

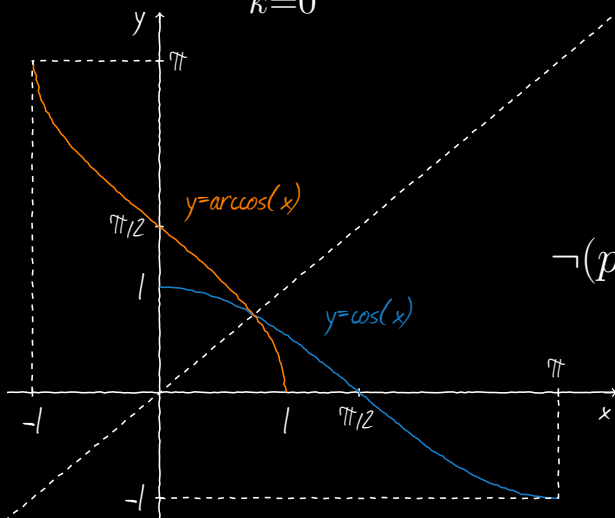
Report for

Digital Image Processing

Fanyong Xue

version 1.1

$$(a + b)^n = \sum_{k=0}^n \binom{n}{k} a^k b^{n-k}$$

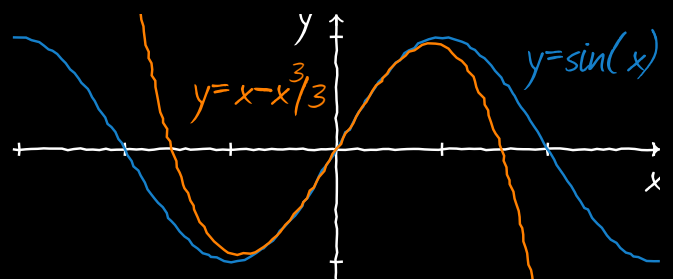


$$\zeta_k = |a|^{1/n} e^{i(\arg(a) + 2k\pi)/n}$$

$$e^{i\pi} + 1 = 0$$

$$\neg(p \vee q) \equiv (\neg p) \wedge (\neg q)$$

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$



Chapter 1 Histogram Equalization Page 4

1.1	OverView	4
1.2	Generate the Histogram	4
	Function – Histogram	
1.3	Transfer Function	5
	Implement the Histogram Equalization Technique – Transfer Function	
1.4	Enhanced Images	7
	Enhanced Function – Enhanced Images	

Chapter 2 Combining spatial enhancement methods Page 9

2.1	OverView	9
2.2	Image b	9
	Laplacian Transform Filter – Laplacian Transform	
2.3	Image c	10
2.4	Image d	11
	Sobel Gradient Masks – Image d	
2.5	Image e	12
2.6	Image f	13
2.7	Image g	14
2.8	Image h	15
	Power-Law Transformation – Image h	

Chapter 3 Filtering in frequency domain Page 16

3.1	OverView	16
3.2	Fourier Transform	16
	Transform Function – Fast Fourier Transform	
3.3	Filtering	16
3.4	Ideal Filter	17
	Ideal Low Pass – Ideal High Pass	
3.5	Butterworth Filter	19
	Butterworth Low Pass – Butterworth High Pass	
3.6	Gaussian Filter	21
	Gaussian Low Pass – Gaussian High Pass	

Chapter 4 Noise and different noise reduction methods Page 23

4.1	OverView	23
-----	----------	----

4.2 Noise23

Uniform Noise – Gaussian Noise – Rayleigh Noise – Erlang(Gamma) Noise – Exponential Noise – Impulse Noise

4.3 Mean Filters29

Filters – Results

4.4 Order-Statistic Filters31

Filters – Results

Chapter 5

Image restoration

Page 34

5.1 OverView34

5.2 This is a section34

This is a subsection – This is a subsection

Appendix A

Codes

Page 35

A.1 Problem135

A.2 Problem 236

A.3 Problem 339

A.4 Problem 446

1.1 OverView

- (a) Write a computer program for computing the histogram of an image.
 - (b) Implement the histogram equalization technique.
 - (c) Your program must be general to allow any gray-level image as its input.
- As a minimum, your report should include the original image, a plot of its histogram, a plot of the transformation function, the enhanced image, and a plot of its histogram.

1.2 Generate the Histogram

1.2.1 Function

Generating the histogram of an image using following function:

$$H(i) = \text{the number of pixel whose value equals to } i \quad (1.1)$$

1.2.2 Histogram

The histogram pictures of Fig1.jpg and Fig2.jpg are listed as follows:

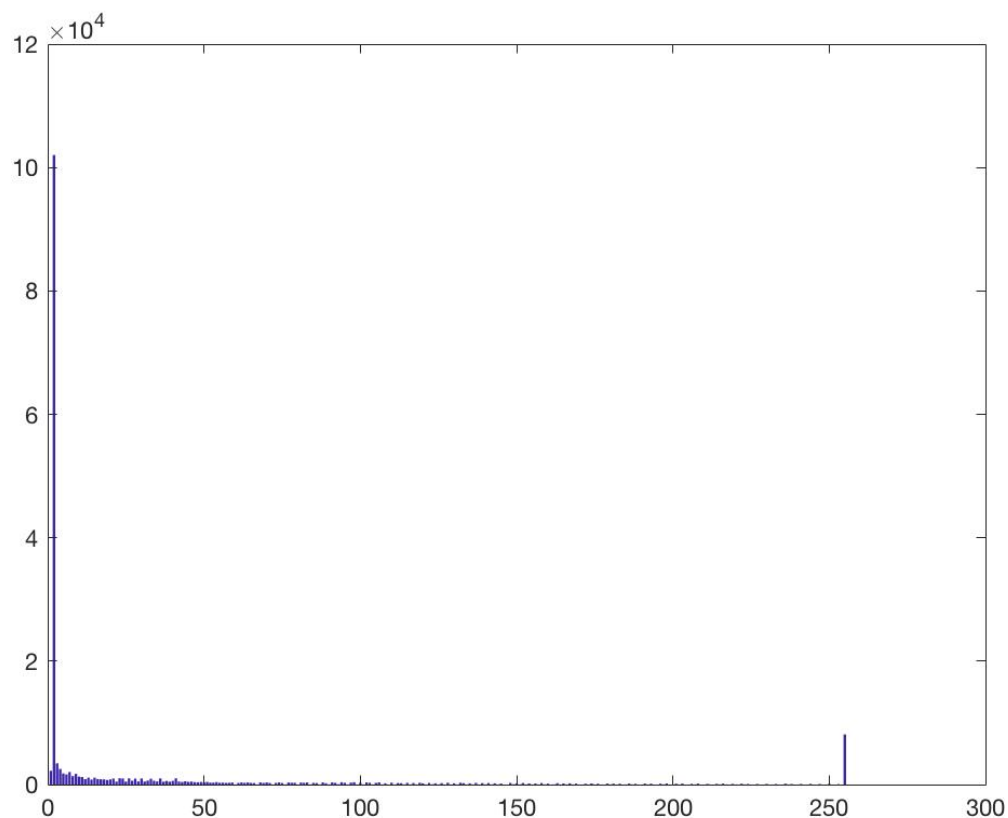


Figure 1.1: Histogram of fig1.jpg

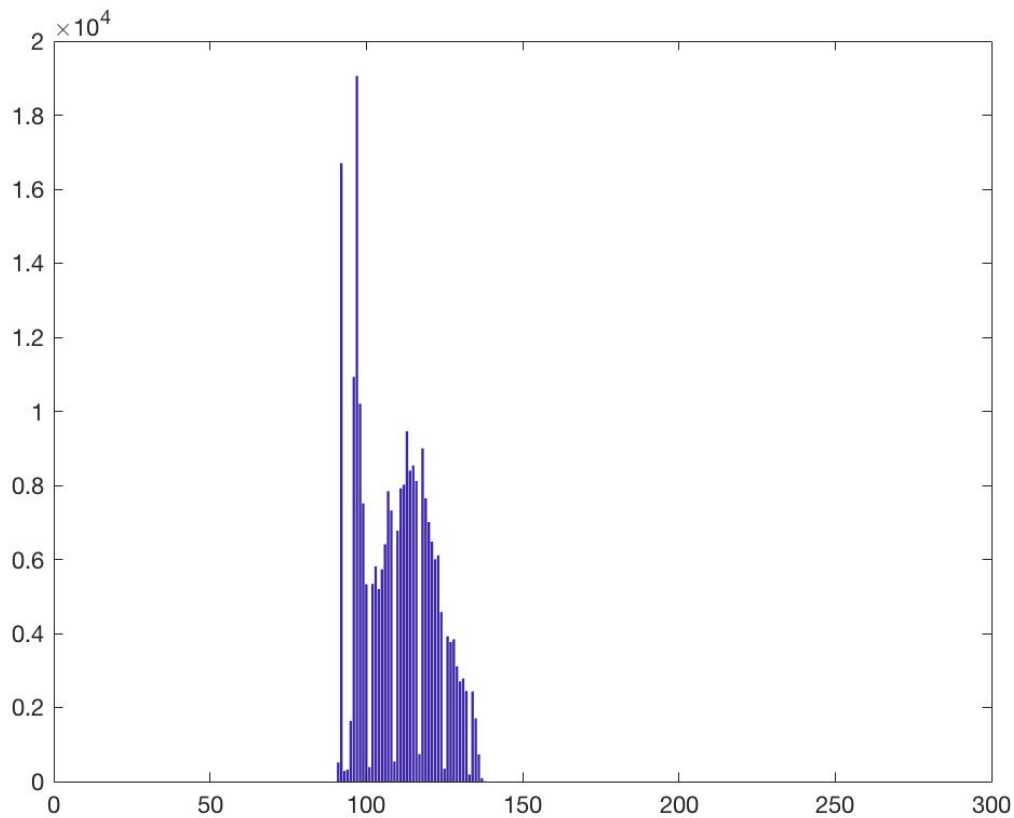


Figure 1.2: Histogram of fig2.jpg

1.3 Transfer Function

1.3.1 Implement the Histogram Equalization Technique

We use those functions to calculate the histogram equalization:

$$L = \text{Max}(\text{image}(r, c)) \quad \forall r \in [1, \text{rows}] \text{ and } \forall c \in [1, \text{cols}] \quad (1.2)$$

$$s(r_k) = L * T(r_k) = L * \sum_{j=0}^k P_r(r_j) = L * \sum_{j=0}^k \frac{n_j}{n} \quad (1.3)$$

1.3.2 Transfer Function

The transfer function of Fig1.jpg and Fig2.jpg are listed as follows:

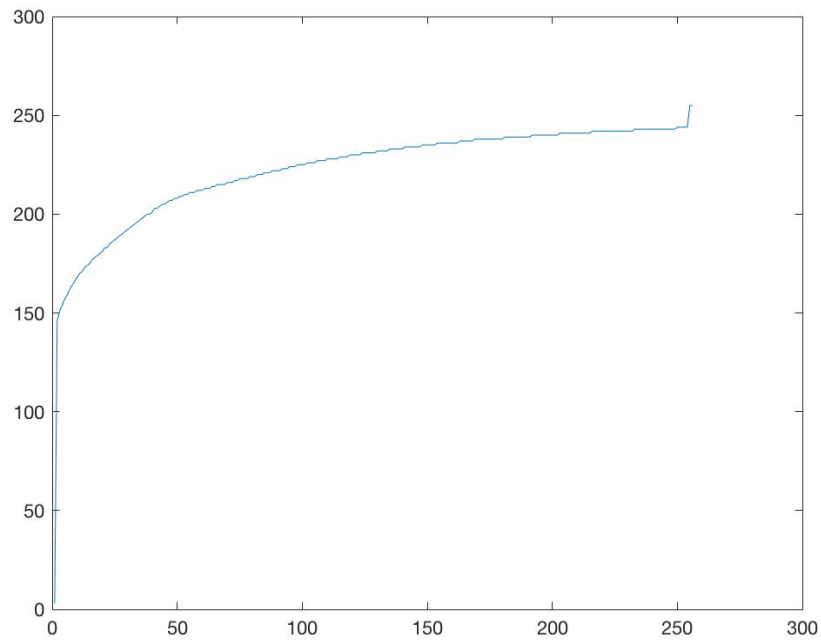


Figure 1.3: Transfer Function of fig1.jpg

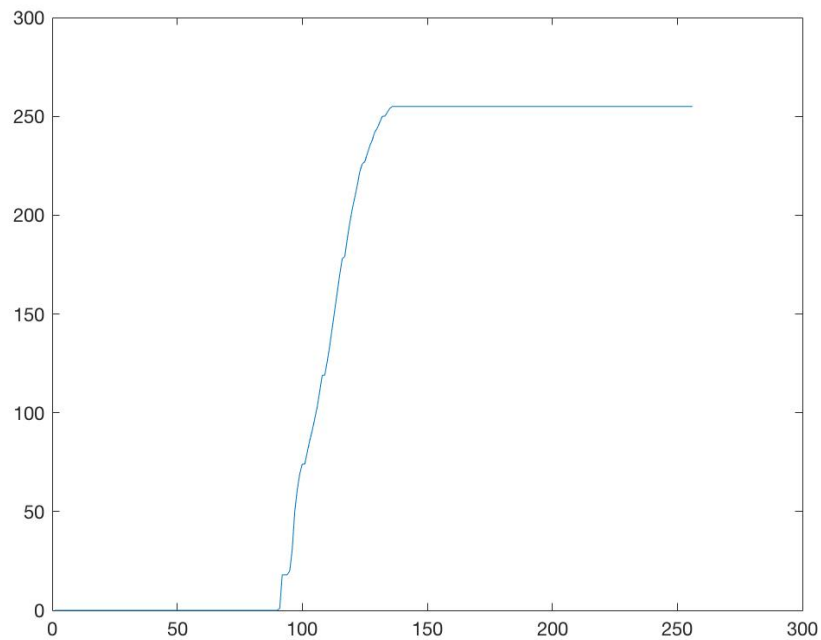


Figure 1.4: Transfer Function of fig2.jpg

1.4 Enhanced Images

1.4.1 Enhanced Function

We use the function

$$New\ Image(r, c) = Transfer\ Function(image(r, c)) \quad \forall r \in [1, rows] \text{ and } \forall c \in [1, cols] \quad (1.4)$$

to enhance the original images.

