(u,v) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi(ux/M+vy/N)} \tag{3.1}$$

and the inverse discrete Fourier transform(IDFT) is

$$F(u,v) = \frac{1}{MN}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F(u,v)e^{j2\pi(ux/M+vy/N)} \tag{3.2}$$

Steps for Filtering in the Frequency Domain are as follows:

1. Given an input image of $f(x,y)$ of size $M \times N$, obtain the padding parameters P and Q where $P \geq 2M-1$ and $Q \geq 2N-1$. This is to avoid wraparound error.
2. Form a padded image, $f_p(x,y)$, of size $P \times Q$ by appending the necessary number of zeros to $f(x,y)$.
3. Multiply $f_p(x,y)$ by $(-1)^{x+y}$ to center its transform.
4. Compute the DFT, $F(u,v)$, of the image of step 3.
5. Generate a real, symmetric filter function, $H(u,v)$, of size $P \times Q$ with center at coordinates $(P/2, Q/2)$. Form the product $G(u,v) = H(u,v)F(u,v)$ using array multiplication; that is, $G(i,k) = H(i,k)F(i,k)$.

6. Obtain the processed image:

$$g_p(x,y) = \left\{real\left[\mathscr{F}^{-1}\left[G(u,v)\right]\right]\right\}(-1)^{x+y} \tag{3.3}$$

where the real part is selected in order to ignore parasitic complex components resulting from computational inaccuracies, and the subscript p indicates that we are dealing with padded arrays.

7. Obtain the final processed result, $g(x,y)$, by extracting the $M \times N$ region from the top,left quadrant of $g_p(x,y)$.

We are required to implement 6 kinds of filters in this assignment, 3 of them are lowpass and the other 3 are highpass. All these filters are dependent by the function $D(u,v)$, where

$$D(u,v) = \left[(u-P/2)^2 + (v-Q/2)^2\right]^{1/2} \tag{3.4}$$

Each type of filters are defined as follows:

1. ideal lowpass filter(ILPF)

$$H(u,v) = \begin{cases} 1, & D(u,v) \le D_0 \\ 0, & D(u,v) > D_0 \end{cases} \tag{3.5}$$

2. ideal highpass filter(IHPF)

$$H(u,v) = \begin{cases} 0, & D(u,v) \le D_0 \\ 1, & D(u,v) > D_0 \end{cases} \tag{3.6}$$

3. Butterworth lowpass filter(BLPF)

$$H(u,v) = \frac{1}{1+[D(u,v)/D_0]^{2n}} \tag{3.7}$$

4. Butterworth highpass filter(BHPF)

$$H(u,v) = 1 - \frac{1}{1+[D(u,v)/D_0]^{2n}} = \frac{1}{1+[D_0/D(u,v)]^{2n}} \tag{3.8}$$

5. Gaussian lowpass filter(GLPF)

$$H(u,v) = e^{-D^2(u,v)/2D_0^2} \tag{3.9}$$

6. Gaussian highpass filter(GHPF)

$$H(u,v) = 1 - e^{-D^2(u,v)/2D_0^2} \tag{3.10}$$

3.2 Implementation

To finish the task in this assignment, we first need to implement the DFT and IDFT function. Rather than use the built-in function **fft2** and **ifft2**, I implement my own function **dip2DDFT.m** and **dip2DIDFT.m**. These two functions are implemented as naive DFT and IDFT, instead of FFT and IFFT, which are not introduced in our textbook. However, the implementation are well coded to use matrix multiplication, which can accelerate these functions to a large extent. Though they are still much slower than **fft2** and **ifft2**, but I believe the thinking behind these two functions still counts, which may also be applied to implement FFT and IFFT.

The idea is to use the separability of the 2-D DFT, and then it can be write as

$$F(u,v) = \sum_{x=0}^{M-1} e^{-j2\pi ux/M} \sum_{y=0}^{N-1} f(x,y) e^{-j2\pi vy/N} \tag{3.11}$$

By observing this equation and thinking a little more, you will found that the result can be calculated by constructing 2 matrices **A** and **B**, and then you just need to multiply them, *s.t.*

$$\mathbf{F} = \mathbf{AfB} \tag{3.12}$$

The structure of matrices are displayed in Figure 3.1, where

$$g(u,x) = e^{-j2\pi ux/M} \tag{3.13}$$

and

$$h(y,v) = e^{-j2\pi vy/N} \tag{3.14}$$

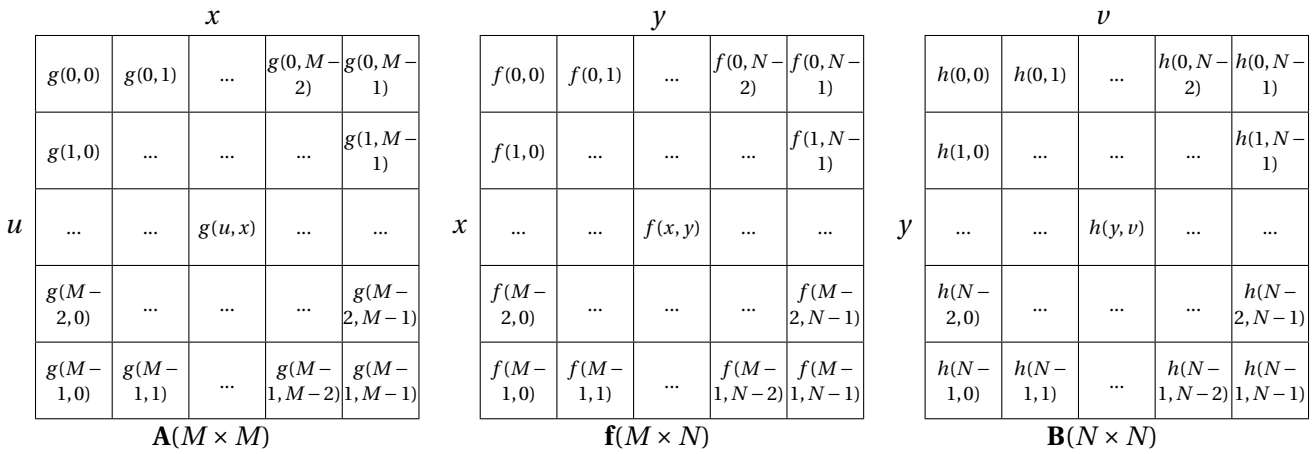


Figure 3.1: Structure of matrices

Once we implement DFT in this way, we can caculate IDFT use DFT as follows:

$$f(x,y) = \frac{1}{MN}\left(\mathscr{F}\left(F^*(u,v)\right)\right)^* \tag{3.15}$$

F^* means the complex conjugate of F.

After DFT and IDFT is implemented, we only need to apply different filter function H to the frequency domain of the image, and every step is done following the steps mentioned above. Finally we can get the expected result.

3.3 Result Analysis

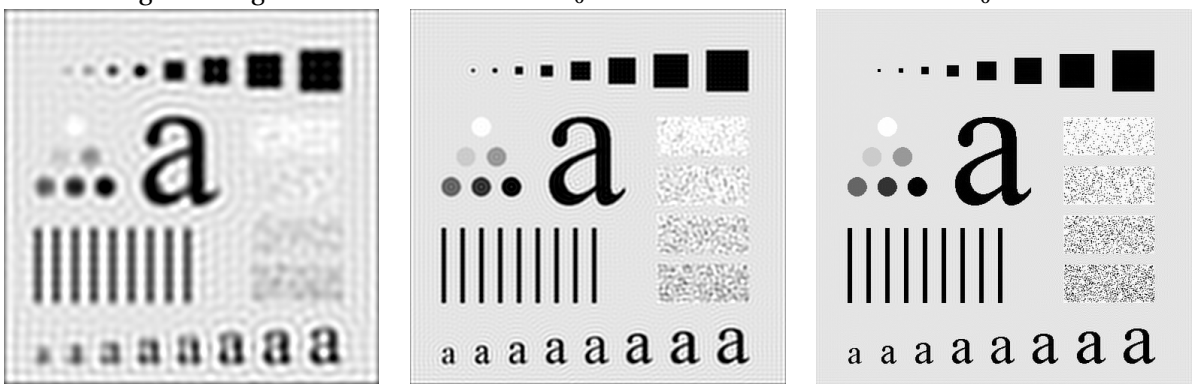
After running the script **testdemo.m**, you will get six groups of results. Each group are result of one kind of filter under different parameters. Only part of results are presented here, you can browse all results by running **testdemo.m** or look into the file folder **./outputs/pro3/**.

As you can see in Figure 3.2, ILPF blurs the original image to different extents when specifying different D_0. The blurring and ringing properties of ILPFs can be explained using the convolution theorem. The result displayed in Figure 3.3 is in frequency domiain, where some circles with different radius and same center point can be easily found. Simlularly, Figure 3.4 and 3.5 presented the result of GLPF. You can find some differences between these two groups of results.

Results of BHPF with $n = 2$ are presented in Figure 3.6 and 3.7. These result images in spatial domain are sharpened to different extents, which is exactly what is highpass filter doing.

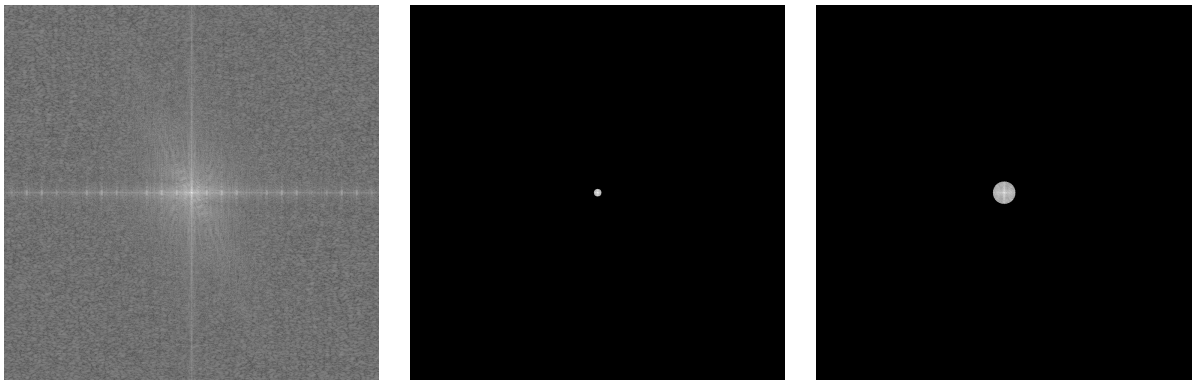
You can compare these results with the images presented in section 4.8-4.9 of the textbook, the similarity between them will announce the correctness of our results.

