

Thierry CANTENOT

REPORT

Digital Image Processing

« Assignments »



2014-2015

Summary

A Histogram Equalization	1
A.1 Problem statement	1
A.2 Python implementation	1
A.3 Figure 1	2
A.3.1 Histogram	2
A.3.2 Histogram equalization	3
A.4 Figure 2	4
A.4.1 Histogram	4
A.4.2 Histogram equalization	5
B Spatial enhancement methods	6
B.1 Problem statement	6
B.2 Python implementation	6
B.3 Results	7
B.3.1 Original image	7
B.3.2 3x3 Laplacian ($A = 0$)	8
B.3.3 3x3 Laplacian ($A = 1$)	9
B.3.4 3x3 Laplacian ($A = 1.7$)	10
B.3.5 Sobel	11
B.3.6 Smoothing, Sharpening and Power-Law transformation	12
B.3.7 Comparison	13
C Filtering in frequency domain	14
C.1 Problem statement	14
C.2 Python implementation	14
C.3 Results	14
C.3.1 Original image	14
C.3.2 Ideal filter	15
C.3.3 Butterworth order 2 filter	17

C.3.4	Butterworth order 5 filter	19
C.3.5	Gaussian filter	21
D	Noise generation and noise reduction	23
D.1	Problem statement	23
D.2	Python implementation	23
D.3	Gaussian noise	24
D.3.1	Noise generation and histogram	24
D.3.2	Noise reduction	26
D.4	Uniform noise	33
D.4.1	Noise generation and histogram	33
D.4.2	Noise reduction	35
E	Image restoration	42
E.1	Problem statement	42
E.2	Python implementation	42
E.3	Blurring and noising	43
E.4	Restoration	45
E.4.1	Inverse filter	45
E.4.2	Wiener filter	46
E.4.3	Experimentations	47
F	Geometric transforms	51
F.1	Problem statement	51
F.2	Python implementation	51
F.3	Translation	53
F.4	Rotation	56
F.5	Scaling	59
G	Transform image compression	62
G.1	Problem statement	62
G.2	Python implementation	62
G.3	Image compression based on DCT	64
G.3.1	Zonal mask	64
G.4	Image compression based on Wavelets	66
G.4.1	Haar	67
G.4.2	Daubechies	68
G.4.3	Symlet	69

G.4.4 Cohen-Daubechies-Feauveau	70
H Morphological Processing	71
H.1 Problem statement	71
H.2 Python implementation	71
H.3 Erosion	72
H.4 Dilation	72
H.5 Opening	73
H.6 Closing	73
H.7 Boundary extraction	74
H.8 Hole filling	74
H.9 Connected component extraction	75
I Image segmentation	76
I.1 Problem statement	76
I.2 Python implementation	76
I.3 Marr-Hildreth edge detector	77
I.4 Canny edge detector	78
I.4.1 Roberts	78
I.4.2 Prewitt	79
I.4.3 Sobel	80
I.5 Thresholding segmentation	80
I.5.1 Otsu	81
I.5.2 Global thresholding	81
J Image representation and description	83
J.1 Problem statement	83
J.2 Python implementation	83
J.3 Boundary following	84
J.4 Resampling grid	86
J.5 Chain code and first difference chain code	87
J.5.1 Chain code - resampling grid (10, 10)	87
J.5.2 Chain code - resampling grid (30, 30)	88
J.6 Principal components	89

A. Histogram Equalization

A.1 Problem statement

1. Write a computer program for computing the histogram of an image.
2. Implement the histogram equalization technique.
3. Your program must be general to allow any gray-level image as its input.

A.2 Python implementation

Usage : **python problem1.py [-h] image_path** Use **python problem1.py -h** to see the help.