1.4.2 Enhanced Images

The original images and enhanced images and histogram comparison are listed as follows.

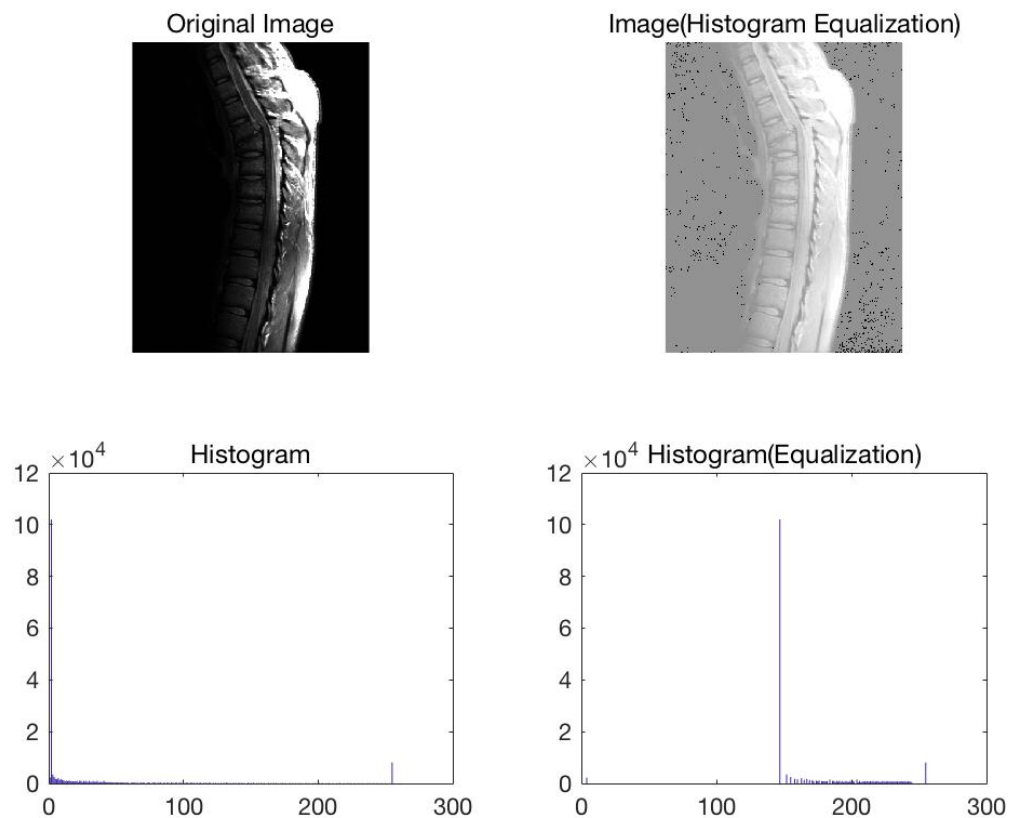


Figure 1.5: original image and enhanced image and histogram comparison of fig1.jpg

Histogram Equalization

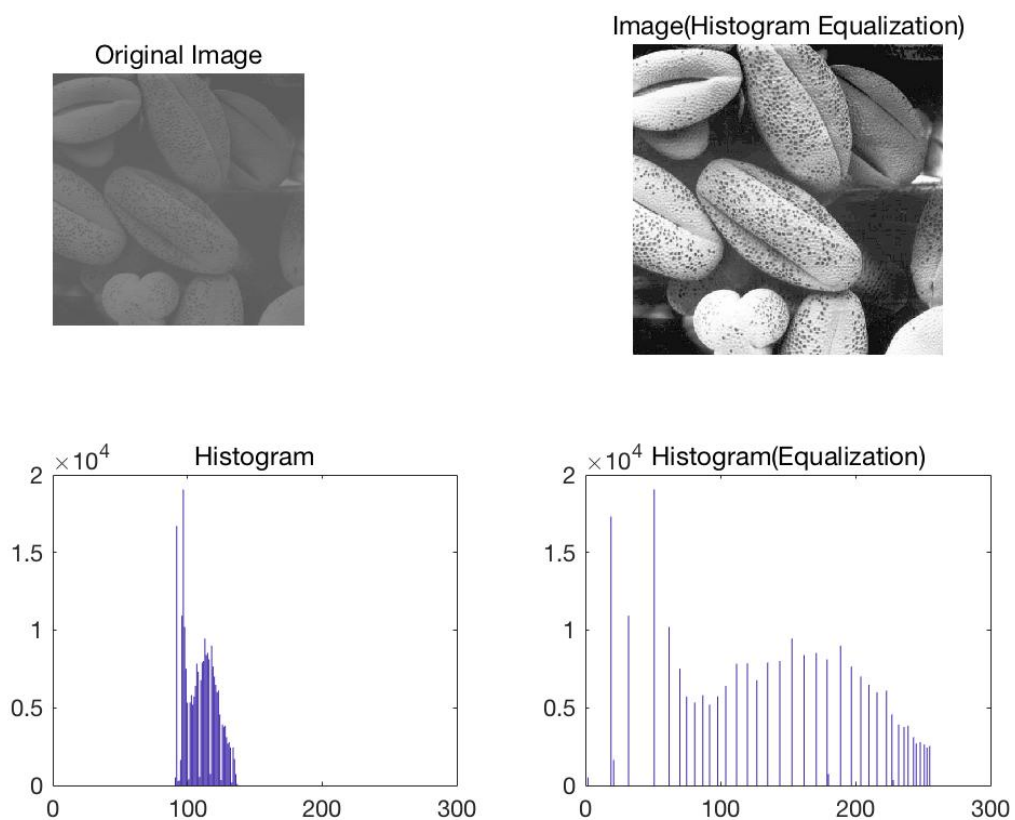


Figure 1.6: original image and enhanced image and histogram comparison of fig2.jpg

2.1 OverView

Implement the image enhancement task of Section 3.7 (Fig 3.43, page 171). The image to be enhanced is `skeleton_orig.tif`. You should implement all steps in Figure 3.43. (You are encouraged to implement all functions by yourself, not to directly use Matlab functions such as `imfilter` or `fspecial`.)

2.2 Image b

2.2.1 Laplacian Transform Filter

We apply Laplacian Transform on the original image to get the image (b) using the following filter.

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

2.2.2 Laplacian Transform

Image b:

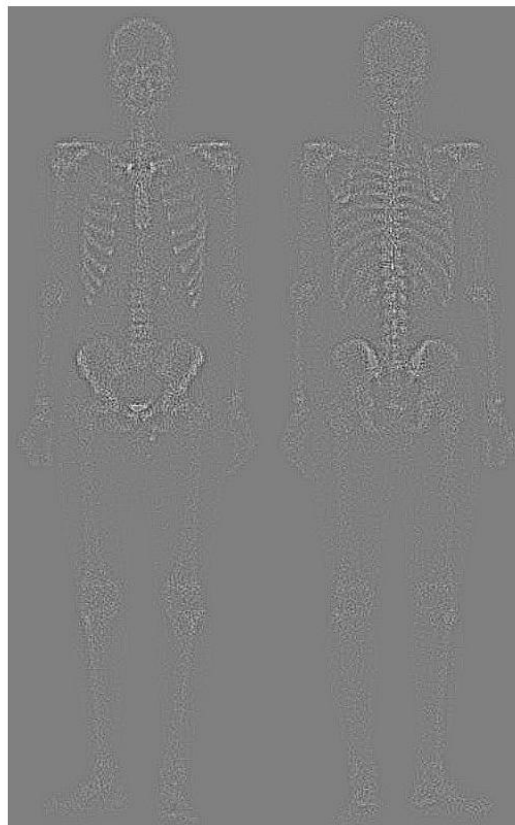


Figure 2.1: Laplacian Transform of `skeleton_orig.tif`

2.3 Image c

When we add the Laplacian of the original image to the original image, we will get the new image c. The new image c is a rather noisy sharpened image. Image c:

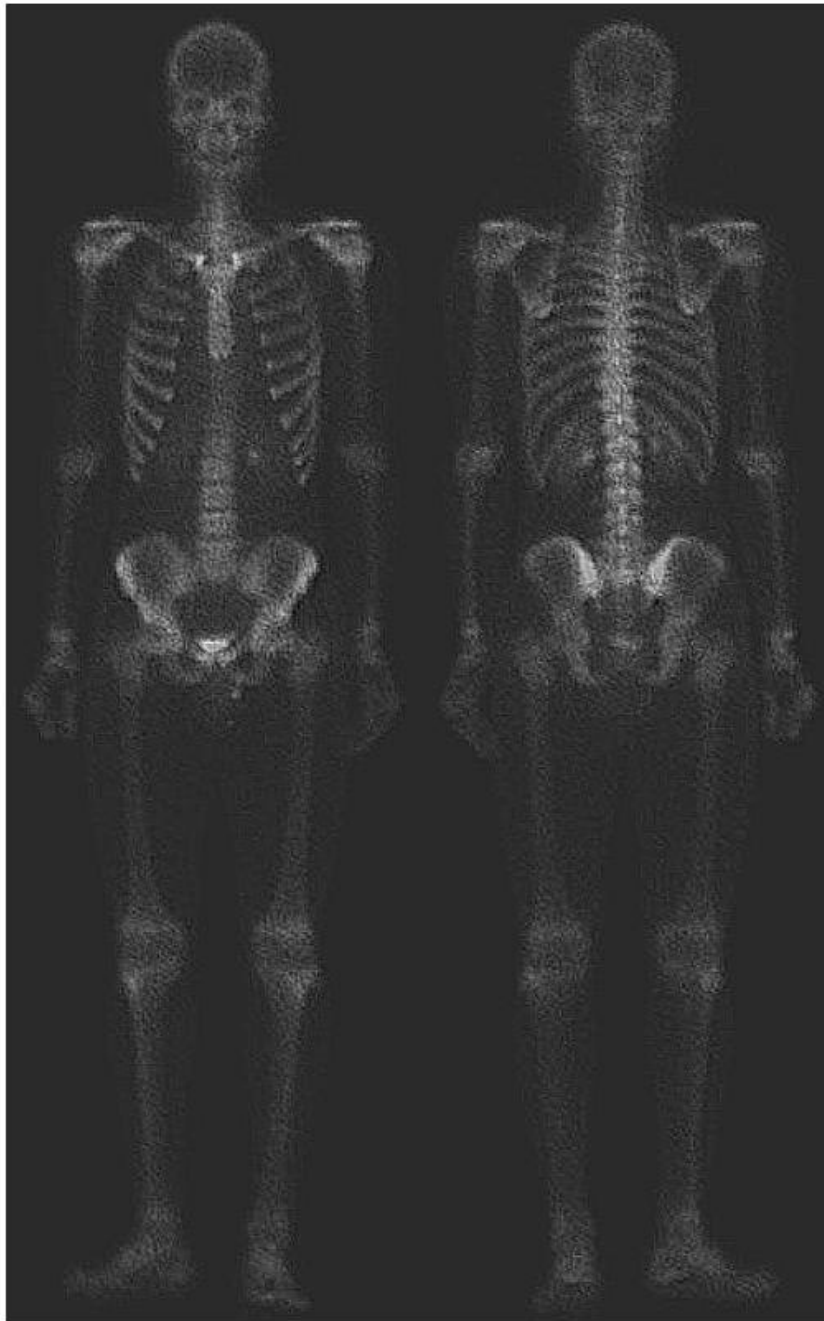


Figure 2.2: Laplacian Transform of skeleton_orig.tif

2.4 Image d

2.4.1 Sobel Gradient Masks

We will use two mask to separately get the components g_x and g_y . Then add the two components together, we will get the the sober gradient of the original image. The new image is as follows. As we can see, edges are much more dominant in this image than in the Laplacian image.

g_x :

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

g_y :

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

2.4.2 Image d

Image d:



Figure 2.3: Laplacian Transform of skeleton_orig.tif

2.5 Image e

Image e is formed by smoothing image d by 5*5 mean filter.
Image e:

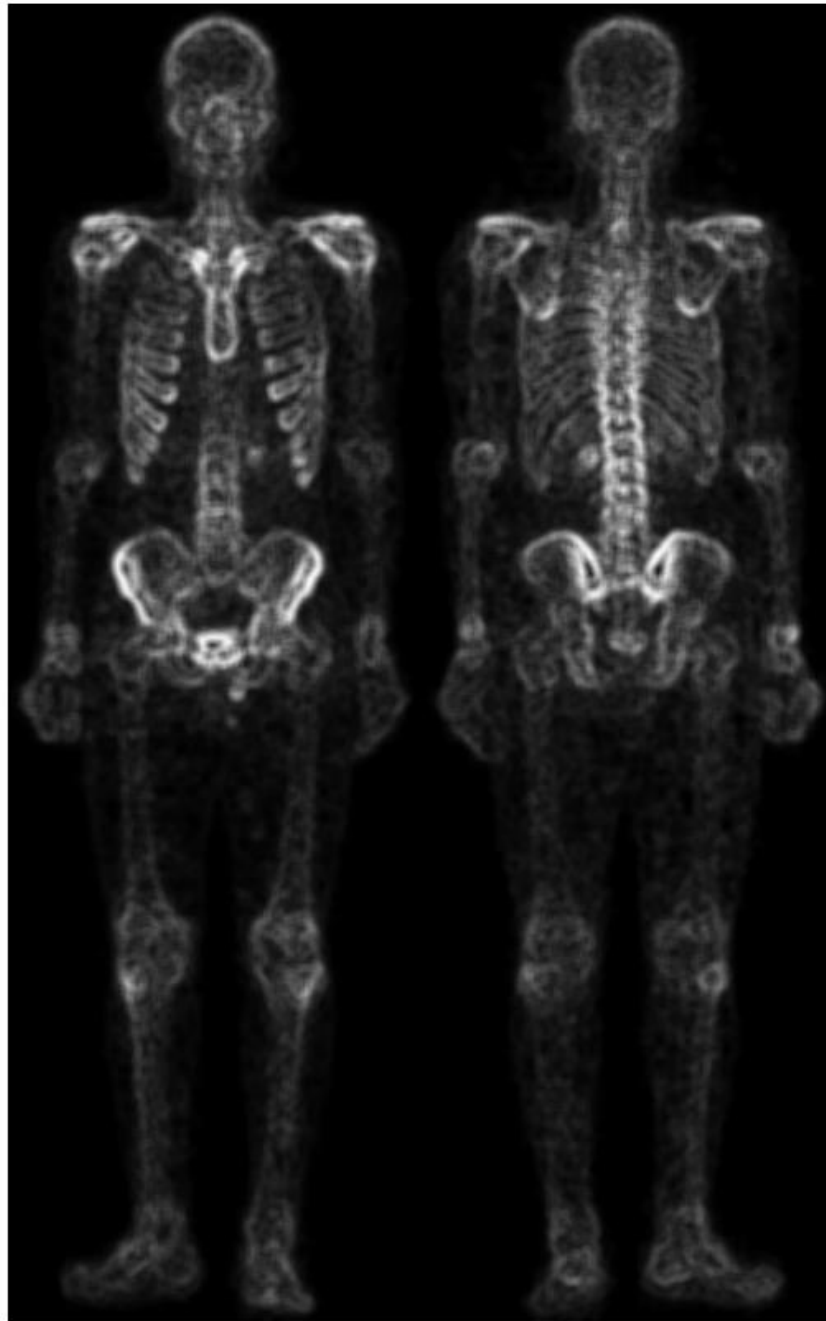


Figure 2.4: Laplacian Transform of skeleton_orig.tif

2.6 Image f

Image f is formed by the product of Laplacian and smoothed-gradient image. The dominance of the strong edges and the relative lack of visible noise, which is the key objective behind masking the Laplacian with a smoothed gradient image.