original image $D_0 = 10$ $D_0 = 30$

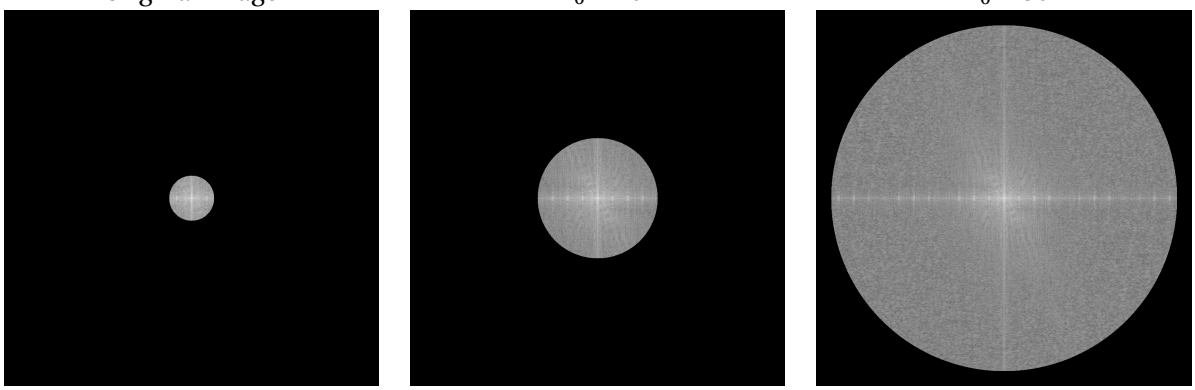


$D_0 = 60$ $D_0 = 160$ $D_0 = 460$

Figure 3.2: ILPF result of **characters_test_pattern.tif** in spatial domain



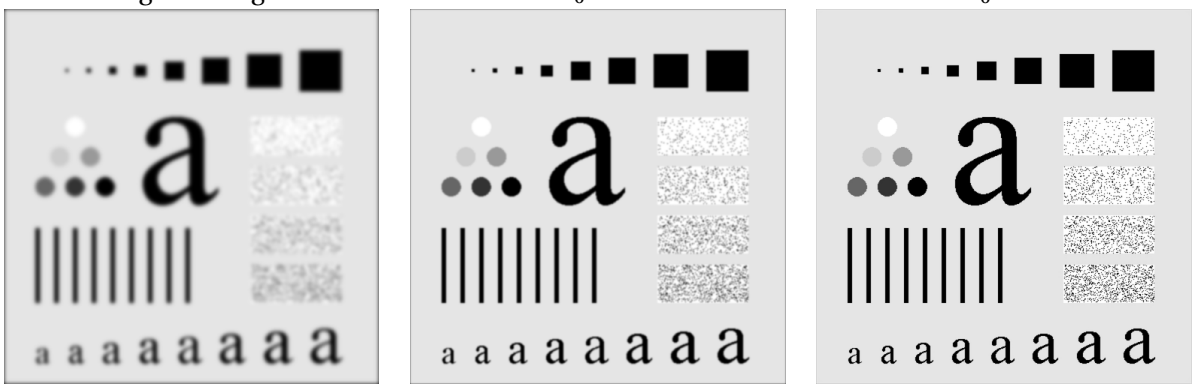
original image $D_0 = 10$ $D_0 = 30$



$D_0 = 60$ $D_0 = 160$ $D_0 = 460$

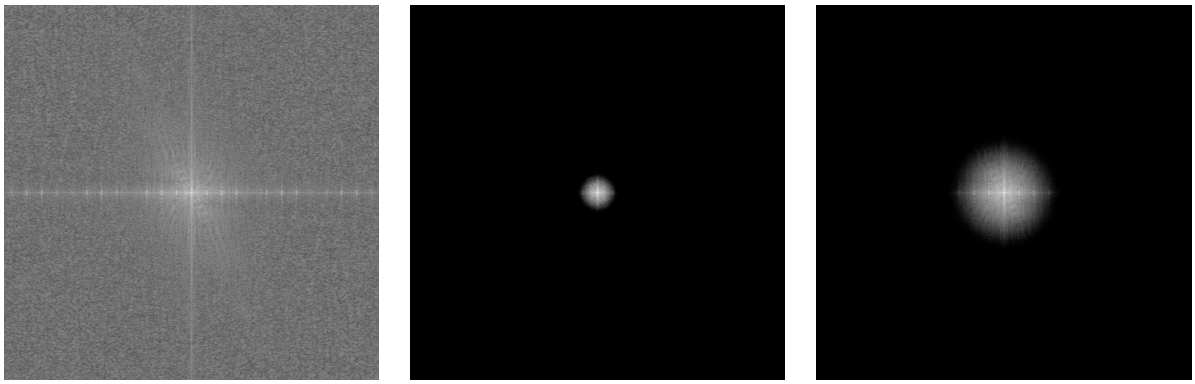
Figure 3.3: ILPF result of **characters_test_pattern.tif** in frequency domain

original image $D_0 = 10$ $D_0 = 30$

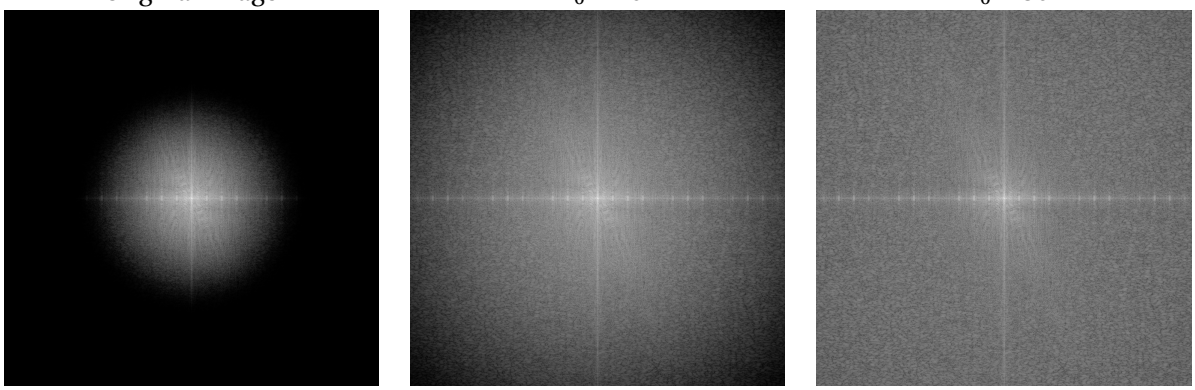


$D_0 = 60$ $D_0 = 160$ $D_0 = 460$

Figure 3.4: GLPF result of **characters_test_pattern.tif** in spatial domain



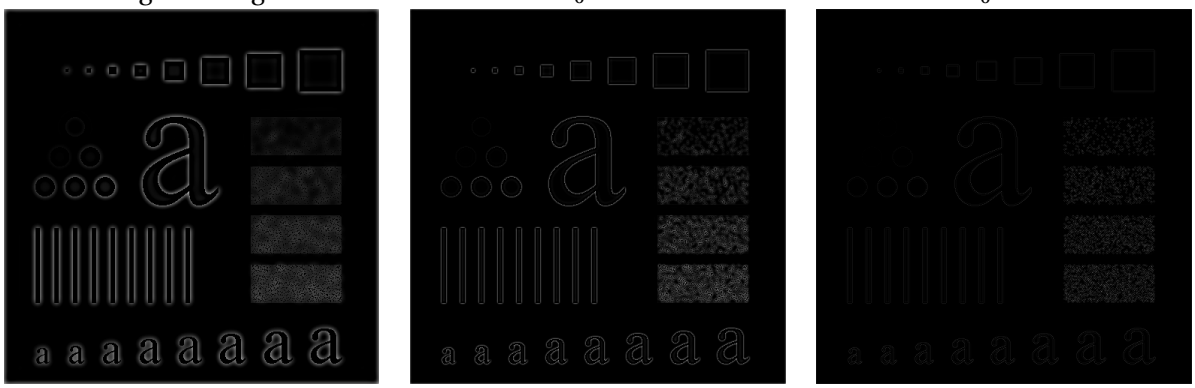
original image $D_0 = 10$ $D_0 = 30$



$D_0 = 60$ $D_0 = 160$ $D_0 = 460$

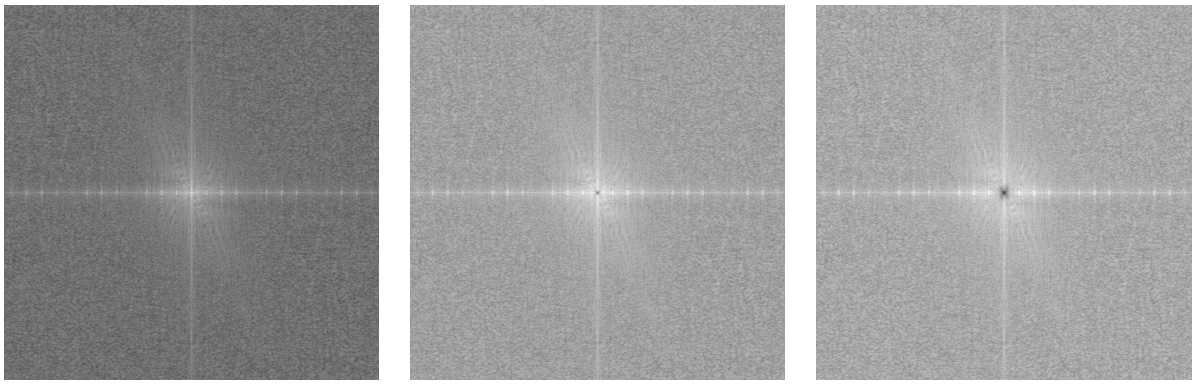
Figure 3.5: GLPF result of **characters_test_pattern.tif** in frequency domain

original image $D_0 = 10$ $D_0 = 30$

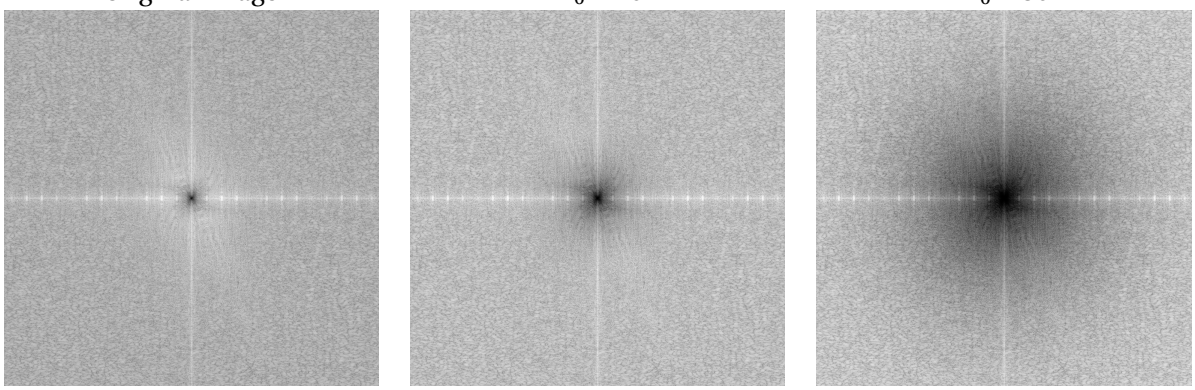


$D_0 = 60$ $D_0 = 160$ $D_0 = 460$

Figure 3.6: BHPF($n = 2$) result of **characters_test_pattern.tif** in spatial domain



original image $D_0 = 10$ $D_0 = 30$



$D_0 = 60$ $D_0 = 160$ $D_0 = 460$

Figure 3.7: BHPF($n = 2$) result of **characters_test_pattern.tif** in frequency domain

4 Noise Generation and Reduction

4.1 Implementation

This assignment requires us to generate noise which yield different distribution function. We need to figure out how to produce a random variable with different distribution function from a uniform random variable. I consulted [2] to find a way.