A.3 Figure 1

A.3.1 Histogram

Original image : [A.1](#) | Original image's histogram : [A.2](#)



FIGURE A.1 – Original Fig1.jpg

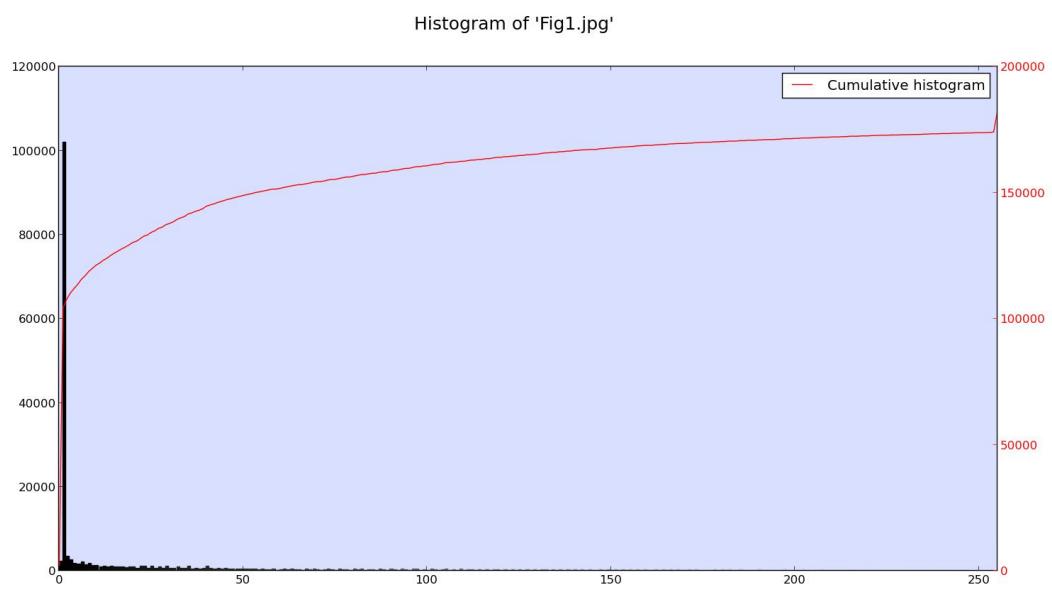


FIGURE A.2 – Histogram of Fig1.jpg

A.3.2 Histogram equalization

Enhanced image : [A.3](#) | Enhanced image's histogram : [A.4](#)



FIGURE A.3 – Enhanced Fig1.jpg

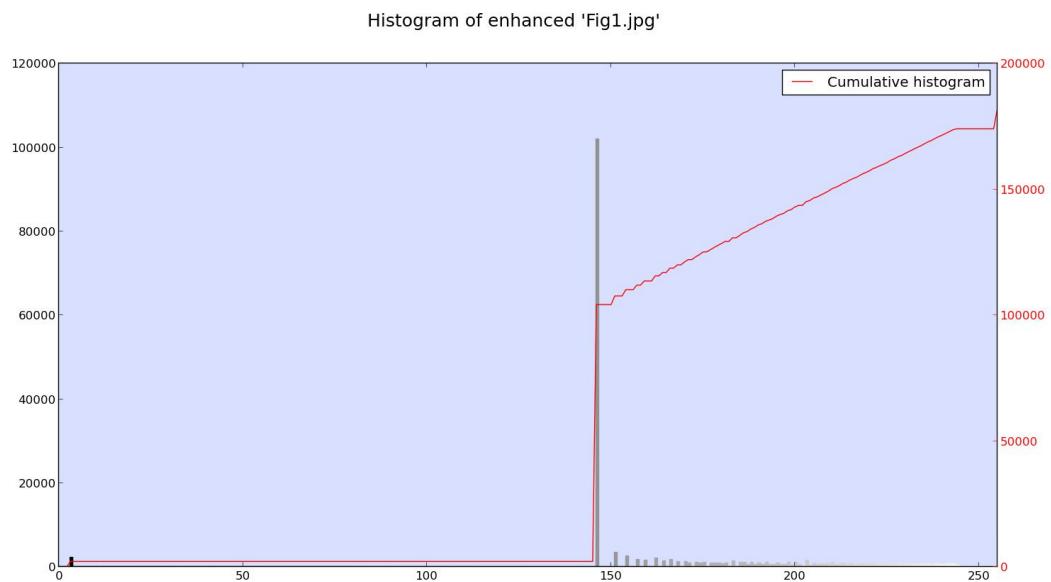


FIGURE A.4 – Equalized histogram of Fig1.jpg

A.4 Figure 2

A.4.1 Histogram

Original image : [A.5](#) | Original image's histogram : [A.6](#)