Image f:

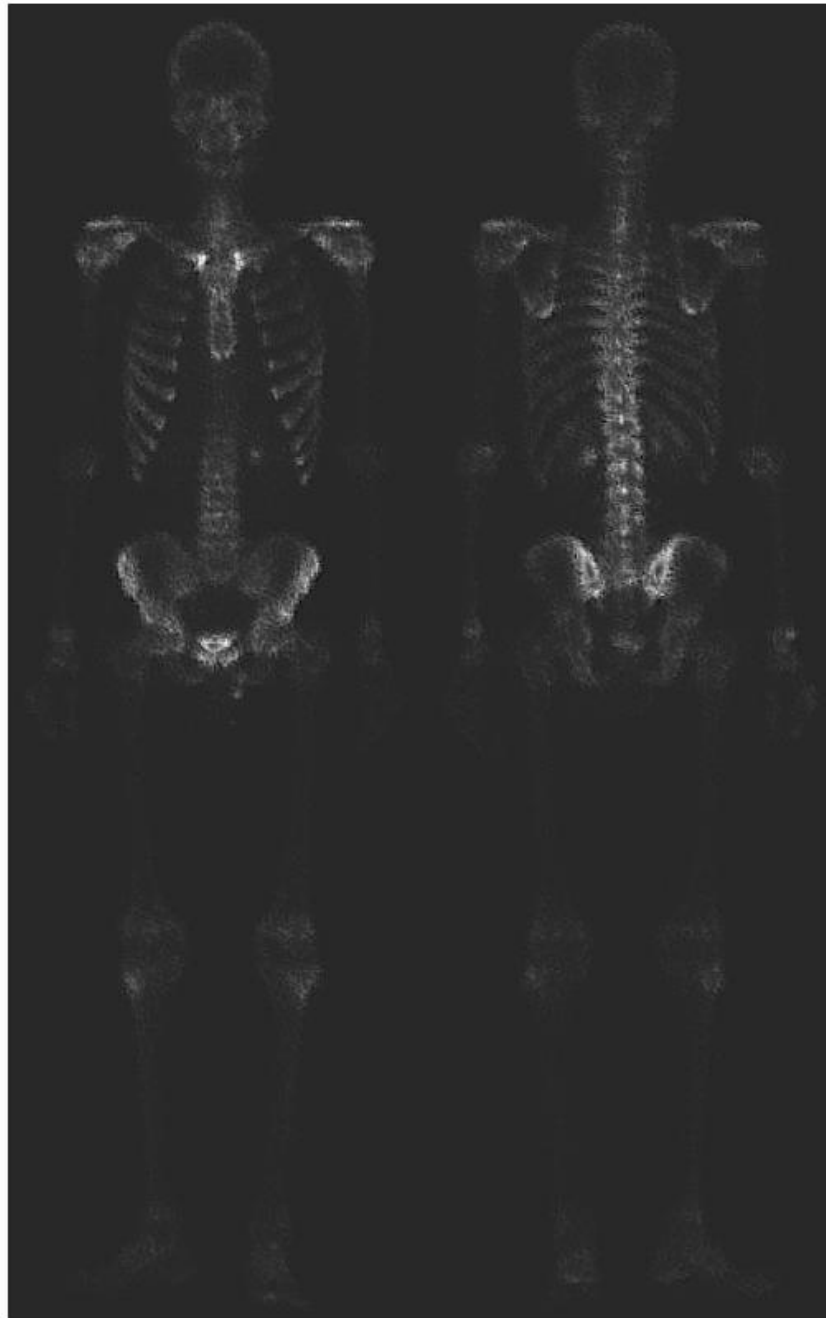


Figure 2.5: Laplacian Transform of skeleton_orig.tif

2.7 Image g

Adding the image f to the original image and then we get image g.
image g:

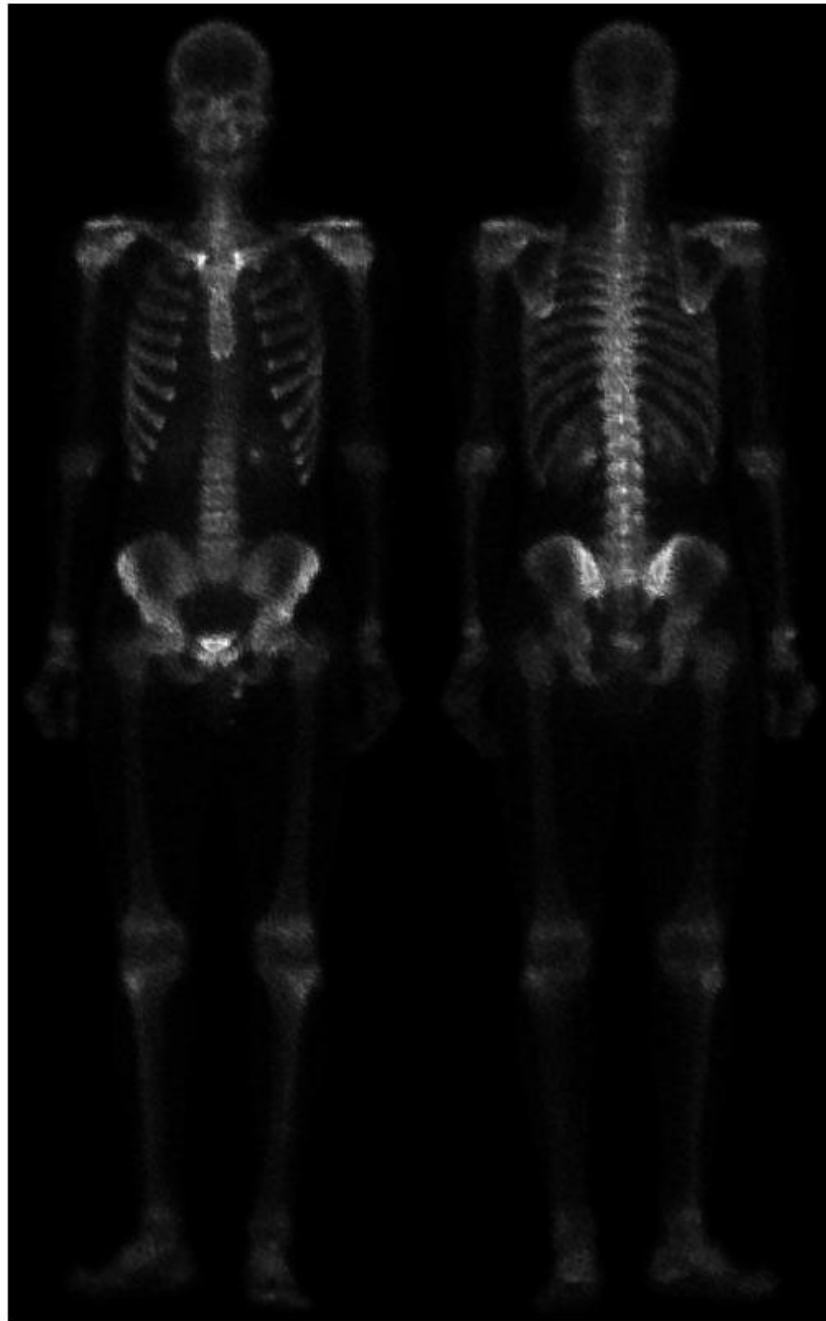


Figure 2.6: Laplacian Transform of skeleton_orig.tif

2.8 Image h

2.8.1 Power-Law Transformation

We use the following function to perform Power-Law Transformation on image g.

$$s = cr^{\gamma} \quad (c = 1 \text{ and } \gamma = 1) \quad (2.1)$$

2.8.2 Image h

Image h:

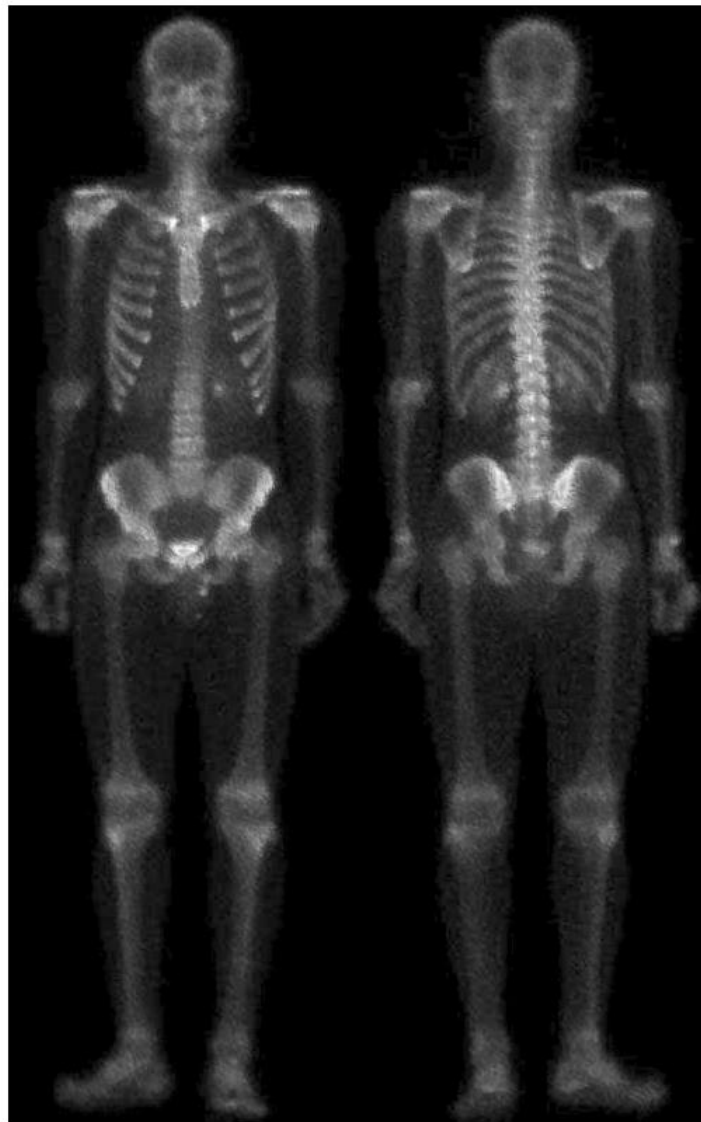


Figure 2.7: Laplacian Transform of skeleton_orig.tif

3.1 OverView

Implement the ideal, Butterworth and Gaussian lowpass and highpass filters and compare the results under different parameters using the image characters_test_pattern.tif (this image file can be found at the ftp server ftp://ftp.cs.sjtu.edu.cn:990/lu-ht/DIP/images) as the test pattern.

3.2 Fourier Transform

3.2.1 Transform Function

We first need transform the image to gray image and then use the 2D Fourier transformation below to change it to the frequency domain.