The common way is to calculate a random variable as follows:

$$X = F_X^{-1}(U) \tag{4.1}$$

where U is a uniform random variable and F_x is the cumulative distribution function of our target random variable X. Take exponential random variable as an example, its CDF is $F_X = 1 - e^{-x/a}, x \geq 0$ and then we get $x = -\frac{1}{a} ln(1-u)$, where us is a uniform random variable and x is our target variable. For rayleigh, things are quite similar.

However, we cannot get the CDF of Gaussian random variable, but I found it has a formula

$$F(x) = \Phi(\frac{x-\mu}{\sigma}) = \frac{1}{2}\left[1 + \text{erf}\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right] \tag{4.2}$$

where $\text{erf}(x) = \frac{2}{\sqrt{2}} \int_0^x e^{-t^2} dt$. This function is built in matlab, and then we can easily implement gaussian noise. However, this is not the only way I found, a usual way we used is an algorithm called Box-Muller method. I have tried this and the distribution is a little different with the one created by the built-in function **randn**. Thus, we choose the first way.

Gamma random variable is defined as the sum of b indenpendent exponential random variable Y_i, each with mean $1/a$. We can use the sum of exponential random variable to generate a Gamma random variable, or we can use uniform variables to generate them directly, which can provide a little acceleration.

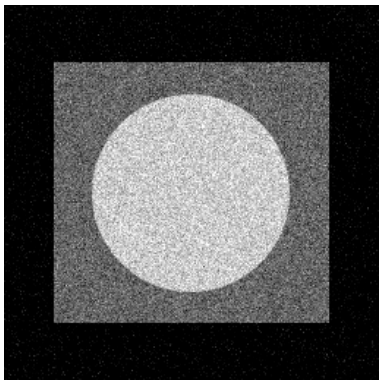
Other part of this assignment are just implemented as the textbook instructed, so we won't talk more about the details.

4.2 Result Analysis

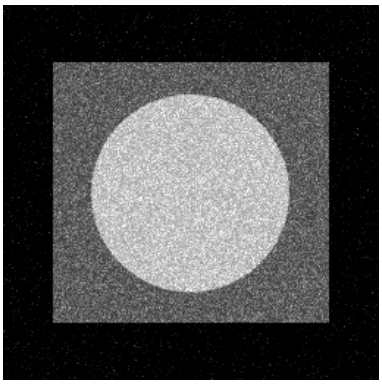
After running the script **testdemo.m**, you will get the following results. Only part of results are presented here, you can browse all results by running **testdemo.m** or look into the file folder **./outputs/pro4**.

As shown in Figure 4.1 and 4.2, our noise generator works well. To get the similar apperance to **FIGURE 5.4** in our textbook, I adjusts the parameters carefully and plot the histogram in a proper range, which is not always [0,255]. This is because some value will extends that range, thus we need to enlarge our display range to see its full apperance.

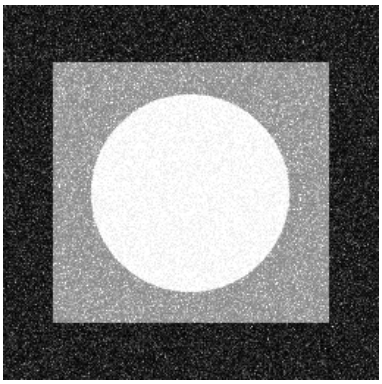
Figure 4.3 presents the result of different kinds of filters applied to an image corrupted with both uniform and pepper&salt noise. As expected, the arithmetic and geometric mean filters (especially the latter) did not do well because of the presence of impulse noise. The median and alpha-trimmed filters performed much better, with the alpha-trimmed filter giving slightly better noise reduction. You can compare with these results with those presented as **FIGURE 5.12** in the textbook. Most of the results are similar, but there are some difference between the two results of geometric filter.



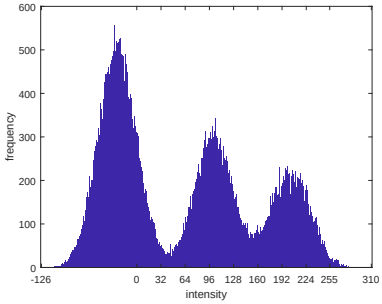
Gaussian noise(a=-0.1,b=0.1)



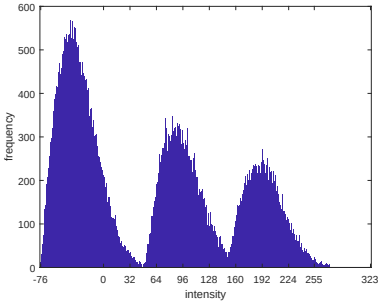
Rayleigh noise(a=-0.3,b=0.04)



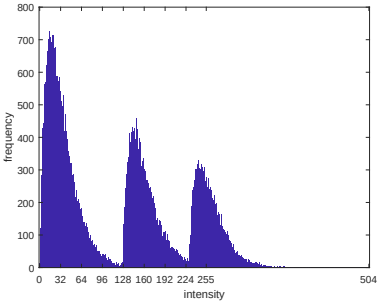
Gamma noise(a=15,b=2)



Histogram of Gaussian noise

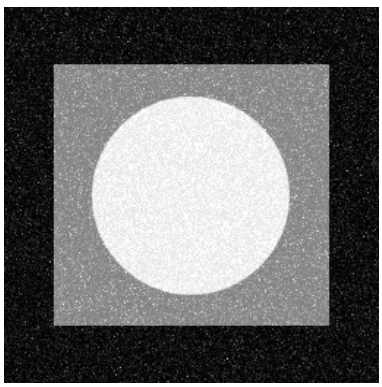


Histogram of Rayleigh noise

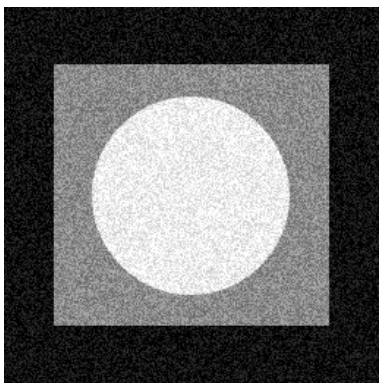


Histogram of Gamma noise

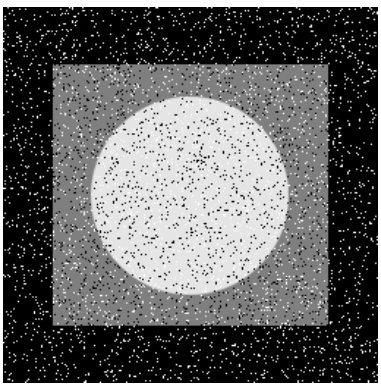
Figure 4.1: Noise generation 1



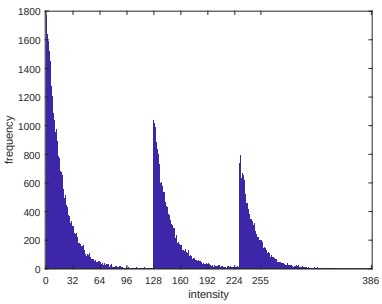
Exponential noise(a=15)



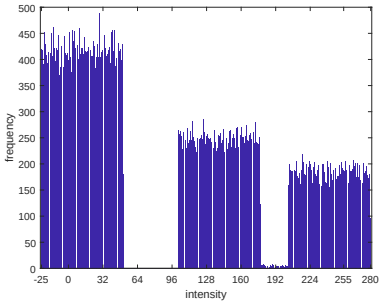
Uniform noise(a=-0.1,b=0.2)



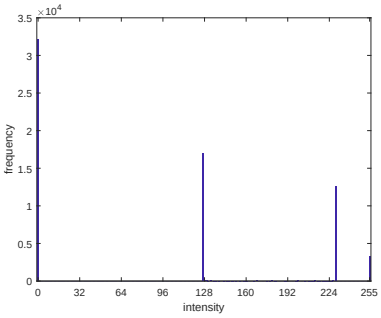
Impulse noise(a=0.05,b=0.05)



Histogram of Exponential noise

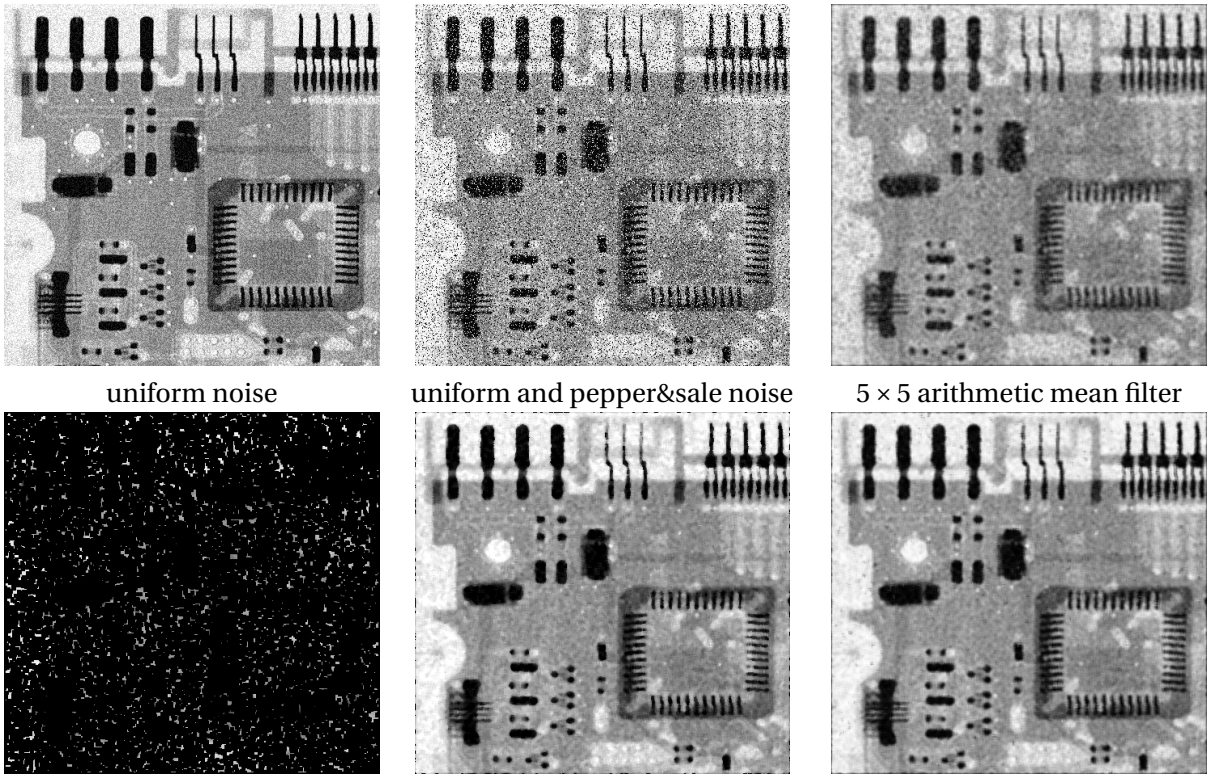


Histogram of Uniform noise



Histogram of Impulse noise

Figure 4.2: Noise generation 2



uniform noise | uniform and pepper&sale noise | 5 × 5 arithmetic mean filter

5 × 5 geometric mean filter | 5 × 5 median filter | 5 × 5 alpha-trimmed filter(d=10)