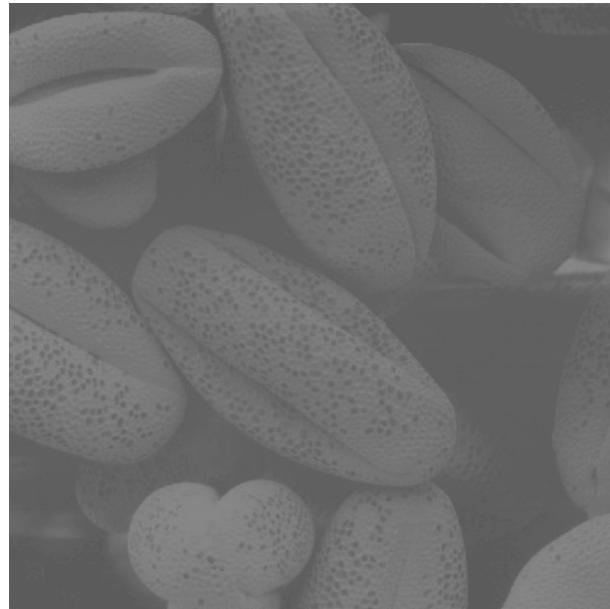


FIGURE A.5 – Original *Fig2.jpg*

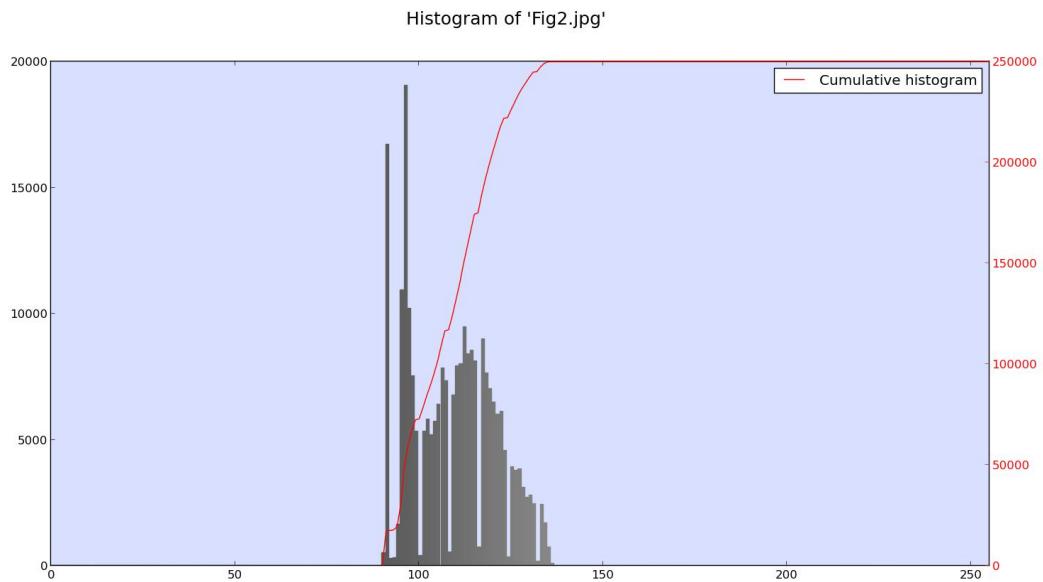


FIGURE A.6 – Histogram of *Fig2.jpg*

A.4.2 Histogram equalization

Enhanced image : [A.7](#) | Enhanced image's histogram : [A.8](#)