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

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

3.2.2 Fast Fourier Transform

The inverse function of FFT is as follows:

$$F(u + \frac{M}{2}, v + \frac{N}{2}) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) e^{-j2\pi(\frac{ux+x\frac{M}{2}}{M} + \frac{vy+v\frac{N}{2}}{N})} \quad (3.3)$$

$$= \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) e^{-j2\pi(\frac{ux}{M} + \frac{vy}{N})} * e^{-j\pi x} * e^{-j\pi y} \quad (3.4)$$

$$= \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} (-1)^{x+y} f(x, y) e^{-j2\pi(\frac{ux}{M} + \frac{vy}{N})} \quad (3.5)$$

3.3 Filtering

- Expand image($M * N$) to image a($(2 * M) * (2 * N)$)
- Perform the fourier transform on the new image and get image b
- Shift image b and get image c
- Perform the convolution on image c and filter and get image d
- Shift image d back and get image e
- Perform the inverse fourier transform on image e and get image f
- Cut image f to get the final result image($M * N$);

3.4 Ideal Filter

3.4.1 Ideal Low Pass

Ideal Low Pass Filter

The ideal low pass filter is defined as follows:

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

$$D(u, v) = \sqrt{\left[u - \frac{P}{2}\right]^2 + \left[v - \frac{Q}{2}\right]^2} \quad (3.7)$$

Result

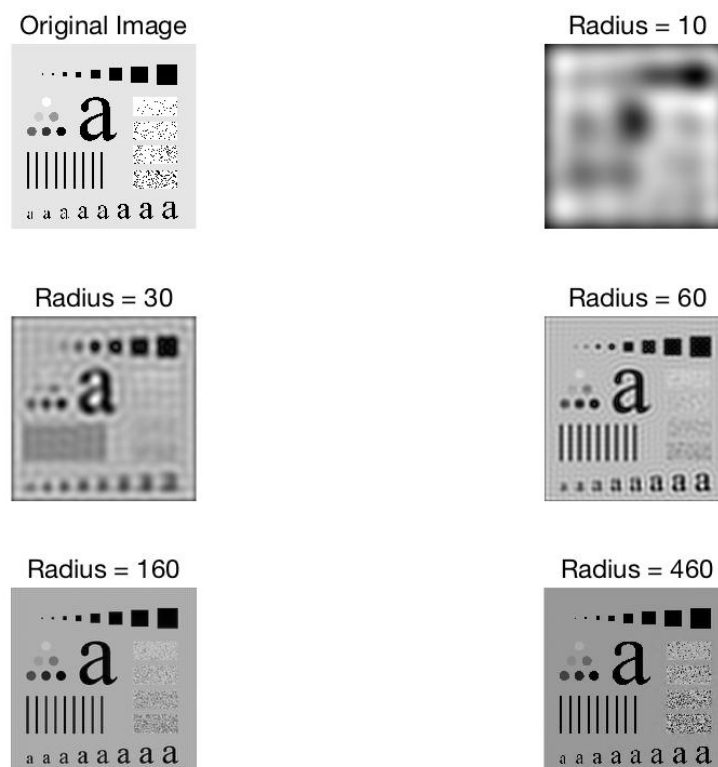


Figure 3.1: ILPF of characters_test_pattern.tif

3.4.2 Ideal High Pass

Ideal High Pass Filter

The ideal high pass filter is defined as follows:

$$H(u, v) = \begin{cases} 0, & D(u, v) \leq 0 \\ 1, & D(u, v) > 0 \end{cases} \quad (3.8)$$

$D(u, v)$ is defined in (3.7)

Result

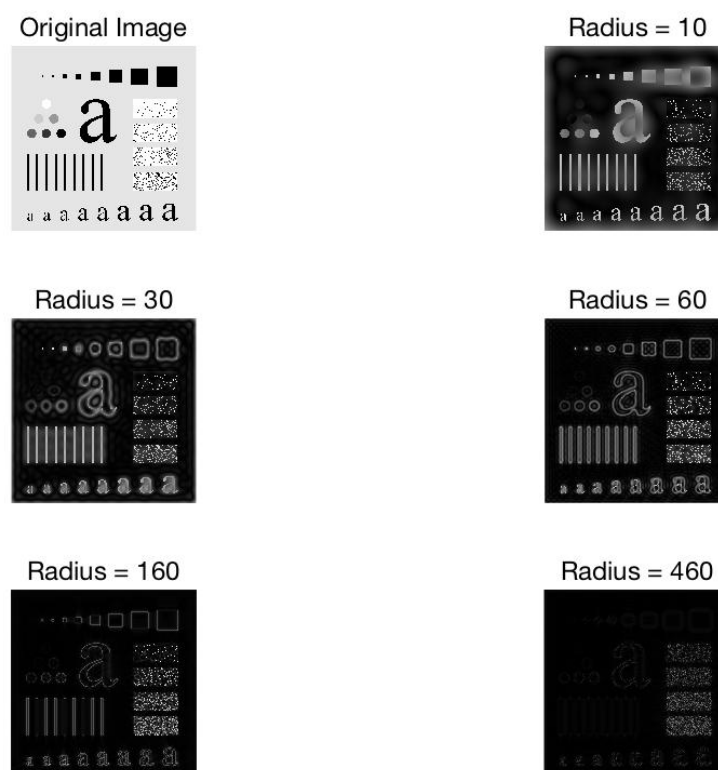


Figure 3.2: ILPF of characters_test_pattern.tif

3.5 Butterworth Filter

3.5.1 Butterworth Low Pass

Butterworth Low Pass Filter

The butterworth low pass filter is defined as follows:

$$H(u, v) = \frac{1}{1 + [D(u, v)/D_0]^{2n}} \quad (3.9)$$

$D(u, v)$ is defined in (3.7)

Result

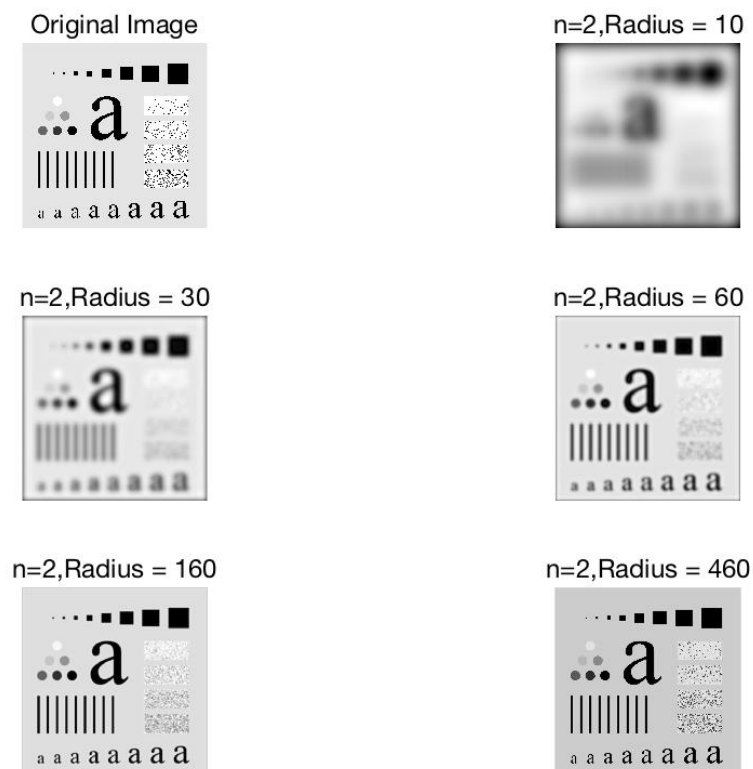


Figure 3.3: BLPF of characters_test_pattern.tif

3.5.2 Butterworth High Pass

Butterworth High Pass Filter

The butterworth high pass filter is defined as follows:

$$H(u, v) = \frac{1}{1 + [D_0/D(u, v)]^{2n}} \quad (3.10)$$

$D(u, v)$ is defined in (3.7)

Result

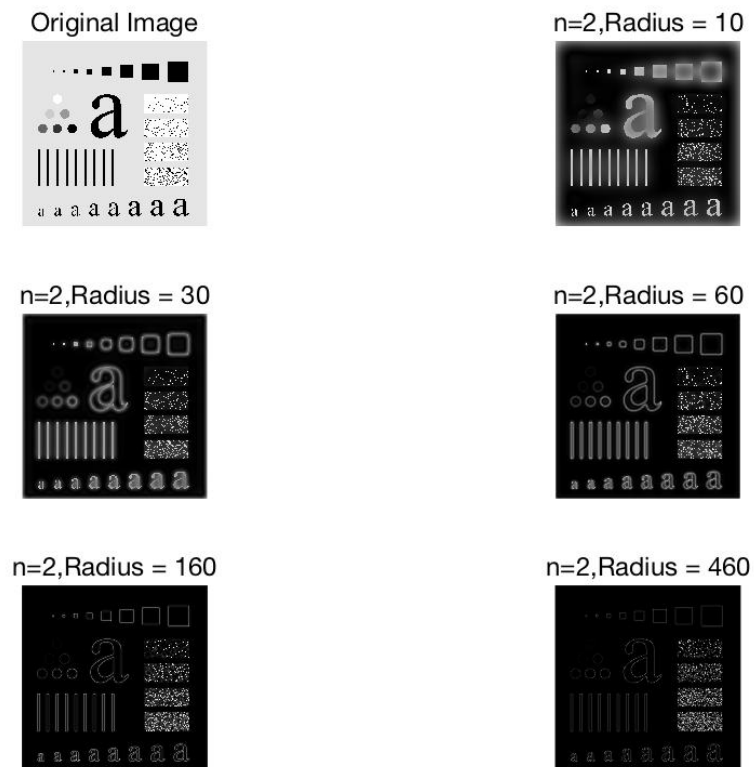


Figure 3.4: BLPF of characters_test_pattern.tif

3.6 Gaussian Filter

3.6.1 Gaussian Low Pass

Gaussian Low Pass Filter

The gaussian low pass filter is defined as follows:

$$H(u, v) = e^{-\frac{D^2(u, v)}{2\sigma^2}} \quad (3.11)$$

$D(u, v)$ is defined in (3.7)

Result

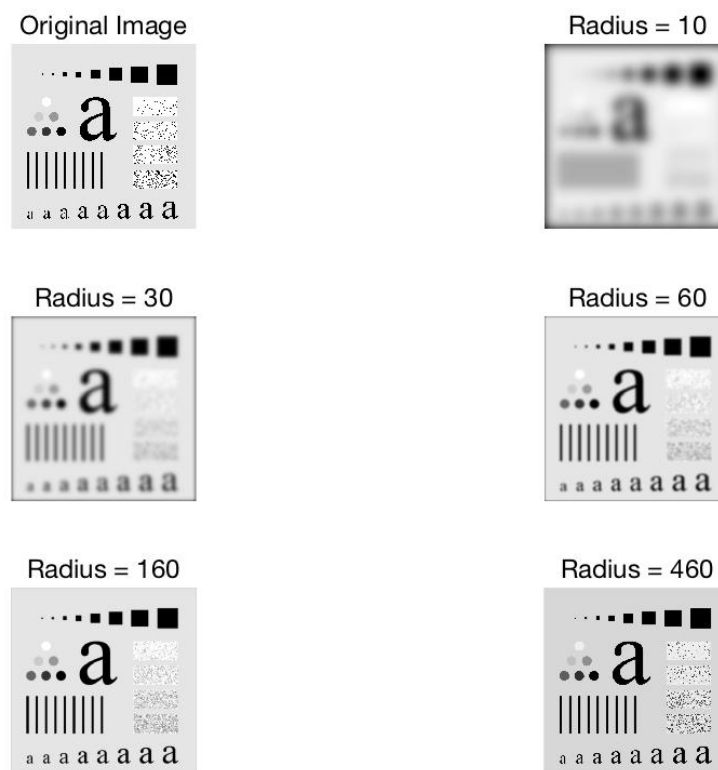


Figure 3.5: GLPF of characters_test_pattern.tif

3.6.2 Gaussian High Pass

Gaussian High Pass Filter

The gaussian high pass filter is defined as follows:

$$H(u, v) = 1 - e^{-\frac{D^2(u, v)}{2\sigma^2}} \quad (3.12)$$

$D(u, v)$ is defined in (3.7)

Result

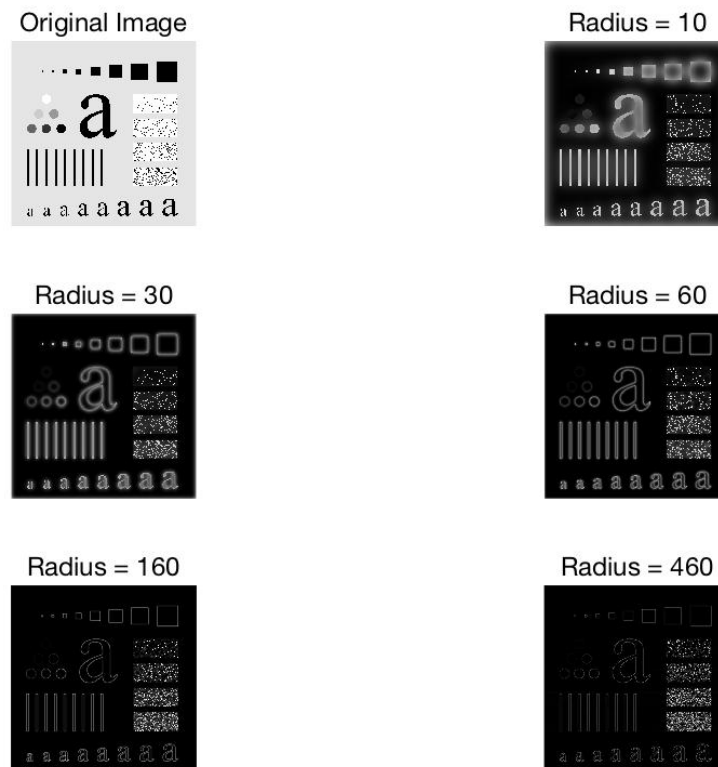


Figure 3.6: GLPF of characters_test_pattern.tif

4.1 OverView

In this problem, you are required to write a program to generate different types of random noise (Uniform, Gaussian, Rayleigh, Gamma, Exponential and Impulse, first started from the uniform noise and then use some functions to convert the uniform noise to Gaussian, Rayleigh, Gamma and Exponential; Impulse noise is generated in a different way, consulting the textbook and some other references) and then add these noises to the test pattern image Fig0503(original_pattern).tif to compare the visual results of the noisy images.

Add some of these noises to the circuit image Circuit.tif (images can be found at <ftp://ftp.cs.sjtu.edu.cn:990/lu-ht/DIP/images>) and investigate the noise reduction results using different mean filters and order statistics filters as the textbook did at pages 344-352 (Pages 322-329 in the electronic version of the textbook).

4.2 Noise

4.2.1 Uniform Noise

The PDF of *uniformnoise* is given by

$$p(z) = \begin{cases} \frac{1}{b-a}, & \text{if } a \leq z \leq b \\ 0, & \text{otherwise} \end{cases} \quad (4.1)$$

The mean of this density function is given by

$$\bar{z} = \frac{a+b}{2} \quad (4.2)$$

and its variance by

$$\sigma^2 = \frac{(b-a)^2}{12} \quad (4.3)$$

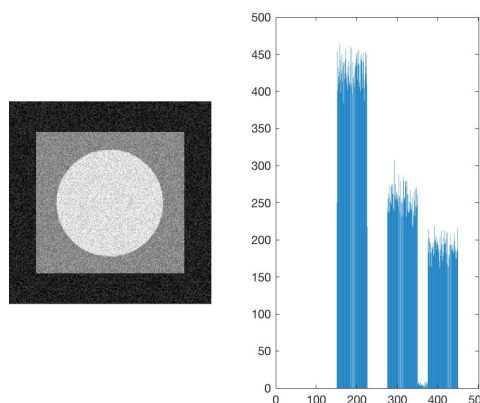


Figure 4.1: Uniform Noise

4.2.2 Gaussian Noise

The PDF of a *Gaussian* random variable, z , is given by

$$p(z) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(z-\bar{z})^2}{2\sigma^2}} \quad (4.4)$$

and it can generate from uniform noise by

$$X = \sqrt{-2\ln U} \cos(2\pi V) \quad (4.5)$$

$$Y = \sqrt{-2\ln U} \sin(2\pi V) \quad (4.6)$$

U and V come from $\text{uniform}(0,1)$.