Figure 4.3: Comparison between different noise reduction mothods

5 Image restoration

5.1 Implementation

In this assignment, I continue using the DFT and IDFT function **dip2DDFT.m** and **dip2DIDFT.m** which implemented in assignment 3. And follow the same steps of frequency filtering as mentioned in section 3. However, to get similar result as **FIGURE 5.26** in textbook presented, we need to circular pad the image itself, *s.t.*, we invoke built-in function **repmat** to repeat the original image 4 times.

For the implementation inverse filter, it will be a problem when there appears zeros or quite small values in the degradation function H. To avoid this, the textbook introduce frequency lowpass filter to reduce the probability of encountering zero values. However, this is not a stable solution and it doesn't work when trying to restore our motion blurred image. The way here I used is adopted from [3]. Now the inverse function is not always $1/H$, when it is too high, the value will be abandoned. That is,

$$H^*(x) = \begin{cases} \frac{1}{H(x)}, & H(x) \geq a \\ 0, & H(x) < a \end{cases} \tag{5.1}$$

where H^* is inverse filter function. The threshold a here in this implementation is set to be 10^{-10}. Finally, I found that if we use threshold way and lowpass filter simultaneously, we will get a very good restoration.

For better comparison, the value K in equation

$$\hat{F}(u,v) = \left[\frac{1}{H(u,v)} \frac{|H(u,v)|^2}{|H(u,v)|^2 + K} \right] G(u,v) \tag{5.2}$$

is calculated as $K = 1/\text{SNR}$.

5.2 Result Analysis

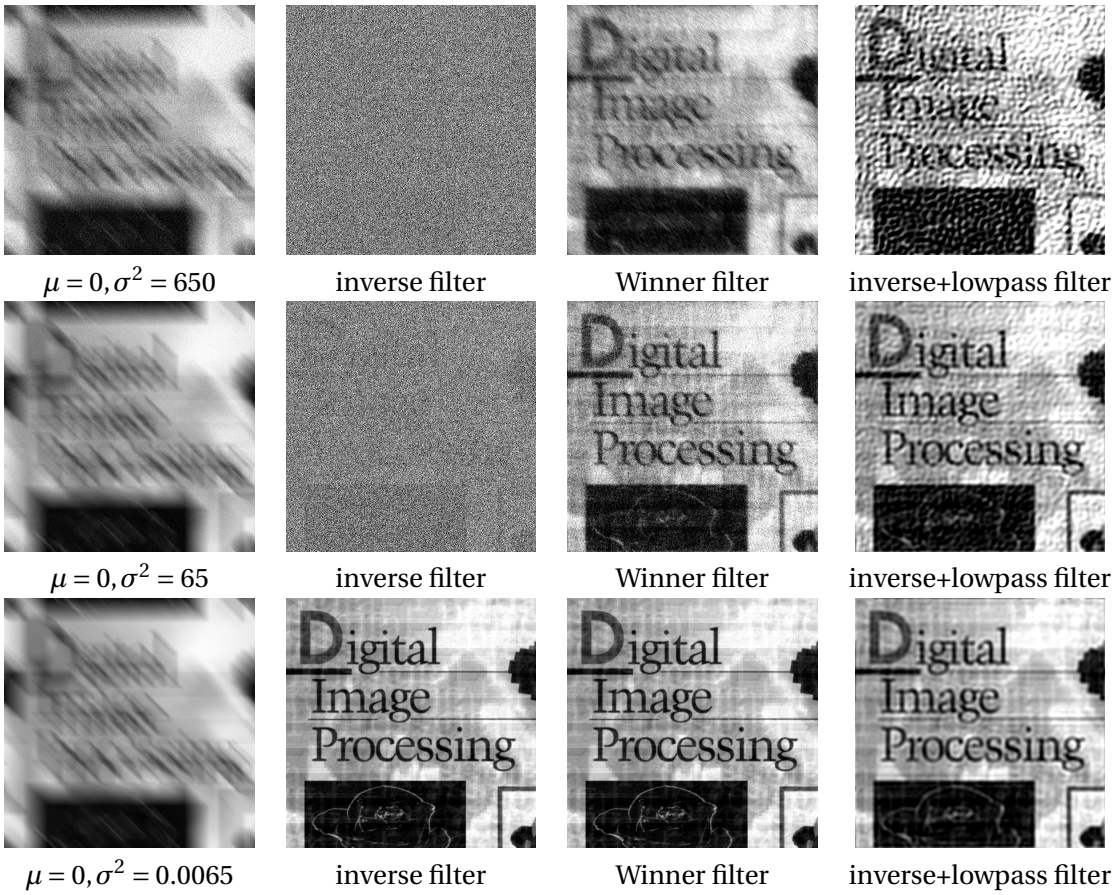
After running the script **testdemo.m**, you will get the following result.

As show in Figure 5.1, our result is similar to those presented as **FIGURE 5.26** in our textbook. However, the parameters are different to get similar result, which are displayed in their caption. Thus, for the rest part of this assignment, we will use $a = b = 0.05, T = 1$ to get our motion blurred image.

Results shown in Figure 5.2 can be a little different with **FIGURE 5.29** in our textbook. This can be caused by using threshold way to implement the inverse filter. Additionly, in the fourth column of the result figure, you can see that using inverse filter with threshold and lowpass filter simultaneously will get a better result than using inverse filter directly.

Original image | Blurred Image(a,b=0.1,T=1) | Blurred Image(a,b=0.05,T=1)

Figure 5.1: Motion blur



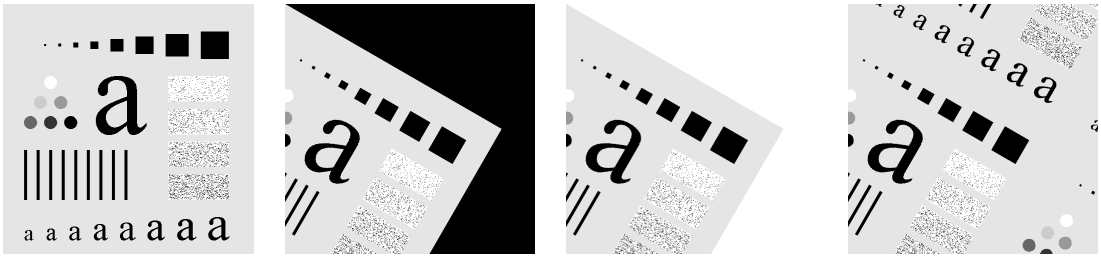
$\mu = 0, \sigma^2 = 650$	inverse filter	Winner filter	inverse+lowpass filter
$\mu = 0, \sigma^2 = 65$	inverse filter	Winner filter	inverse+lowpass filter
$\mu = 0, \sigma^2 = 0.0065$	inverse filter	Winner filter	inverse+lowpass filter

Figure 5.2: Restoration with inverse and Winner filter

After running the script **testdemo.m**, you will get the following result.

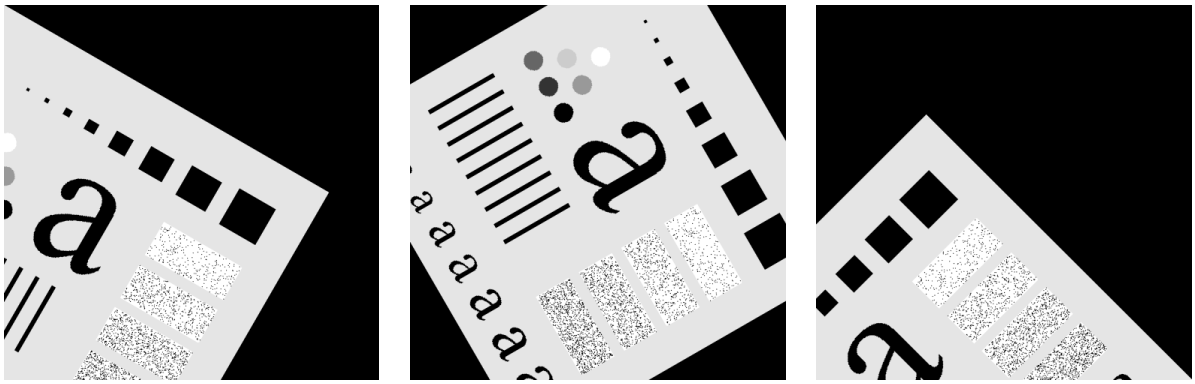
These results shows that our implementation has a good compatibility. The rotation can be applied around different center point and with different angles. Translate with different shift value can all be satisfied. Scaling of image can enlarge or shrink the original image or even flip it with a nagative paramter specified. For ratating and translating, you can choose 3 types of padding, which are padding black, white and images.

By constrasting with two images in Figure 6.5,6.6 and 6.7, especially the first one and the third one, you can see difference between near neighbor interpolation and Bilinear interpolation. The former will appear more coarse and artificiial while the latter will get smoother result.