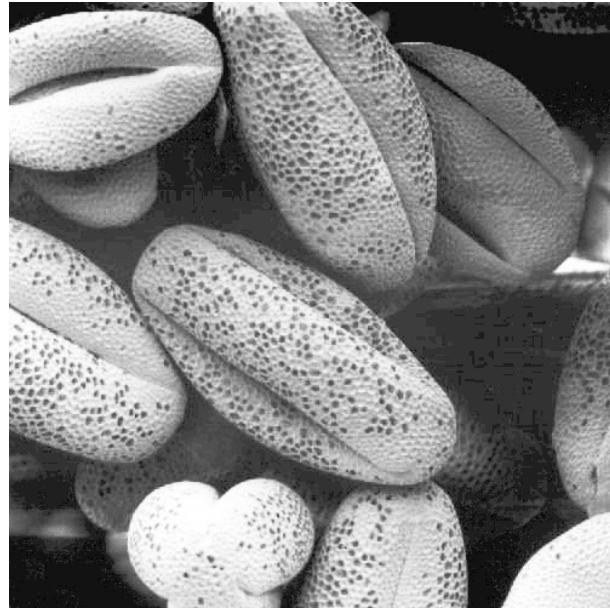


FIGURE A.7 – Enhanced Fig2.jpg

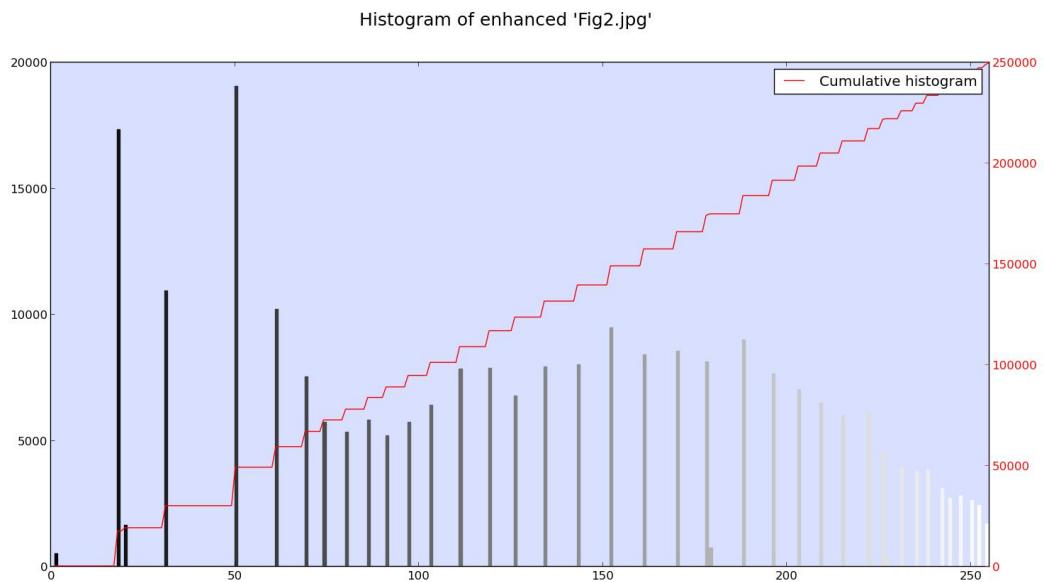


FIGURE A.8 – Equalized histogram of Fig2.jpg

B. Spatial enhancement methods

B.1 Problem statement

Implement the image enhancement task of Section 3.7 (Fig 3.43) (Section 3.8, Fig 3.46 in our slides).

The image to be enhanced is *skeleton_orig.tif*.

You should implement all steps in Figure 3.43.

(You cannot directly use functions of Matlab such as imfilter or fspecial, implement all functions by yourself).

B.2 Python implementation

Usage : `python problem2.py [-h] [-laplacian] [-sobel] [-a A] [-g G] [-c C] image_path`

For example, to run the full image enhancement described in the assignment, using a 3×3 Laplacian filter with $A = 1.7$, then a Sobel, a smoothing filter and a Power-Law transformation with $c = 1$ and gamma = 0.5 type :

```
python problem2.py -laplacian -a 1.7 -sobel -g 0.5 -c 1 skeleton_orig.tif
```