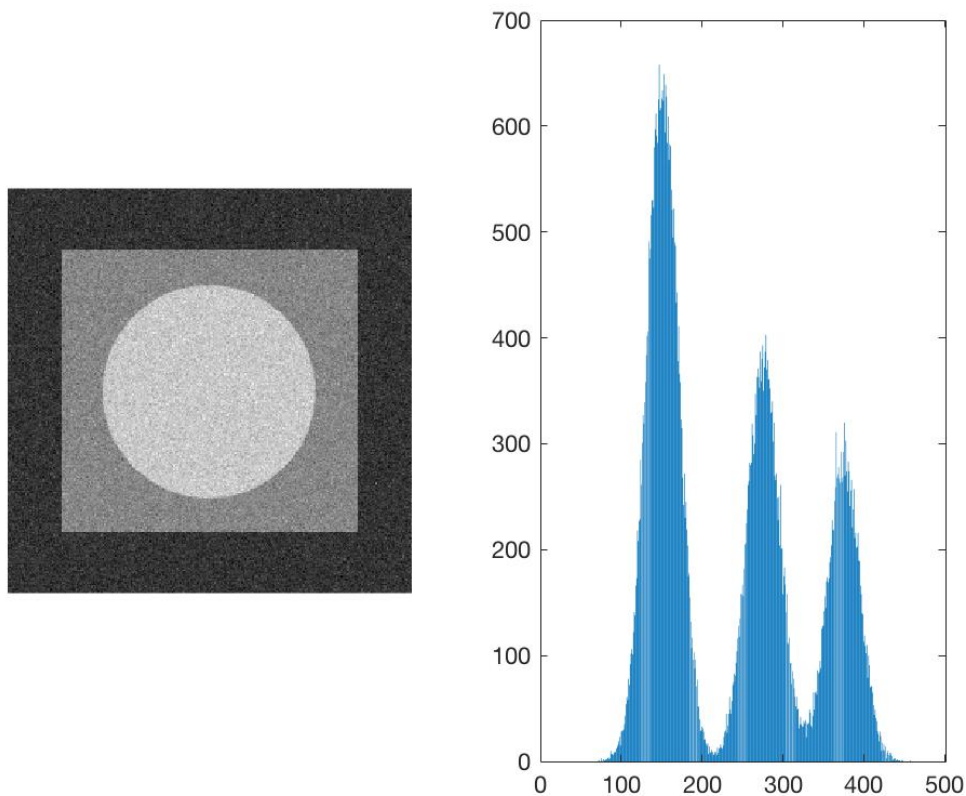


Figure 4.2: Gaussian Noise

4.2.3 Rayleigh Noise

The PDF of *Rayleighnoise* is given by

$$p(z) = \begin{cases} \frac{2}{b}(z-a)e^{-\frac{(z-a)^2}{b}}, & \text{for } z \geq a \\ 0, & \text{for } z < a \end{cases} \quad (4.7)$$

The mean and variance of this density are given by

$$\bar{z} = a + \sqrt{\frac{b\pi}{4}} \quad (4.8)$$

and

$$\sigma^2 = \frac{b(4-\pi)}{4} \quad (4.9)$$

and it can generate from uniform noise by

$$z = a + \sqrt{-b \ln[1 - U(0,1)]} \quad (4.10)$$

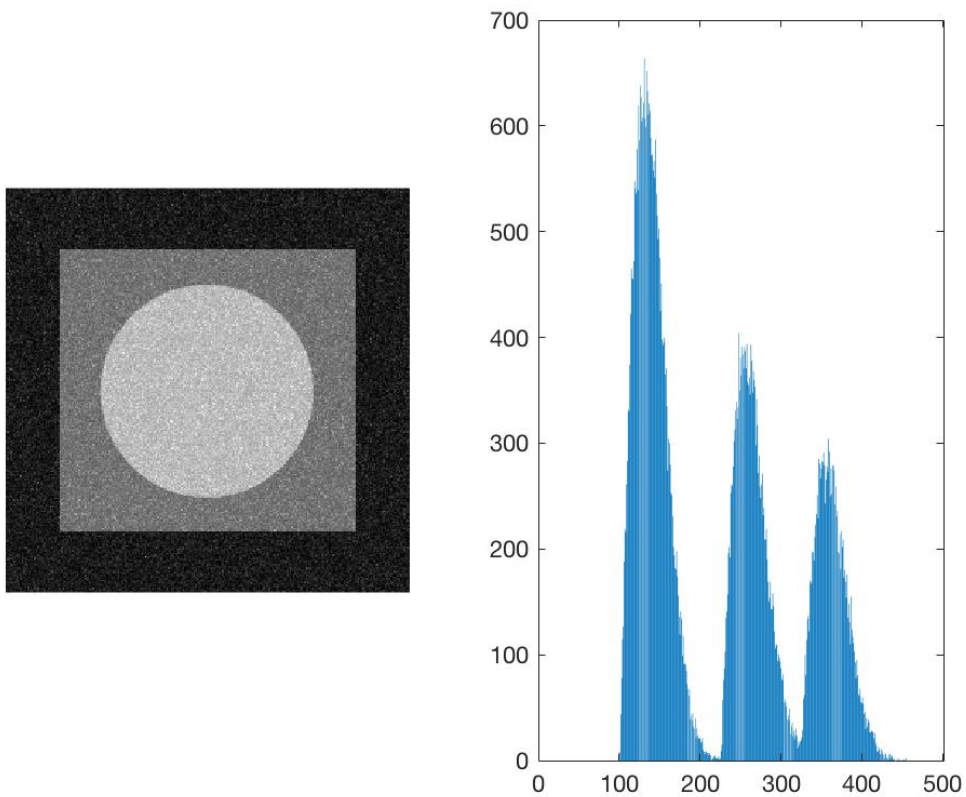


Figure 4.3: Rayleigh Noise

4.2.4 Erlang(Gamma) Noise

The PDF of *Erlangnoise* is given by

$$p(z) = \begin{cases} \frac{a^b z^{b-1}}{(b-1)!} e^{-az}, & \text{for } z \geq 0 \\ 0, & \text{for } z < 0 \end{cases} \quad (4.11)$$

The mean and variance of this density are given by

$$\bar{z} = \frac{b}{a} \quad (4.12)$$

and

$$\sigma^2 = \frac{b}{a^2} \quad (4.13)$$

and it can generate form uniform noise by

$$E_i = -\frac{1}{a} \ln[1 - U(0, 1)] \quad (4.14)$$

$$z = E_1 + E_2 + \dots + E_b \quad (4.15)$$

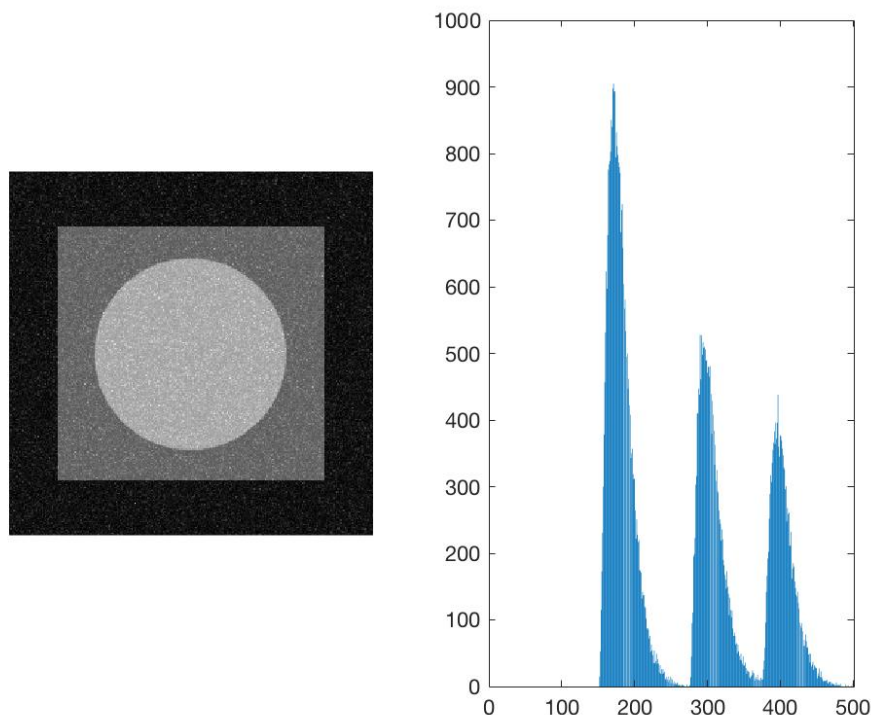


Figure 4.4: Gamma Noise

4.2.5 Exponential Noise

The PDF of *exponentialnoise* is given by

$$p(z) = \begin{cases} ae^{-az}, & \text{for } z \geq 0 \\ 0, & \text{for } z < 0 \end{cases} \quad (4.16)$$

The mean and variance of this density function are

$$\bar{z} = \frac{1}{a} \quad (4.17)$$

and

$$\sigma^2 = \frac{1}{a^2} \quad (4.18)$$

and it can generate from uniform noise by

$$z = -\frac{1}{a} \ln[1 - U(0, 1)] \quad (4.19)$$

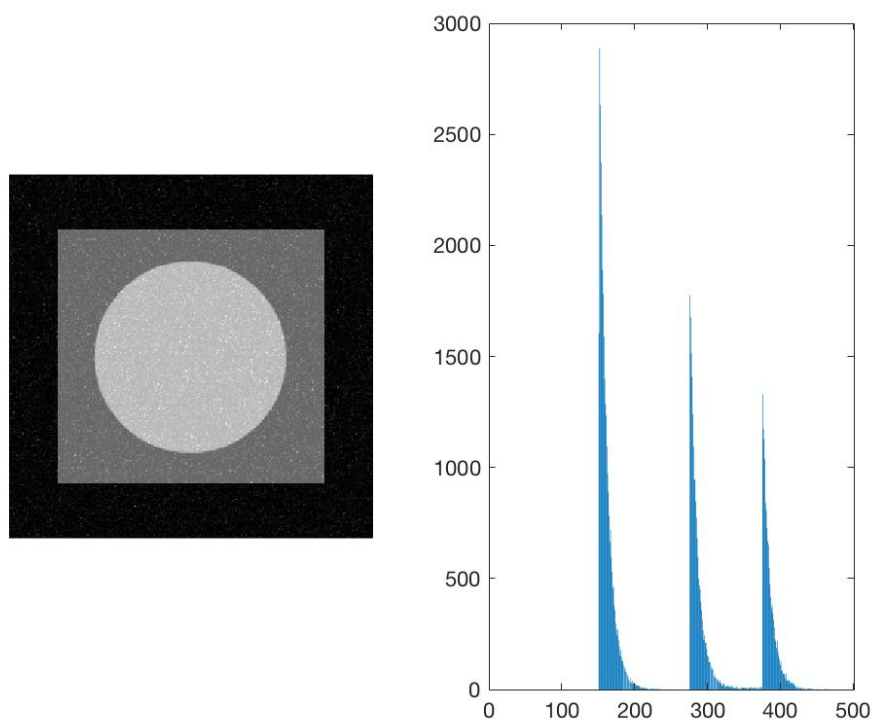


Figure 4.5: Exponential Noise

4.2.6 Impulse Noise

The PDF of *(bipolar)impulsenoise* is given by

$$p(z) = \begin{cases} P_a, & \text{for } z = a \\ P_b, & \text{for } z = b \\ 0, & \text{otherwise} \end{cases} \quad (4.20)$$

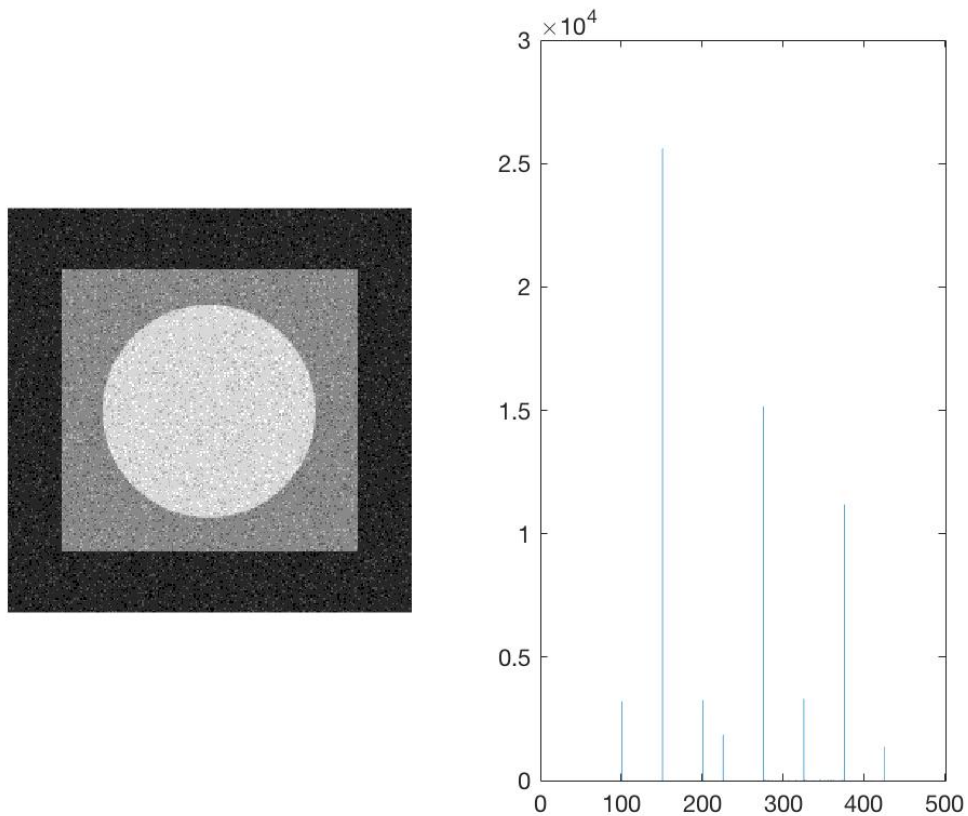


Figure 4.6: Impulse Noise

4.3 Mean Filters

4.3.1 Filters

Arithmetic mean filter

$$\hat{f}(x, y) = \frac{1}{mn} \sum_{(s,t) \in S_{xy}} g(s, t) \quad (4.21)$$

Geometric mean filter

$$\hat{f}(x, y) = \left[\prod_{(s,t) \in S_{xy}} g(s, t) \right]^{\frac{1}{mn}} \quad (4.22)$$