Original image Black padded White padded Image padded

Figure 6.1: Rotate with different paddings



$c = (0,0), \alpha = 30°$ $c = (M/2, N/2), \alpha = 60°$ $c = (M-1, N-1), \alpha = -45°$

Figure 6.2: Rotate with different center points and angles

Original image Black+(30,30) White(-30.5,40.3) Image+(80,40)

Figure 6.3: Translate image with different pad type and displacement

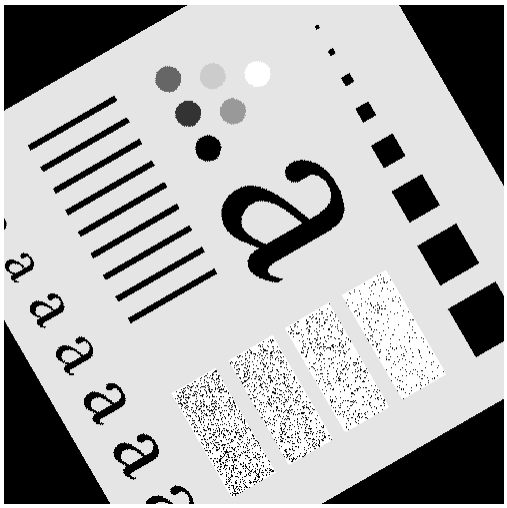
Original image

$\times[0.5, 0.5]$

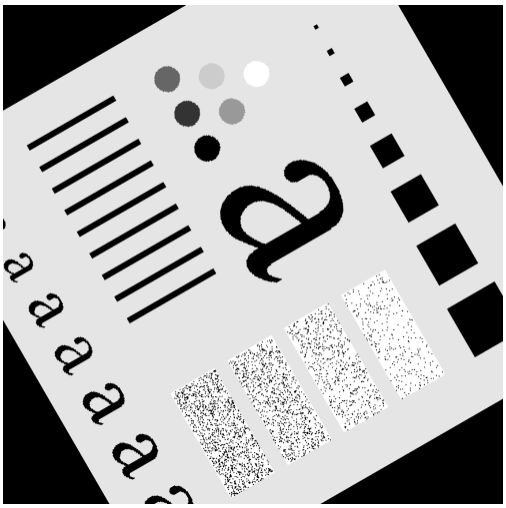
$\times[1.5, 1.5]$

$\times[1, -1]$

Figure 6.4: Scale image with different enlarge parameters

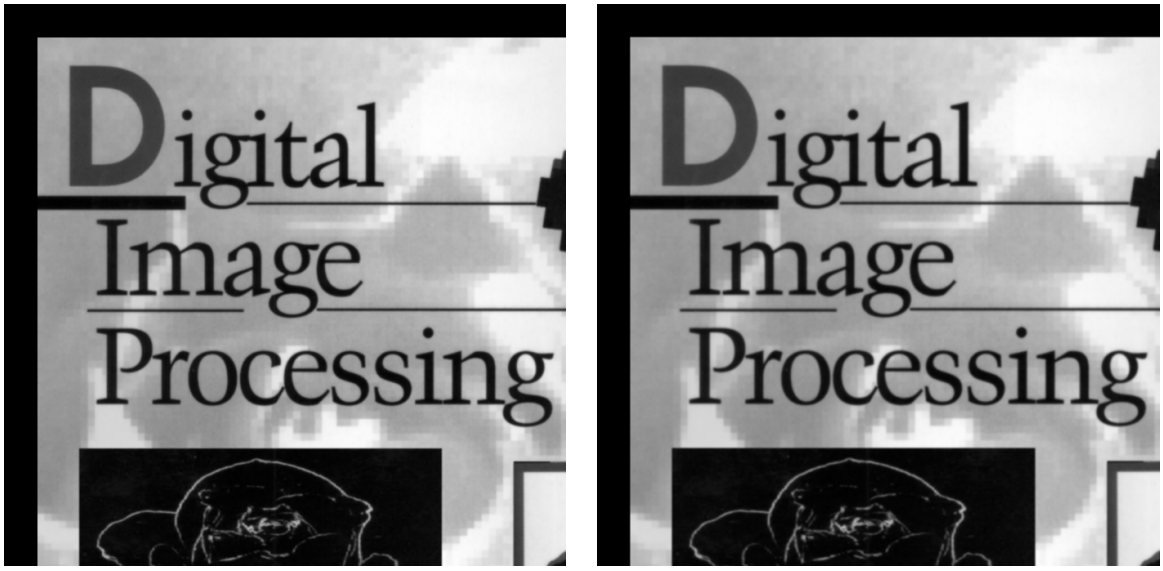


Nearest neighbor



Bilinear

Figure 6.5: Rotate with different interpolation methods



Nearest neighbor Bilinear

Figure 6.6: Translate with different interpolation methods

Nearest neighbor Bilinear

Figure 6.7: Scale with different interpolation methods

7 TRANSFORM IMAGE COMPRESSION

After running the script **testdemo.m**, you will get the following result.

Figure 7.3 shows the result of DCT compression using zonal mask and 7.1,7.2 display the result of same compression with different threshold masks. The threshold mask way is implemented with normalization array. You can easily find that a larger normalization array will get a lower quality image. The implementation of DCT is similar to the DFT in a vectorize way.

The result of DWT compression with different wavelets are displayed in Figure 7.4. The implementation of DWT is fast discrete wavelet transform as the textbook instructs. I use the built-in function **conv2** to speed up the whole process, since the convolution implementation is not the critical part of this assignment.

Figure 7.1: DCT compression using Threshold mask(with normalization array)

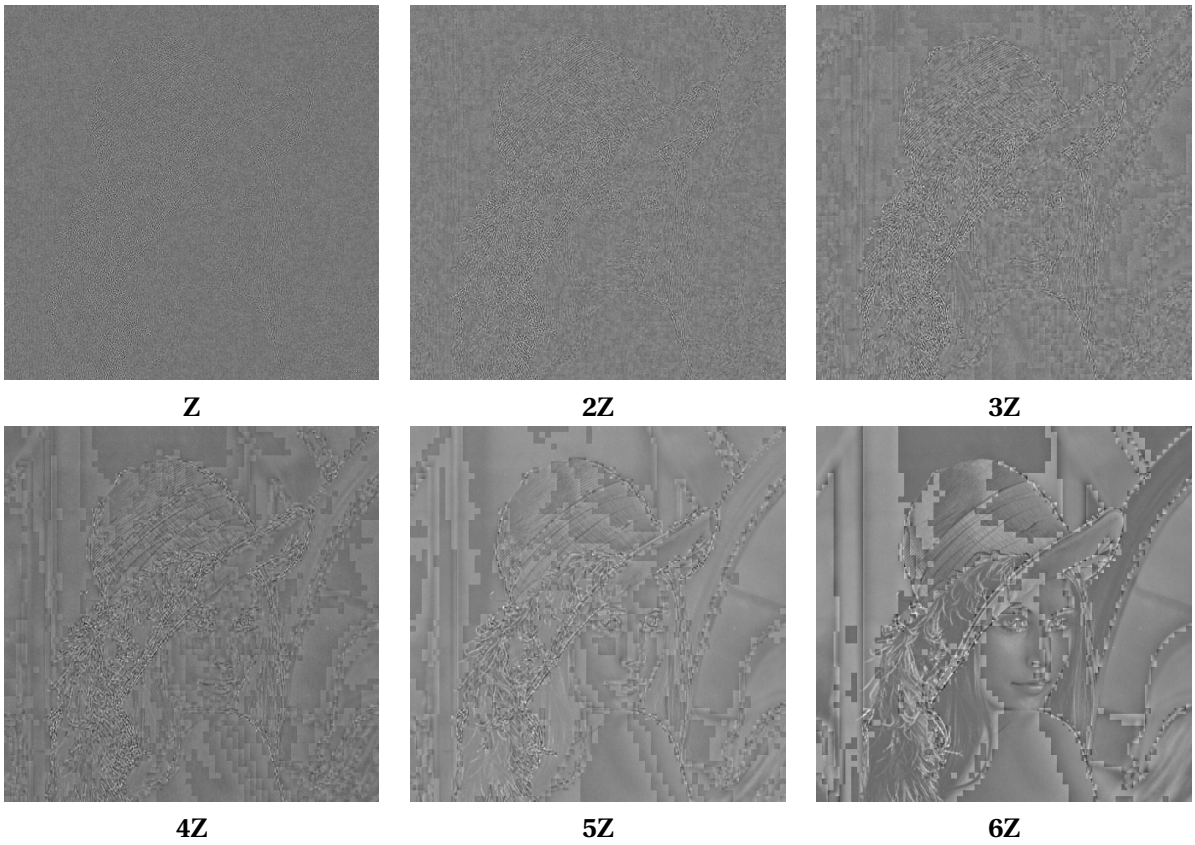
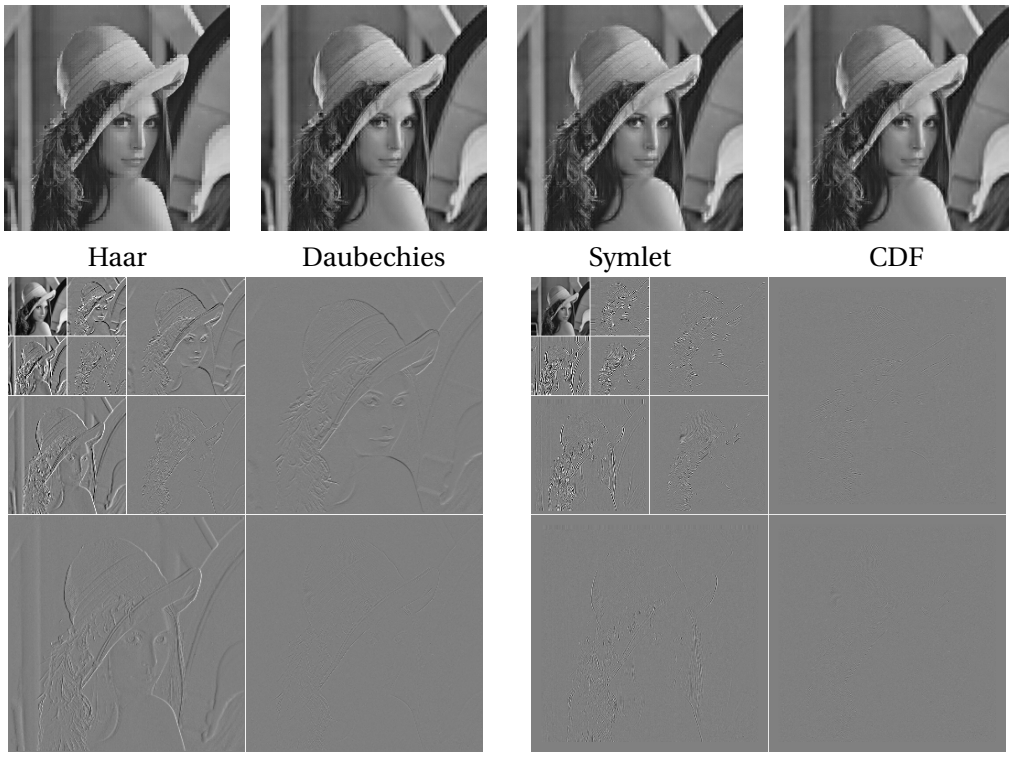


Figure 7.2: Difference images using Threshold mask(with normalization array)

Original image | Reconstructed image | difference image

Figure 7.3: DCT compression using zonal mask



Haar | Daubechies | Symlet | CDF

DWT using Haar | DWT using Daubechies

DWT using Symlet | DWT using Cohen-Daubechies-Feauveau

Figure 7.4: DWT compression with different wavelets

8 MORPHOLIGICAL PROCESSING

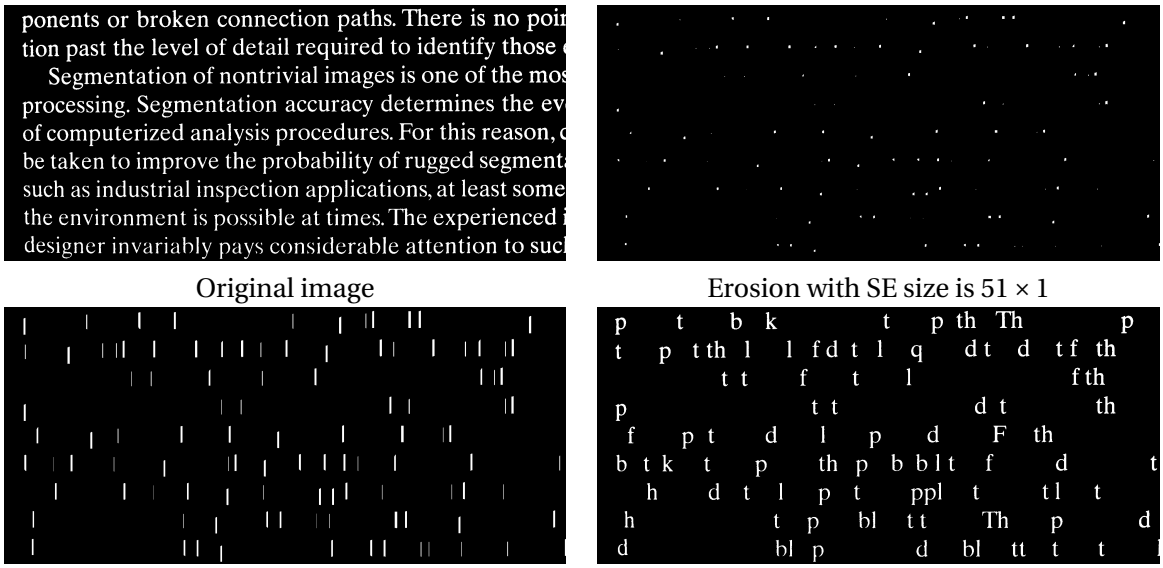
8.1 IMPLEMENTATION

The critical part of this assignment is reconstruction with geodesic dilation. To accelerate the reconstruction process, I implement this reconstruction function as [4] instructs. You can learn about more details in this article.

8.2 RESULT ANALYSIS

After running the script **testdemo.m**, you will get the following result.

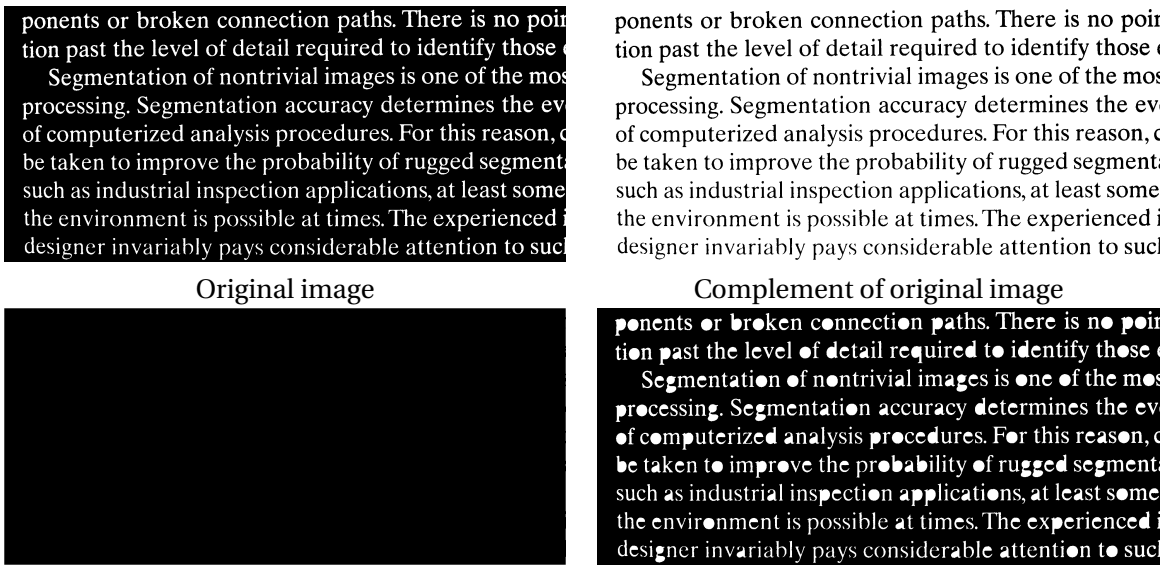
Figure 8.1,8.2 and 8.3 shows the result of opening by reconstruction, filling holes and border clearing. These results are exactly the same as **FIGURE 9.29**,**9.31** and **9.32** in the textbook.



Original image | Erosion with SE size is 51 × 1

Opening of original image with same size SE | Result of opening by reconstruction

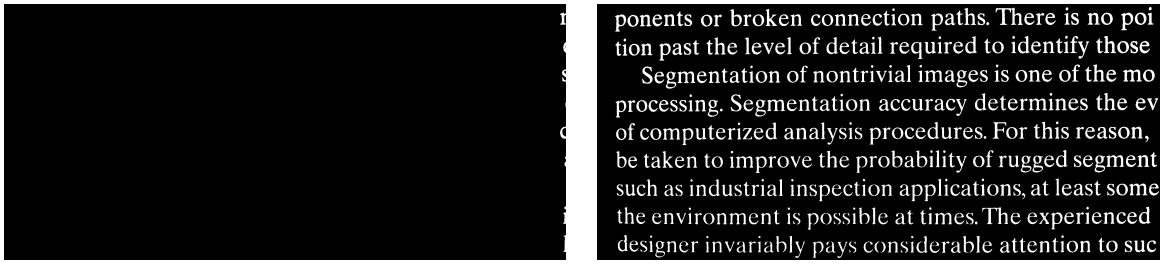
Figure 8.1: Opening by reconstruction



Original image | Complement of original image

Marker image | Result of hole-filling

Figure 8.2: Filling holes



Marker image Result of border clearing

Figure 8.3: Border clearing

9 IMAGE SEGMENTATION

9.1 IMPLEMENTATION

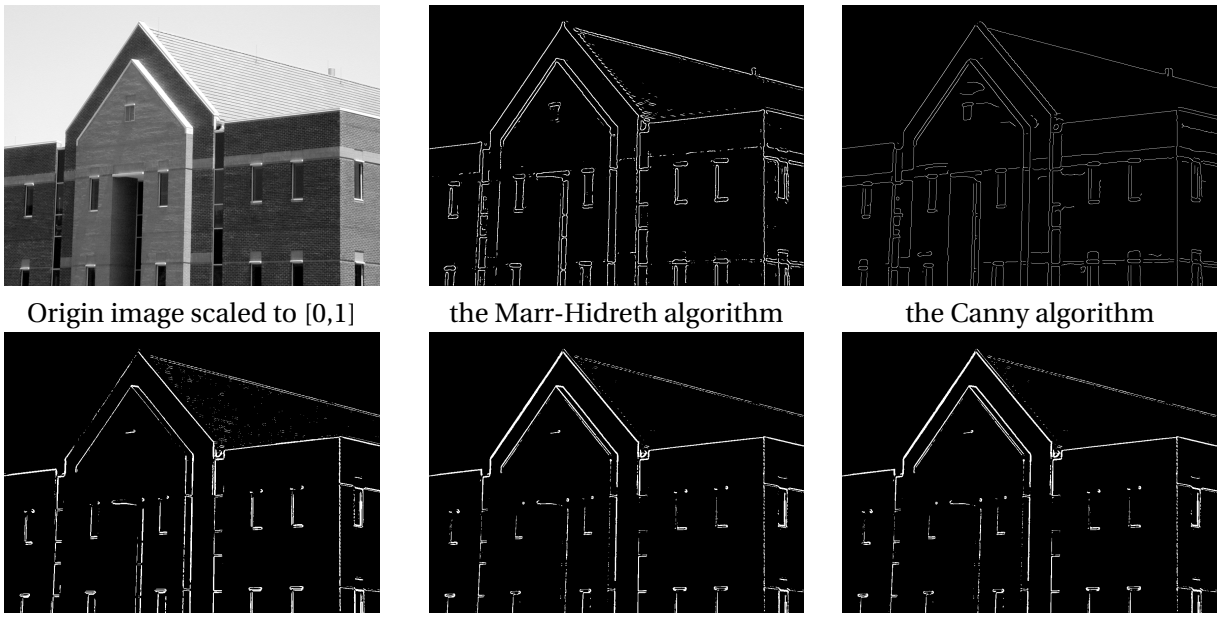
The critical part of the first task in this assignment is to implement Marr-Hildreth edge detector and canny detector. The first one can be directly implemented as our textbook instructs. However, our textbook doesn't introduce clearly about the edge connection process in canny detector. This process is kind of similar to the BFS process. The difference is that we can only choose strong edge pixels as start point. For more details, you can consult the comments in my codes.

The second task thresholding is a easy task. To implement the basic global thresholding, we need to find the point which divide the intensity to two parts with same average intensity. This goal can be achieved by operating the histogram without iteration. The Otsu's way can be directly implemented as our textbook tells.

9.2 RESULT ANALYSIS

After running the script **testdemo.m**, you will get the following result.

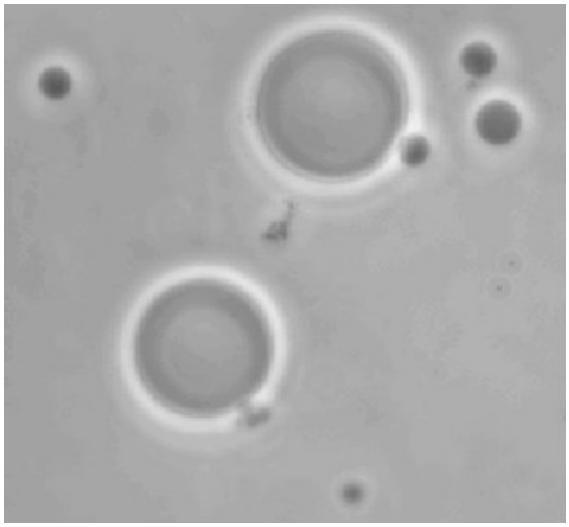
Figure 9.1 shows the comparison between different edge detectors and and 9.2 dipslays the result of two kinds of thresholding. You can compare these two figures with **FIGURE 10.25** and **10.39** in the textbook, the similarity between them will announce the correctness of our results.