Harmonic mean filter

$$\hat{f}(x, y) = \frac{mn}{\sum_{(s,t) \in S_{xy}} \frac{1}{g(s, t)}} \quad (4.23)$$

Contraharmonic mean filter

$$\hat{f}(x, y) = \frac{\sum_{(s,t) \in S_{xy}} g(s, t)^{Q+1}}{\sum_{(s,t) \in S_{xy}} g(s, t)^Q} \quad (4.24)$$

4.3.2 Results

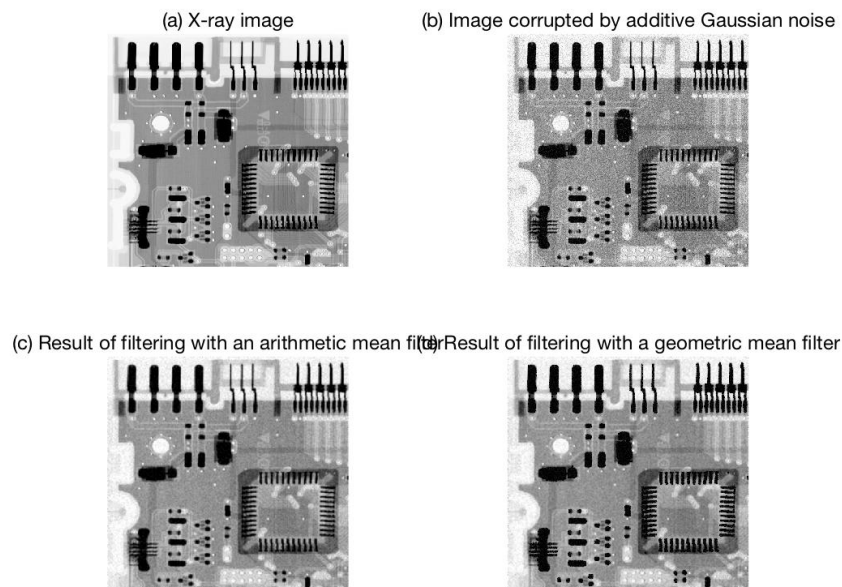


Figure 4.7: Original Book FIGURE 5.7

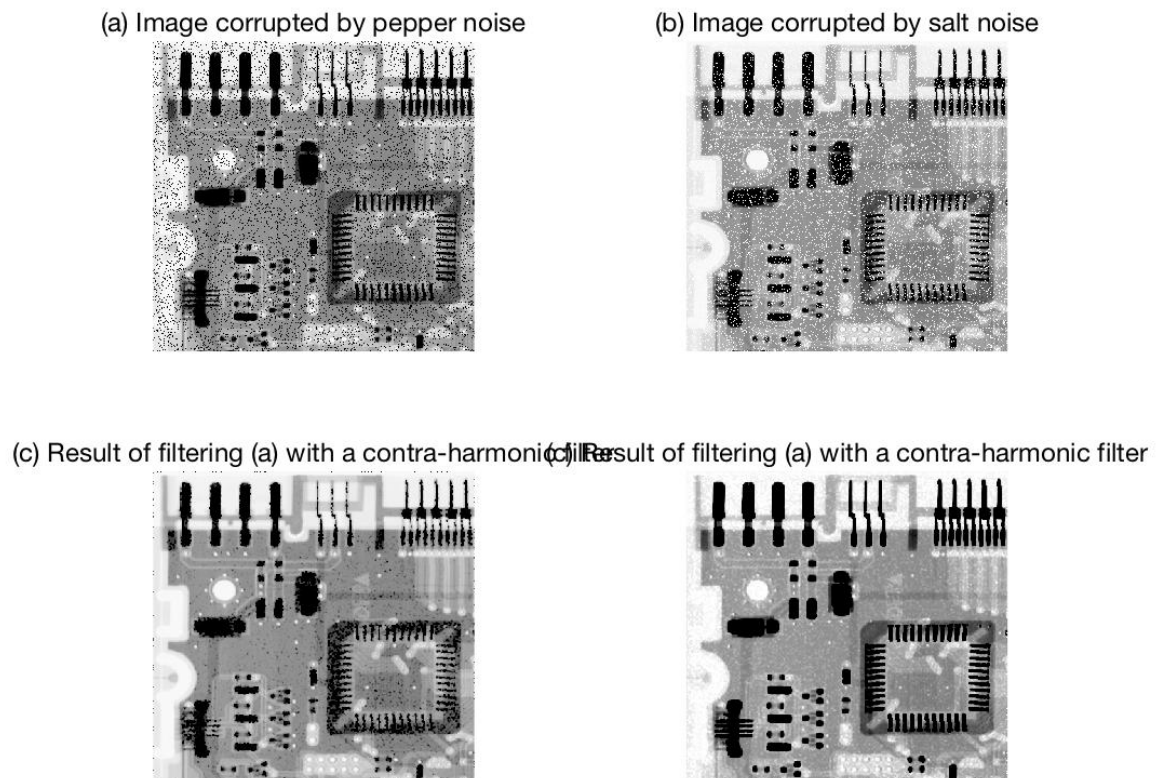


Figure 4.8: Original Book FIGURE 5.8

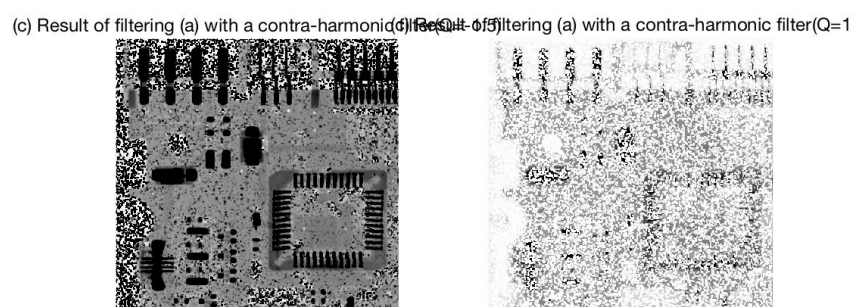


Figure 4.9: Original Book FIGURE 5.9

4.4 Order-Statistic Filters

4.4.1 Filters

Median filter

$$\hat{f}(x, y) = \text{median}_{(s,t) \in S_{xy}} g(s, t) \quad (4.25)$$

Max filters

$$\hat{f}(x, y) = \max_{(s,t) \in S_{xy}} g(s, t) \quad (4.26)$$

Min filters

$$\hat{f}(x, y) = \min_{(s,t) \in S_{xy}} g(s, t) \quad (4.27)$$

Midpoint filter

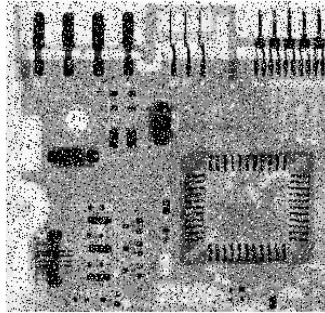
$$\hat{f}(x, y) = \frac{1}{2} [\max_{(s,t) \in S_{xy}} g(s, t) + \min_{(s,t) \in S_{xy}} g(s, t)] \quad (4.28)$$

Alpha-trimmed mean filter

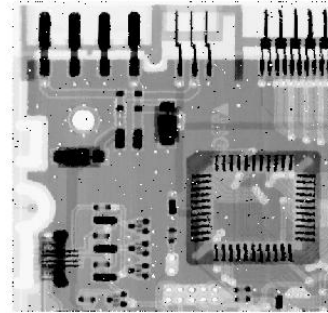
$$\hat{f}(x, y) = \frac{1}{mn - d} \sum_{(s,t) \in S_{xy}} g_r(s, t) \quad (4.29)$$

4.4.2 Results

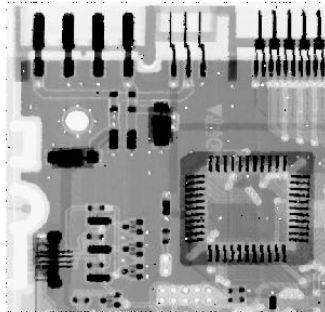
Image corrupted by salt-and-pepper noise



(b) Result of one pass with a median filter



(c) Result of processing (b) with the same filter



(d) Result of processing (c) with the same filter

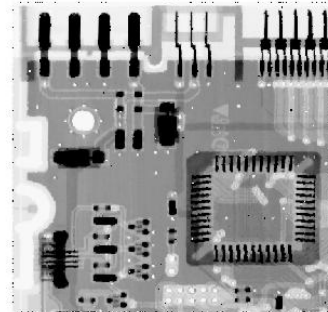


Figure 4.10: Original Book FIGURE 5.10

(a) Result of filtering pepper noise with a max filter (b) Result of filtering salt noise with a min filter

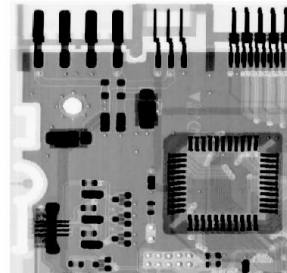
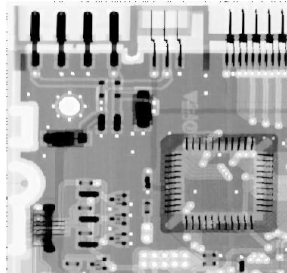
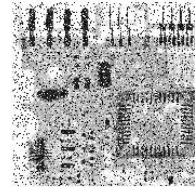
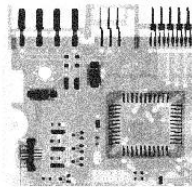
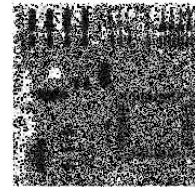
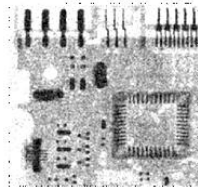


Figure 4.11: Original Book FIGURE 5.11

(a) Image corrupted by additive uniform noise (b) Image corrupted by additive salt-and-pepper noise



(c) Image (b) filtered with an arithmetic mean filter (d) Image (b) filtered with a geometric mean filter



(e) Image (b) filtered with a median filter (f) Image (b) filtered with an alpha-trimmed mean filter

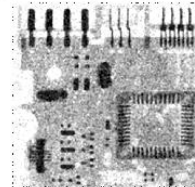
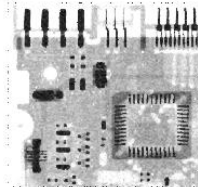


Figure 4.12: Original Book FIGURE 5.12

5.1 OverView

5.2 This is a section

5.2.1 This is a subsection

5.2.2 This is a subsection

A.1 Problem1

```

1 % Problem 1
2 % by Xue Fanyong
3 % Student ID:515030910443
4 % Histogram Equalizatio
5
6 %% Main Part
7 image1 = imread(' Image Path/Fig1.jpg' );
8 image2 = imread(' Image Path/Fig2.jpg' );
9
10 [histogram1,histogram_e1,transfer_f1,image_e1] =
11 histogram_equalization(image1);
12 [histogram2,histogram_e2,transfer_f2,image_e2] =
13 histogram_equalization(image2);
14
15 plot_data(image1,image_e1,histogram1,histogram_e1,transfer_f1);
16 plot_data(image2,image_e2,histogram2,histogram_e2,transfer_f2);
17
18 %% Functions Part
19
20 % get histogram of image
21 % image: get histogram of it
22 % histogram: the histogram of image
23 function histogram = get_histogram(image)
24     histogram = zeros(256,1);
25     [row,col]=size(image);
26     for r = 1:row
27         for c = 1:col
28             gray = image(r,c);
29             histogram(gray+1)=histogram(gray+1)+1;
30         end
31     end
32 end
33
34 % do the histogram_equalization for image
35 % image: do histogram_equalization for it
36 % histogram: original histogram; histogram_e:
37 % histogram after histogram
38 % equalizatio; transfer_f: transfer function;
39 % image_e: image after histogram
40 % equalizatio
41 function [histogram,histogram_e,transfer_f,image_e] =
42 histogram_equalization(image)
43     [row,col]=size(image);
44     transfer_f = zeros(256,1);
45     histogram = get_histogram(image);
46     transfer_f(1) = 256*histogram(1)/(row*col);
47
48     for i = 2:256
49         transfer_f(i) = transfer_f(i-1)+255*histogram(i)/(row*col);

```

```

50     end
51     transfer_f = round(transfer_f);
52
53     image_e = image;
54     for r = 1:row
55         for c = 1:col
56             image_e(r,c)=transfer_f(image(r,c)+1);
57         end
58     end
59     histogram_e = get_histogram(image_e);
60 end
61
62 % plot data
63 % image: original image; image_e:
64 % image after histogram equalization;
65 % histogram: original histogram;
66 % histogram_e: histogram after histogram equalization;
67 % transfer_f: transfer function
68 function plot_data(image,image_e,histogram,histogram_e,transfer_f)
69     figure();
70     subplot(2,3,1);
71     imshow(image);
72     title(" Original Image" );
73     subplot(2,3,2);
74     imshow(image_e);
75     title(" Image(Histogram Equalization)" );
76     subplot(2,3,3);
77     bar(histogram);
78     title(" Histogram" );
79     subplot(2,3,4);
80     bar(histogram_e);
81     title(" Histogram(Equalization)" );
82     subplot(2,3,5);
83     plot(transfer_f);
84     title(" Transfer Function" );
85 end

```

A.2 Problem 2

```

1 % Problem 2
2 % by Xue Fanyong
3 % Student ID:515030910443
4 % Combining spatial enhancement methods
5
6 %% Main Part
7 image = imread(' Image Path/skeleton_orig.tif ');
8 [row,col] = size(image);
9 mask = [-1 -1 -1;-1 8 -1;-1 -1 -1];
10 mask = double(mask);