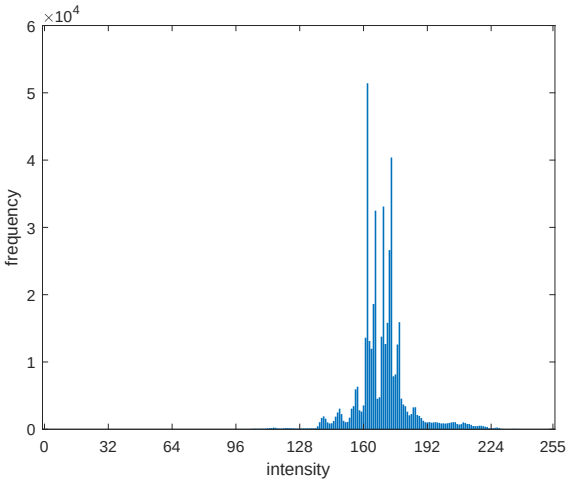
Origin image scaled to [0,1] the Marr-Hidreth algorithm the Canny algorithm

5 × 5 average filter+Roberts mask 5 × 5 average filter+Prewitt mask 5 × 5 average filter+Sobel mask

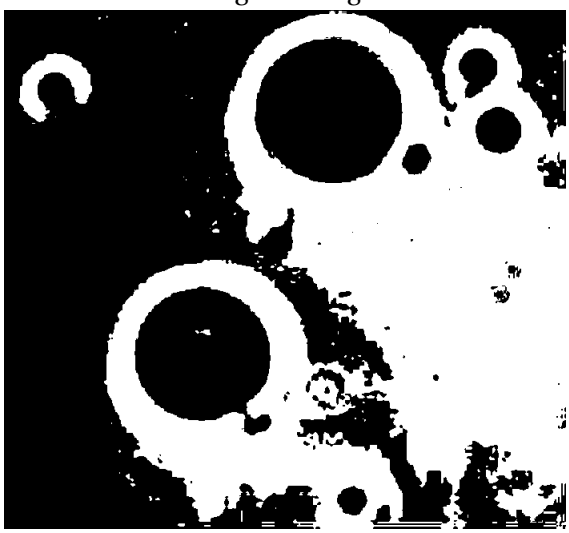
Figure 9.1: Comparison between different edge detectors



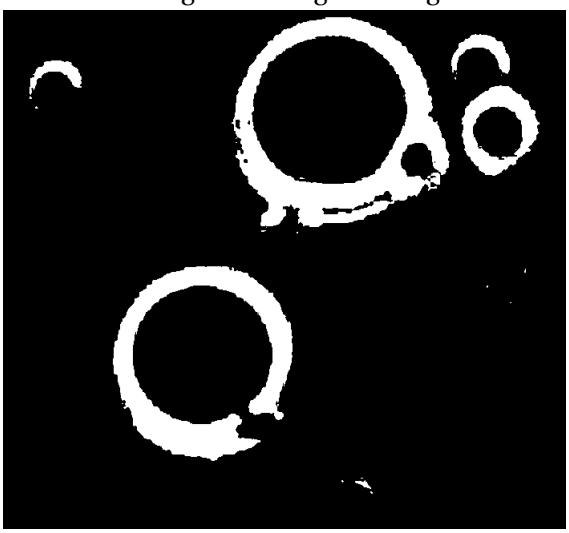
Original image



Histogram of original image



Basic global algorithm($T = 170, \eta = 0.1987$)



Otsu's method($T = 181, \eta = 0.2174$)

Figure 9.2: Thresholding

10 IMAGE REPRESENTATION AND DESCRIPTION

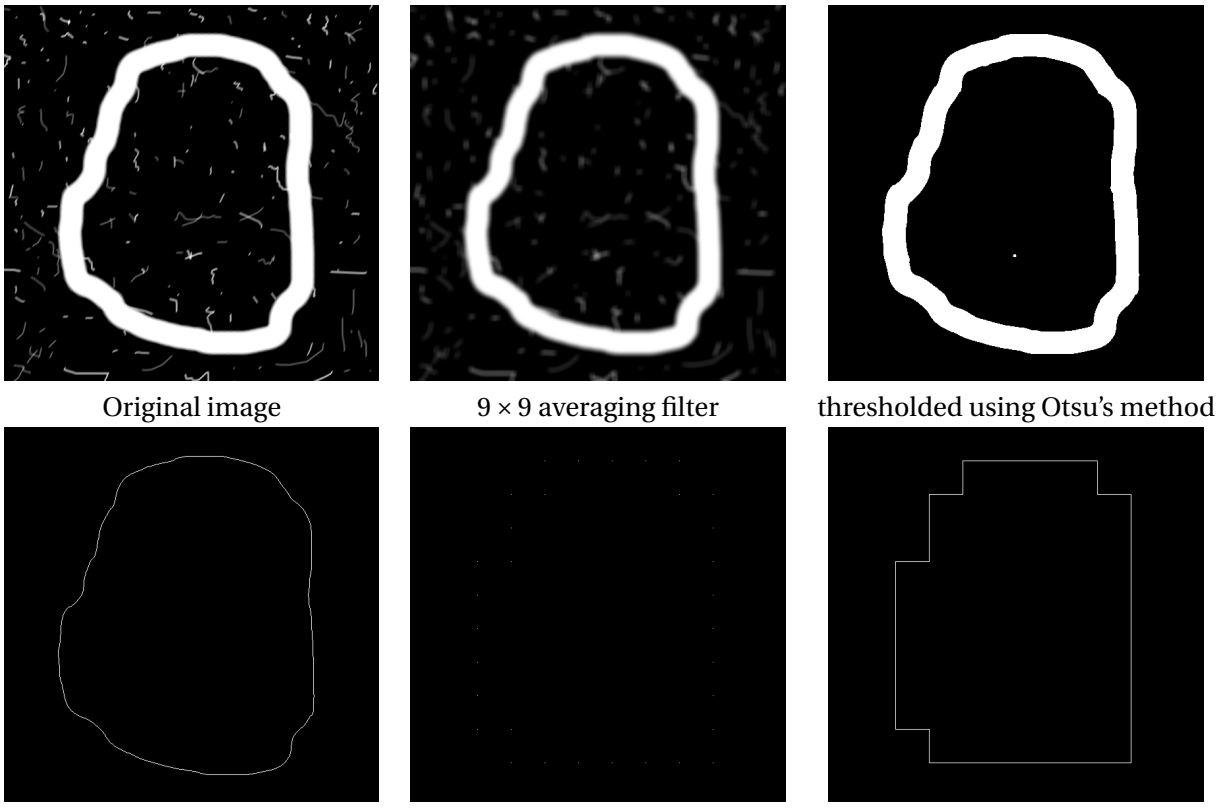
After running the script **testdemo.m**, you will get the following result.

```
For image noisy_stroke.tif:
Default 8-directional chain code start with (2,5):
0 0 0 0 6 0 6 6 6 6 6 6 6 6 4 4 4 4 4 4 2 4 2 2 2 2 2 0 2 2 0 2
and its first difference is:
0 0 0 6 2 6 0 0 0 0 0 0 0 6 0 0 0 0 0 6 2 6 0 0 0 0 6 2 0 6 2 6
Normalized 8-directional chain code starts with (2,5), its value is:
0 0 0 0 6 0 6 6 6 6 6 6 6 6 4 4 4 4 4 4 2 4 2 2 2 2 2 0 2 2 0 2
and its first difference is:
0 0 0 6 2 6 0 0 0 0 0 0 0 6 0 0 0 0 0 6 2 6 0 0 0 0 6 2 0 6 2 6
Eigenvalues of the covariance matrices obtained from all 6 images:
lambda_1=10344.304804
lambda_2=2965.897734
lambda_3=1400.635021
```

```
lambda_4=203.455480
lambda_5=94.277394
lambda_6=31.037280
```

The process of getting Freeman chaincode is shown in Figure 10.1. The chaincode are output as vectors and displayed in the frame above. Figure 10.2,10.3,10.4,10.5 shows the the result of principal components analysis.

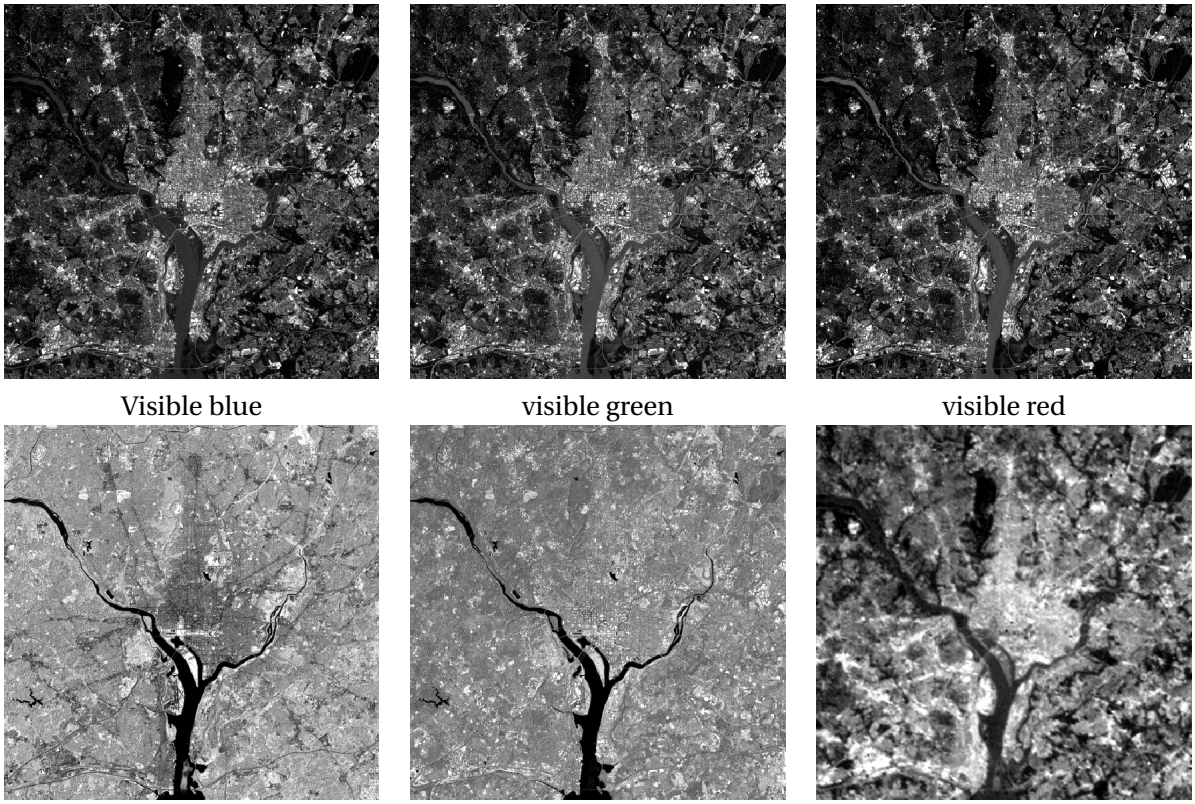
You can compare these figures with **FIGURE 11.5**, **11.38**, **11.39**, **11.41** and **11.42** in the textbook. Most of them are similar to each other. However, images in Figure 10.5 is much differnet from the images in **FIGURE 11.42** in our textbook. This may be caused by the different use of scaling way. But these images are similar to the images in **FIGURE 12.33** in the book *DIP Using Matlab*.



Original image | 9 × 9 averaging filter | thresholded using Otsu's method

Boundary following | Subsample boundary(scale 50) | Connected resampled boundary

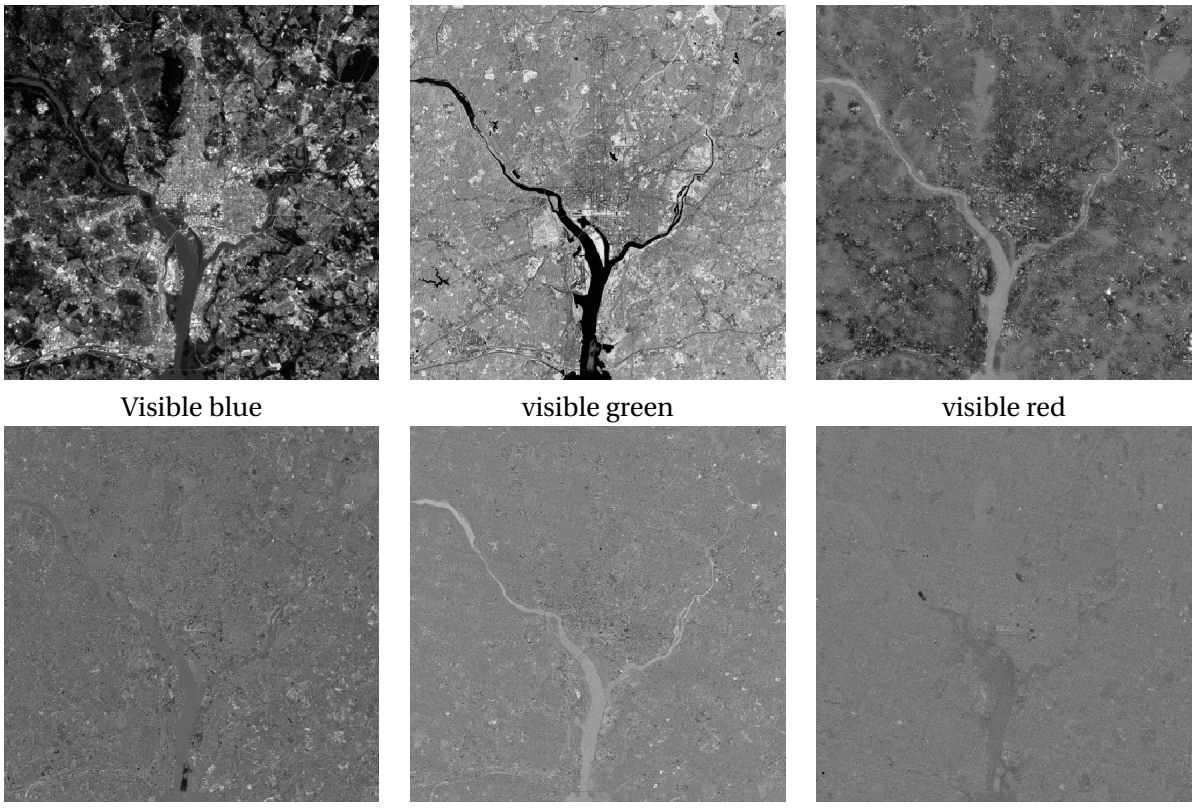
Figure 10.1: Freeman chaincode



Visible blue | visible green | visible red

near infrared | middle infrared | thermal infrared bands

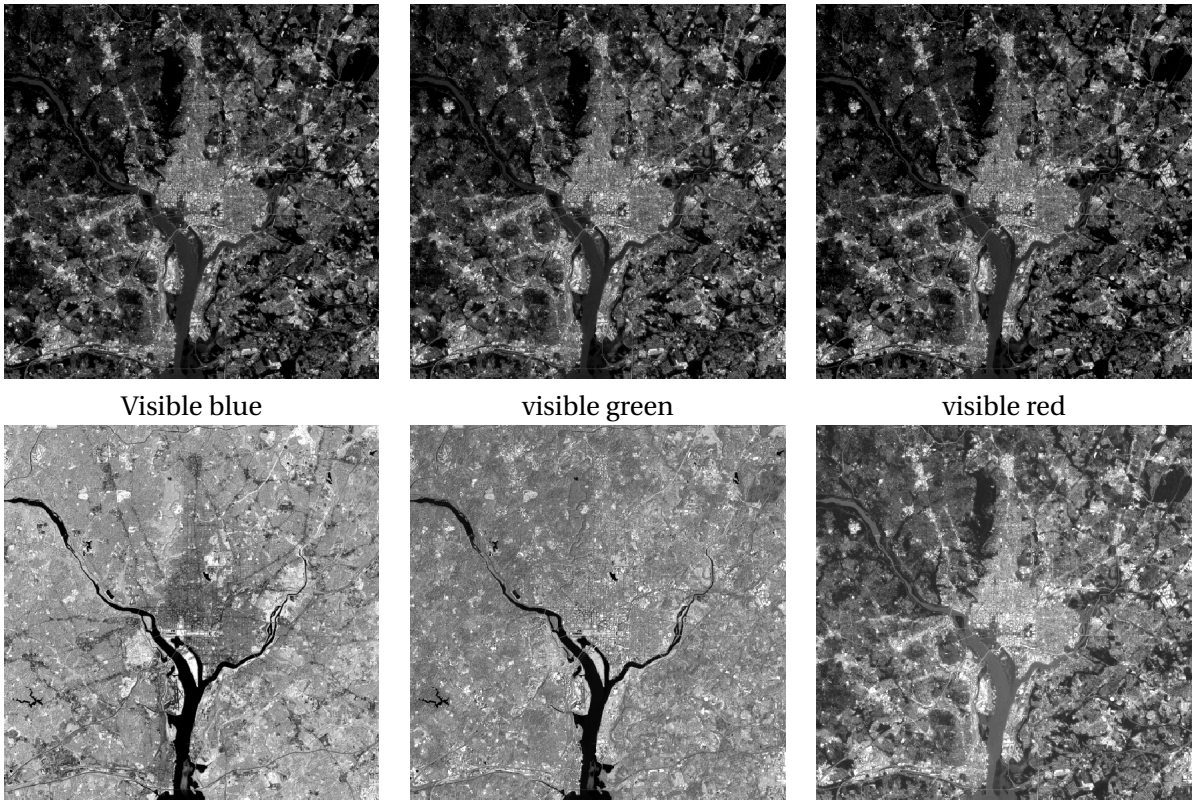
Figure 10.2: Original multispectral images



Visible blue | visible green | visible red

near infrared | middle infrared | thermal infrared bands

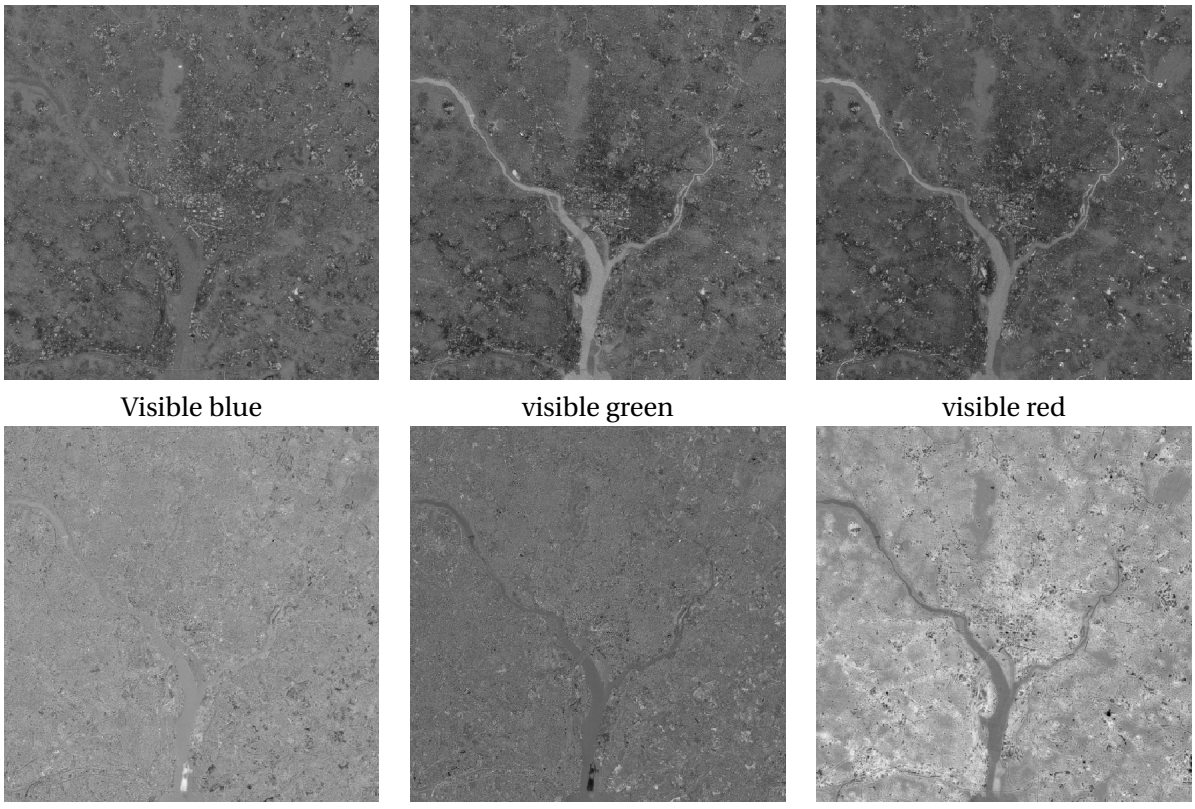
Figure 10.3: Principal component images



Visible blue visible green visible red

near infrared middle infrared thermal infrared bands

Figure 10.4: Images reconstructed using only 2 PCs with largest eightvalues



Visible blue visible green visible red

near infrared middle infrared thermal infrared bands

Figure 10.5: Difference between the original and reconstructed images

REFERENCES

[1] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing (3rd Ed)*, page 1. Prentice Hall, Upper Saddle River, NJ., 2008. 1

[2] 2wb05 simulation lecture 8: Generating random variables. https://www.win.tue.nl/~marko/2WB05/lecture8.pdf. Accessed: 2017-12-31. 12

[3] Image restoration. http://www.ee.columbia.edu/~xlx/ee4830/notes/lec7.pdf. Accessed: 2017-12-31. 14

[4] Luc Vincent. Morphological grayscale reconstruction in image analysis: Applications and efficient algorithms. *IEEE transactions on image processing*, 2(2):176–201, 1993. 21