```

```

11 b_image = laplace_transformations(image,mask);
12 c_image = b_image+im2double(image);
13 d_image = sobel_gradient(image);
14 e_image = smooth(d_image);
15 f_image = im2double(e_image).*c_image;
16 g_image = abs(f_image)+im2double(image);
17 h_image = sqrt(g_image);
18 plot_data(image,b_image,c_image,d_image,
19           e_image,f_image,g_image,h_image);
20
21 %% Function Part
22
23 % Laplace Transfromation for image using mask
24 % Input:
25 %   image:image you want to perform
26 %   mask:Laplace mask you want to use
27 % Output:
28 %   image_l:image after laplace transformation
29
30 function image_l = laplace_transformations(image,mask)
31     [row,col] = size(image);
32     mask = double(mask);
33     %append image
34     image_l = im2double(image);
35     image = [zeros(row,2) image zeros(row,2)];
36     image = [zeros(2,col+4);image;zeros(2,col+4)];
37     image_append = im2double(image);
38
39     for r = 1:row
40         for c = 1:col
41             image_l(r,c) = sum(sum(image_append(r:r+2,c:c+2).*mask));
42         end
43     end
44 end
45 %{
46     sobel gradient for image
47 %}
48 function image_s = sobel_gradient(image)
49     [row,col] = size(image);
50     x_mask = [-1 -2 -1;0 0 0;1 2 1];
51     y_mask = [-1 0 1;-2 0 2;-1 0 1];
52     image_s = image;
53     image = double(image);
54
55     for r = 2:row-1
56         for c = 2:col-1
57             image_s(r,c) =
58                 abs(sum(sum(image(r-1:r+1,c-1:c+1).*x_mask)))+
59                 abs(sum(sum(image(r-1:r+1,c-1:c+1).*y_mask)));
60         end
61     end

```

```

62 end
63 %{
64     smooth image using 5*5 mean filter
65 %}
66 function image_s = smooth(image)
67     [row,col] = size(image);
68     image_s = image;
69     for r = 3:row-2
70         for c = 3:col-2
71             image_s(r,c) = mean(mean(image(r-2:r+2,c-2:c+2)));
72         end
73     end
74 end
75 %{
76     plot data
77 %}
78 function plot_data(a,b,c,d,e,f,g,h)
79     figure();
80
81     subplot(241);
82     imshow(a);
83     title(' (a) Oringinal Image' );
84
85     subplot(242);
86     imshow(b,[]);
87     title(' (b) Laplacian of (a)' );
88
89     subplot(245);
90     imshow(c,[]);
91     title(' (c) Sharpened image' );
92
93     subplot(246);
94     imshow(d);
95     title(' (d) Sobel gradient' );
96
97     subplot(243);
98     imshow(e);
99     title(' (e) Smoothed sobel image' );
100
101     subplot(244);
102     imshow(f,[]);
103     title(' (f) Product of (c) and (e)' );
104
105     subplot(247);
106     imshow(g,[]);
107     title(' (g) Sharpened image' );
108
109     subplot(248);
110     imshow(h,[]);
111     title(' (h) Final result' );
112 end

```

A.3 Problem 3

```

1  %{
2      Problem 3
3      by Fanyong Xue
4      Student ID:515030910443
5      Filtering in frequency domain
6
7  %}
8
9  %% Main Part
10 image = imread(' Image Path/characters_test_pattern.tif ');
11
12 ideal_low_plot_data(image);
13 ideal_high_plot_data(image);
14 butterworth_low_plot_data(image);
15 butterworth_high_plot_data(image);
16 gaussian_low_plot_data(image);
17 gaussian_high_plot_data(image);
18
19
20 %% Function Part
21
22 % Ideal
23 function ideal_high_plot_data(image)
24
25     figure(' name' , 'Ideal High Pass' );
26
27     subplot(321);
28     imshow(image);
29     title(' Original Image' );
30
31     subplot(322);
32     imshow(IHPF(image,10),[]);
33     title(' Radius = 10' );
34
35     subplot(323);
36     imshow(IHPF(image,30),[]);
37     title(' Radius = 30' );
38
39     subplot(324);
40     imshow(IHPF(image,60),[]);
41     title(' Radius = 60' );
42
43     subplot(325);
44     imshow(IHPF(image,160),[]);
45     title(' Radius = 160' );
46
47     subplot(326);
48     imshow(IHPF(image,460),[]);
49     title(' Radius = 460' );

```

```

50
51 end
52 function ideal_low_plot_data(image)
53
54     figure('name','Ideal Low Pass');
55
56     subplot(321);
57     imshow(image);
58     title('Original Image');
59
60     subplot(322);
61     imshow(ILPF(image,10),[]);
62     title('Radius = 10');
63
64     subplot(323);
65     imshow(ILPF(image,30),[]);
66     title('Radius = 30');
67
68     subplot(324);
69     imshow(ILPF(image,60),[]);
70     title('Radius = 60');
71
72     subplot(325);
73     imshow(ILPF(image,160),[]);
74     title('Radius = 160');
75
76     subplot(326);
77     imshow(ILPF(image,460),[]);
78     title('Radius = 460');
79
80 end
81 function image_i = ILPF(image,radius)
82
83     fliter = ILPF_fliter(image,radius);
84     image_i = transfer(image,fliter);
85     %{
86     image_f = fft2(image,2*row,2*col);
87     image_f = fftshift(image_f);
88     %image_f = log(1+abs(image_f));
89
90     %%%%%%%%%%
91     image_i = image_f.*fliter;
92
93     image_i = ifftshift(image_i);
94     image_i = ifft2(image_i);
95
96     %image_i = abs(image_i);
97     image_i = image_i(1:row,1:col);
98     image_i = real(image_i);
99     image_f = log(1+abs(image_f));
100    %}

```



```

101 end
102 function image_i = IHPF(image,radius)
103     fliter = 1-ILPF_fliter(image,radius);
104     image_i = transfer(image,fliter);
105 end
106 function fliter = ILPF_fliter(image,radius)
107     [row,col]=size(image);
108     fliter = zeros(2*row,2*col);
109
110     for r = 1:2*row
111         for c = 1:2*col
112             if sqrt((r-row)^2+(c-col)^2) <= radius
113                 fliter(r,c) = 1;
114             end
115         end
116     end
117 end
118
119 % Butterworth
120 function butterworth_low_plot_data(image)
121     figure('name',' Butterworth Low Pass' );
122
123     subplot(321);
124     imshow(image);
125     title(' Original Image' );
126
127     subplot(322);
128     imshow(BLFP(image,2,10),[]);
129     title(' n=2,Radius = 10' );
130
131     subplot(323);
132     imshow(BLFP(image,2,30),[]);
133     title(' n=2,Radius = 30' );
134
135     subplot(324);
136     imshow(BLFP(image,2,60),[]);
137     title(' n=2,Radius = 60' );
138
139     subplot(325);
140     imshow(BLFP(image,2,160),[]);
141     title(' n=2,Radius = 160' );
142
143     subplot(326);
144     imshow(BLFP(image,2,460),[]);
145     title(' n=2,Radius = 460' );
146
147 end
148 function butterworth_high_plot_data(image)
149     figure('name',' Butterworth High Pass' );
150
151

```

```

152     subplot(321);
153     imshow(image);
154     title(' Original Image' );
155
156     subplot(322);
157     imshow(BHFP(image,2,10),[]);
158     title(' n=2,Radius = 10' );
159
160     subplot(323);
161     imshow(BHFP(image,2,30),[]);
162     title(' n=2,Radius = 30' );
163
164     subplot(324);
165     imshow(BHFP(image,2,60),[]);
166     title(' n=2,Radius = 60' );
167
168     subplot(325);
169     imshow(BHFP(image,2,160),[]);
170     title(' n=2,Radius = 160' );
171
172     subplot(326);
173     imshow(BHFP(image,2,460),[]);
174     title(' n=2,Radius = 460' );
175
176 end
177 function image_b = BLFP(image, n,radius)
178
179     fliter = BLFP_fliter(image, n,radius);
180     image_b = transfer(image,fliter);
181
182 end
183 function image_b = BHFP(image, n,radius)
184
185     fliter = 1-BLFP_fliter(image, n,radius);
186     image_b = transfer(image,fliter);
187
188 end
189 function fliter = BLFP_fliter(image, n,radius)
190     [row,col]=size(image);
191     fliter = zeros(2*row,2*col);
192
193     for r = 1:2*row
194         for c = 1:2*col
195             fliter(r,c) = 1/(1+(sqrt((r-row)^2+
196                                     (c-col)^2)/radius)^(2*n));
197         end
198     end
199 end
200
201 % Gaussian
202 function gaussian_low_plot_data(image)

```

```

203     figure( name , ' Gaussian Low Pass' );
204
205     subplot(321);
206     imshow(image);
207     title( ' Original Image' );
208
209     subplot(322);
210     imshow(GLFP(image,10) ,[]);
211     title( ' Radius = 10' );
212
213     subplot(323);
214     imshow(GLFP(image,30) ,[]);
215     title( ' Radius = 30' );
216
217     subplot(324);
218     imshow(GLFP(image,60) ,[]);
219     title( ' Radius = 60' );
220
221     subplot(325);
222     imshow(GLFP(image,160) ,[]);
223     title( ' Radius = 160' );
224
225     subplot(326);
226     imshow(GLFP(image,460) ,[]);
227     title( ' Radius = 460' );
228 end
229 function gaussian_high_plot_data(image)
230     figure( name , ' Gaussian High Pass' );
231
232     subplot(321);
233     imshow(image);
234     title( ' Original Image' );
235
236     subplot(322);
237     imshow(GHFP(image,10) ,[]);
238     title( ' Radius = 10' );
239
240     subplot(323);
241     imshow(GHFP(image,30) ,[]);
242     title( ' Radius = 30' );
243
244     subplot(324);
245     imshow(GHFP(image,60) ,[]);
246     title( ' Radius = 60' );
247
248     subplot(325);
249     imshow(GHFP(image,160) ,[]);
250     title( ' Radius = 160' );
251
252     subplot(326);
253     imshow(GHFP(image,460) ,[]);

```

```

254     title(' Radius = 460' );
255 end
256 function image_g = GLFP(image,radius)
257     fliter = GLFP_fliter (image,radius);
258     image_g = transfer (image,fliter );
259 end
260 function image_g = GHFP(image,radius)
261     fliter = 1-GLFP_fliter (image,radius);
262     image_g = transfer (image,fliter );
263 end
264 function fliter = GLFP_fliter (image,radius)
265     [row,col]=size(image);
266     fliter = double(zeros(2*row,2*col));
267
268     for r = 1:2*row
269         for c = 2:2*col
270             fliter(r,c) = exp(-1*(((r-row)^2+
271                                     (c-col)^2)/(2*radius^2)));
272         end
273     end
274 end
275
276 function image_t=transfer (image,fliter )
277     [row,col]=size(image);
278     image_f = fft2 (image,2*row,2*col);
279     image_f = fftshift (image_f);
280     image_t = image_f.* fliter;
281     image_t = ifftshift (image_t);
282     image_t = ifft2 (image_t);
283     image_t = image_t(1:row,1:col);
284     image_t = abs(image_t);
285 end
286 %{
287
288 function [ mfft2 ] = JCGuoFFT2( data )
289     h = size(data, 1);
290     w = size(data, 2);
291     mfft2 = data;
292
293     if power(2, log2(h)) ~= h || power(2, log2(w)) ~= w
294         disp(' JCGuoFFT2 exit: h and w must be the power of 2!' )
295     else
296         for i = 1 : h
297             mfft2(i, :) = IterativeFFT(mfft2(i, :));
298         end
299
300         for j = 1 : w
301             mfft2(:, j) = IterativeFFT(mfft2(:, j));
302         end
303     end
304 end

```

```

305
306 function image_s = shift_image(image)
307
308     [row,col]=size(image);
309     image_s = image;
310
311     for r = 1:row
312         for c = 1:col
313             image_s(r,c) = image(r,c)*(-1)^(r+c);
314         end
315     end
316 end
317
318
319 function image_f = DFT(image,rows,cols)
320     [row,col]=size(image);
321
322     %pad image to rows*cols
323     image = [image zeros(row,cols-col)];
324     image = [image;zeros(rows-row,cols)];
325     image = double(image);
326     for i = 1:rows
327         k = cols/2;
328         M = round(log2(cols));
329         for j = 1:cols-2
330             if j<k
331                 t = image(i,k);
332                 image(i,k) = image(i,j);
333                 image(i,j) = t;
334             end
335             l = cols/2;
336             while l<=k
337                 k = k-1;
338                 l = l/2;
339             end
340             k = k+1;
341         end
342         for m = 1:M
343             la = 2^m;
344             lb = la/2;
345             for l = 1:lb
346                 r = (l-1)*2^(M-m);
347                 n = l-1;
348                 while n<rows-1
349                     lc = n+lb;
350                     t = image(lc,j)*exp(2*pi*r/rows);
351                     image(i,lc) = image(i,n) - t;
352                     image(i,n) = image(i,n) + t;
353                     n = n+la;
354                 end
355             end

```

```

356         end
357     end
358     image_f = image;
359 end
360 function image_f=DFT(image,rows,cols)
361
362 end
363
364 function v = DFT_1(V)
365     n = length(V);
366     fft_m = BitReverseCopy(V);
367
368     for r = 1:log2(n)
369         m = power(2,r);
370         wm = exp(- 2 * pi * i / m);
371
372         for k = 0 : m : n - 1
373             w = 1;
374             for j = 0 : m / 2 - 1
375                 t = w * fft_m(k + j + m / 2 + 1);
376                 u = fft_m(k + j + 1);
377                 fft_m(k + j + 1) = u + t;
378                 fft_m(k + j + m / 2 + 1) = u - t;
379                 w = w * wm;
380             end
381         end
382     end
383 end
384 %}

```

A.4 Problem 4

```

1  %[
2      Problem 4
3      by Fanyong Xue
4      Student ID:515030910443
5      Generating different types of noise and comparing different noise reduction
6  %]
7
8  %% Main Part
9  image = imread(' Image Path/Fig0503.tif' );
10 plot_data_noises(image);
11
12 circuit = imread(' Image Path/Circuit.tif' );
13 plot_data_noises(circuit);
14 plot_mean_filter(circuit);
15 plot_order_statistic_filter(circuit);
16
17 %% Function Part
18 % plot data by mean filters

```

```

19 function plot_mean_filter(image)
20     figure(' name' , 'Mean Filters1' );
21
22     [row,col] = size(image);
23     image = im2double(image);
24     subplot(221);
25     imshow(image);
26     title(' (a) X-ray image' );
27
28     n = uniform_noise(row,col,0,0.3);
29     g = gaussian_noise(n,0,0.08);
30     image_b = image+g;
31     subplot(222);
32     imshow(image_b);
33     title(' (b) Image corrupted by additive Gaussian noise' );
34
35     subplot(223);
36     imshow(arithmetic_mean_filter(image_b,3,3));
37     title(' (c) Result of filtering with an arithmetic mean filter' );
38
39     subplot(224);
40     imshow(geometric_mean_filter(image_b,3,3));
41     title(' (d) Result of filtering with a geometric mean filter' );
42
43     figure(' name' , 'Mean Filters2' );
44
45     subplot(221);
46     image_a_ = impulse_noise(image,0.1,0,-1,0);
47     imshow(image_a_);
48     title(' (a) Image corrupted by pepper noise' );
49
50     subplot(222);
51     image_b_ = impulse_noise(image,0.1,0,1,0);
52     imshow(image_b_);
53     title(' (b) Image corrupted by salt noise' );
54
55
56     subplot(223);
57     imshow(contraharmonic_mean_filter(image_a_,1.5,3,3));
58     title(' (c) Result of filtering (a) with a contra-harmonic filter' );
59
60     subplot(224);
61     imshow(contraharmonic_mean_filter(image_b_,-1.5,3,3));
62     title(' (c) Result of filtering (a) with a contra-harmonic filter' );
63
64     figure(' name' , 'Mean Filters3' );
65
66     subplot(121);
67     imshow(contraharmonic_mean_filter(image_a_,-1.5,3,3));
68     title(' (c) Result of filtering (a) with a
69         contra-harmonic filter (Q=-1.5)' );

```

```

70
71 subplot(122);
72 imshow(contraharmonic_mean_filter(image_b_,1.5,3,3));
73 title(' (c) Result of filtering (a) with a
74         contra-harmonic filter(Q=1.5)' );
75 end
76 % plot data by order statistic filters
77 function plot_order_statistic_filter(image)
78     figure(' name' , 'Order-Statistic Filters1' );
79     image = im2double(image);
80     image_a = impulse_noise(image,0.1,0.1,-1,1);
81
82     subplot(221);
83     imshow(image_a);
84     title(' Image corrupted by salt-and-pepper noise' );
85
86     subplot(222);
87     image_b = median_filter(image_a,3,3);
88     imshow(image_b);
89     title(' (b) Result of one pass with a median filter' );
90
91     subplot(223);
92     image_c = median_filter(image_b,3,3);
93     imshow(image_c);
94     title(' (c) Result of processing (b) with the same filter' );
95
96     subplot(224);
97     imshow(median_filter(image_c,3,3));
98     title(' (d) Result of processing (c) with the same filter' );
99
100    figure(' name' , 'Order-Statistic Filters2' );
101
102    image_a__ = impulse_noise(image,0.1,0,-1,0);
103    subplot(121);
104    imshow(max_filter(image_a__,3,3));
105    title(' (a) Result of filtering pepper noise with a max filter' );
106
107    image_b__ = impulse_noise(image,0.1,0,1,0);
108    subplot(122);
109    imshow(min_filter(image_b__,3,3));
110    title(' (a) Result of filtering salt noise with a min filter' );
111
112    figure(' name' , 'Order-Statistic Filters3' );
113    [row,col] = size(image);
114    n = uniform_noise(row,col,0,0.3);
115    image_a_ = image + n;
116    subplot(321);
117    imshow(image_a_);
118    title(' (a) Image corrupted by additive uniform noise' );
119
120    image_b_ = impulse_noise(image_a_,0.1,0.1,-1,1);

```



```

121 subplot(322);
122 imshow(image_b_);
123 title(' (b) Image corrupted by additive salt-and-pepper noise' );
124
125 subplot(323);
126 imshow(arithmetic_mean_filter(image_b_,5,5));
127 title(' (c) Image (b) filtered with an arithmetic mean filter' );
128
129 subplot(324);
130 imshow(geometric_mean_filter(image_b_,5,5));
131 title(' (c) Image (b) filtered with a geometric mean filter' );
132
133 subplot(325);
134 imshow(median_filter(image_b_,5,5));
135 title(' (c) Image (b) filtered with a median filter' );
136
137 subplot(326);
138 imshow(alpha_trimmed_mean_filter(image_b_,5,5,5));
139 title(' (c) Image (b) filtered with an alpha-trimmed mean filter' );
140 end
141 % arithmetic mean filter with m*n mask
142 function image_ = arithmetic_mean_filter(image,m,n)
143     [row,col] = size(image);
144     image_ = image;
145     %m and n should be odd numbers
146     m_ = floor(m/2);
147     n_ = floor(n/2);
148     for r = ceil(m/2):row-m_
149         for c = ceil(n/2):col-n_
150             image_(r,c) = mean(mean(image(r-m_:r+m_,c-n_:c+n_)));
151         end
152     end
153 end
154 % geometric mean filter with m*n mask
155 function image_ = geometric_mean_filter(image,m,n)
156     [row,col] = size(image);
157     image_ = image;
158     m_ = floor(m/2);
159     n_ = floor(n/2);
160     for r = ceil(m/2):row-m_
161         for c = ceil(n/2):col-n_
162             image_(r,c) = nthroot(prod(prod(
163                 image(r-m_:r+m_,c-n_:c+n_))),m*n);
164         end
165     end
166 end
167 % harmonic mean filter with m*n mask
168 function image_ = harmonic_mean_filter(image,m,n)
169     [row,col] = size(image);
170     image_ = image;
171     m_ = floor(m/2);

```

```

172     n_ = floor(n/2);
173     for r = ceil(m/2):row-m_
174         for c = ceil(n/2):col-n_
175             image_(r,c) = (m*n)/
176                 (sum(sum(1./image(
177                     r-m_:r+m_,c-n_:c+n_))));
178         end
179     end
180 end
181 % contraharmonic mean filter with m*n mask and its order is q
182 function image_ = contraharmonic_mean_filter(image,q,m,n)
183     [row,col] = size(image);
184     image_ = image;
185     m_ = floor(m/2);
186     n_ = floor(n/2);
187     for r = ceil(m/2):row-m_
188         for c = ceil(n/2):col-n_
189             image_(r,c) = (sum(sum(image(
190                 r-m_:r+m_,c-n_:c+n_).^(q+1))))/
191                 (sum(sum(image(
192                     r-m_:r+m_,c-n_:c+n_).^q)));
193         end
194     end
195     image_ = real(image_);
196 end
197 % median filter with m*n mask
198 function image_ = median_filter(image,m,n)
199     [row,col] = size(image);
200     image_ = image;
201     m_ = floor(m/2);
202     n_ = floor(n/2);
203     for r = ceil(m/2):row-m_
204         for c = ceil(n/2):col-n_
205             image_(r,c) = median(median(image(r-m_:r+m_,c-n_:c+n_)));
206         end
207     end
208 end
209 % max filter with m*n mask
210 function image_ = max_filter(image,m,n)
211     [row,col] = size(image);
212     image_ = image;
213     m_ = floor(m/2);
214     n_ = floor(n/2);
215     for r = ceil(m/2):row-m_
216         for c = ceil(n/2):col-n_
217             temp = image(r-m_:r+m_,c-n_:c+n_);
218             image_(r,c) = max(temp(:));
219         end
220     end
221 end
222 % min filter with m*n mask

```

```

223 function image_ = min_filter(image,m,n)
224     [row,col] = size(image);
225     image_ = image;
226     m_ = floor(m/2);
227     n_ = floor(n/2);
228     for r = ceil(m/2):row-m_
229         for c = ceil(n/2):col-n_
230             temp = image(r-m_:r+m_,c-n_:c+n_);
231             image_(r,c) = min(temp(:));
232         end
233     end
234 end
235 % midpoint filter with m*n mask
236 function image_ = midpoint_filter(image,m,n)
237     [row,col] = size(image);
238     image_ = image;
239     m_ = floor(m/2);
240     n_ = floor(n/2);
241     for r = ceil(m/2):row-m_
242         for c = ceil(n/2):col-n_
243             temp = image(r-m_:r+m_,c-n_:c+n_);
244             image_(r,c) = (max(temp(:)) + min(temp(:))) / 2;
245         end
246     end
247 end
248 % alpha trimmed mean filter with m*n mask(deleta d pixels)
249 function image_ = alpha_trimmed_mean_filter(image,d,m,n)
250     [row,col] = size(image);
251     image_ = image;
252     m_ = floor(m/2);
253     n_ = floor(n/2);
254     for r = ceil(m/2):row-m_
255         for c = ceil(n/2):col-n_
256             temp = image(r-m_:r+m_,c-n_:c+n_);
257             temp_ = sort(temp(:));
258             image_(r,c) = mean(temp_(floor(d/2):m*n-floor(d/2)));
259         end
260     end
261 end
262
263 % adding noises to image and plot them
264 function plot_data_noises(image)
265
266     [row,col] = size(image);
267     image = im2double(image);
268     figure('name','Uniform Noise');
269     n = uniform_noise(row,col,0,0.3);
270     image_ = image + n;
271     subplot(121);
272     imshow(image_,[]);
273     subplot(122);

```

```

274     bar(get_histogram(image_));
275
276     figure(' name' , 'Gaussian Noise' );
277     g = gaussian_noise(n,0,0.08);
278     image_ = image+g;
279     subplot(121);
280     imshow(image_,[]);
281     subplot(122);
282     bar(get_histogram(image_));
283
284     figure(' name' , 'Rayleigh Noise' );
285     r = rayleigh_noise(n,-0.2,0.03);
286     image_ = image+r;
287     subplot(121);
288     imshow(image_,[]);
289     subplot(122);
290     bar(get_histogram(image_));
291
292
293     figure(' name' , 'Exponential Noise' );
294     e = exponential_noise(n,25);
295     image_ = image+e;
296     subplot(121);
297     imshow(image_,[]);
298     subplot(122);
299     bar(get_histogram(image_));
300
301     figure(' name' , 'Gamma Noise' );
302     ga = gamma_noise(n,25,3);
303     image_ = image+ga;
304     subplot(121);
305     imshow(image_,[]);
306     subplot(122);
307     bar(get_histogram(image_));
308
309     figure(' name' , 'Impulse Noise' );
310     image_ = impulse_noise(image,0.1,0.1,0.2,-0.2);
311     subplot(121);
312     imshow(image_,[]);
313     subplot(122);
314     bar(get_histogram(image_));
315 end
316 % row*col uniform noise from low~high
317 function n = uniform_noise(row,col,low,high)
318     n = low + (high-low)*rand([row col]);
319 end
320 % normalizing uniform noise to 0~1
321 function n = normalized(uniform_noise)
322     max_ = max(uniform_noise(:));
323     min_ = min(uniform_noise(:));
324     n = double(uniform_noise-min_);

```

```

325     n = n/double(max_-min_);
326 end
327 % rayleigh noise
328 function n = rayleigh_noise(uniform_noise,a,b)
329     uniform_noise_ = normalized(uniform_noise);
330     n = a + sqrt(-b*log(1-uniform_noise_));
331 end
332 % exponential noise
333 function n = exponential_noise(uniform_noise,a)
334     uniform_noise_ = normalized(uniform_noise);
335     n = -1/a*log(1-uniform_noise_);
336 end
337 % impulse noise
338 function image_ = impulse_noise(image,pa,pb,a,b)
339     [row,col]=size(image);
340     uniform_noise_ = rand([row col]);
341     image_ = image;
342     for r = 1:row
343         for c = 1:col
344             if uniform_noise_(r,c)<pa
345                 image_(r,c) = image(r,c)+a;
346             elseif uniform_noise_(r,c)>(1-pb)
347                 image_(r,c) = image(r,c)+b;
348             end
349         end
350     end
351 end
352 % gamma noise
353 function n = gamma_noise(uniform_noise_,a,b)
354     uniform_noise_ = normalized(uniform_noise_);
355     n = -1/a*log(1-uniform_noise_);
356     [row,col] = size(uniform_noise_);
357     for i = 2:b
358         uniform_noise__=uniform_noise(row,col,0,1);
359         n = n-1/a*log(1-uniform_noise__);
360     end
361 end
362 % gaussian noise
363 function n = gaussian_noise(uniform_noise,ex,sigma)
364     [row,col] = size(uniform_noise);
365     n_ = normalized(uniform_noise);
366     n = n_;
367     for r = 1:row
368         if mod(col,2)==0
369             for c = 1:2:col
370                 n(r,c) = sqrt(-2*log(n_(r,c)))*cos(2*pi*n_(r,c+1));
371                 n(r,c+1) = sqrt(-2*log(n_(r,c)))*sin(2*pi*n_(r,c+1));
372             end
373         else
374             for c = 1:2:col-1

```

```

376         n(r,c) = sqrt(-2*log(n_(r,c)))*cos(2*pi*n_(r,c+1));
377         n(r,c+1) = sqrt(-2*log(n_(r,c)))*sin(2*pi*n_(r,c+1));
378     end
379     n(r,col) = sqrt(-2*log(n_(r,c)))*cos(2*pi*n_(r,1));
380 end
381 end
382 n = n*double(sigma);
383 n = n+ex;
384 end
385 % plot image' histogram (but add other 150 to display the negative value)
386 function histogram = get_histogram(image)
387     [row,col]=size(image);
388     histogram = zeros(500,1);%-150~350
389     for r = 1:row
390         for c = 1:col
391             gray = int16(image(r,c) /0.004);
392             if gray >349
393                 gray = 349;
394             end
395             if gray <-150
396                 gray = -150;
397             end
398             histogram(gray+1+150)=histogram(gray+1+150)+1;
399         end
400     end
401 end

```