B.3 Results

B.3.1 Original image

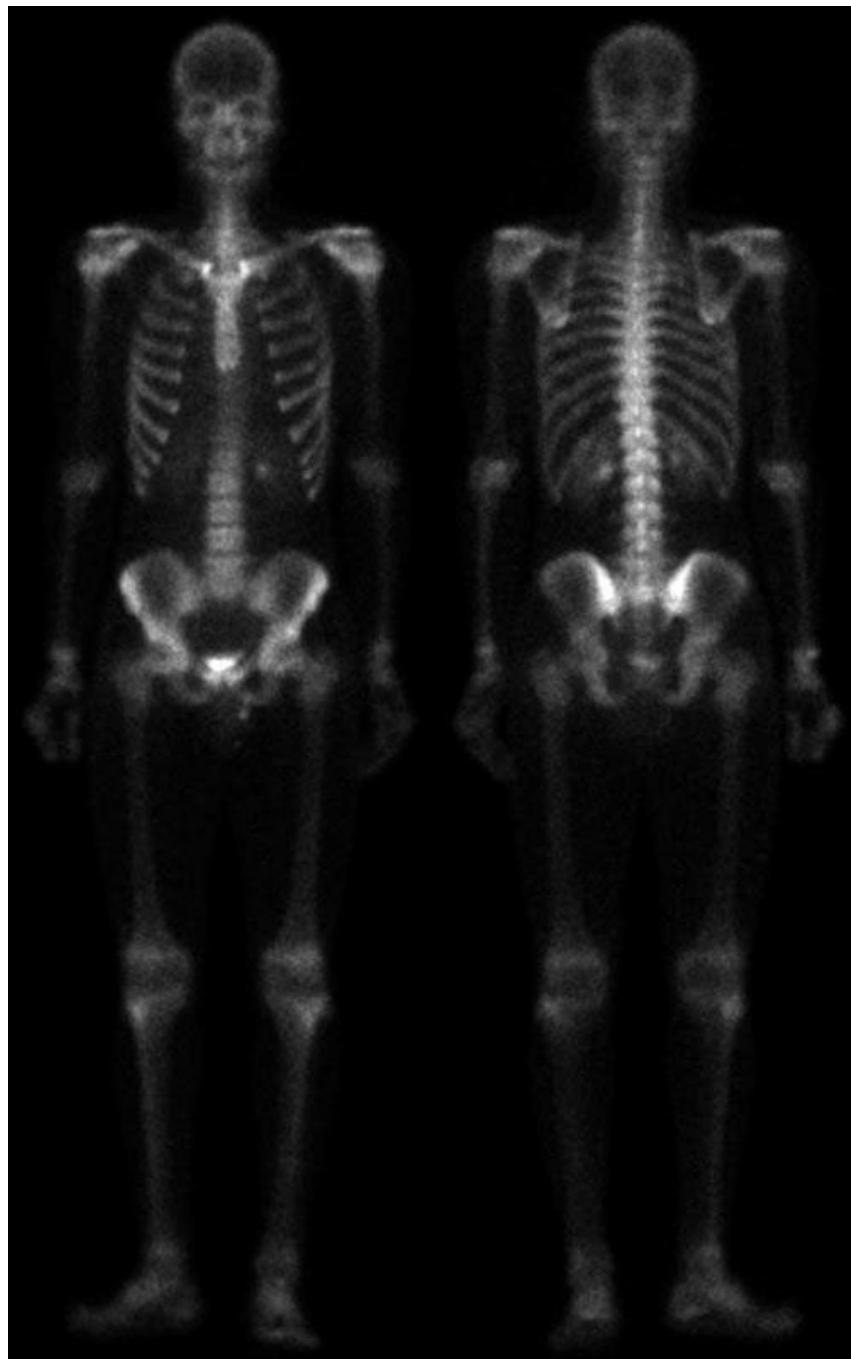


FIGURE B.1 – Original *skeleton_orig.tif*

B.3.2 3x3 Laplacian ($A = 0$)

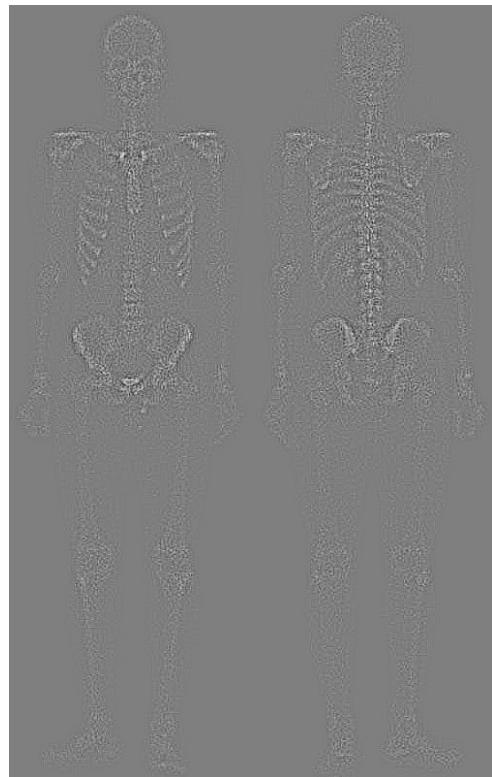


FIGURE B.2 – Laplacian ($A=0$)

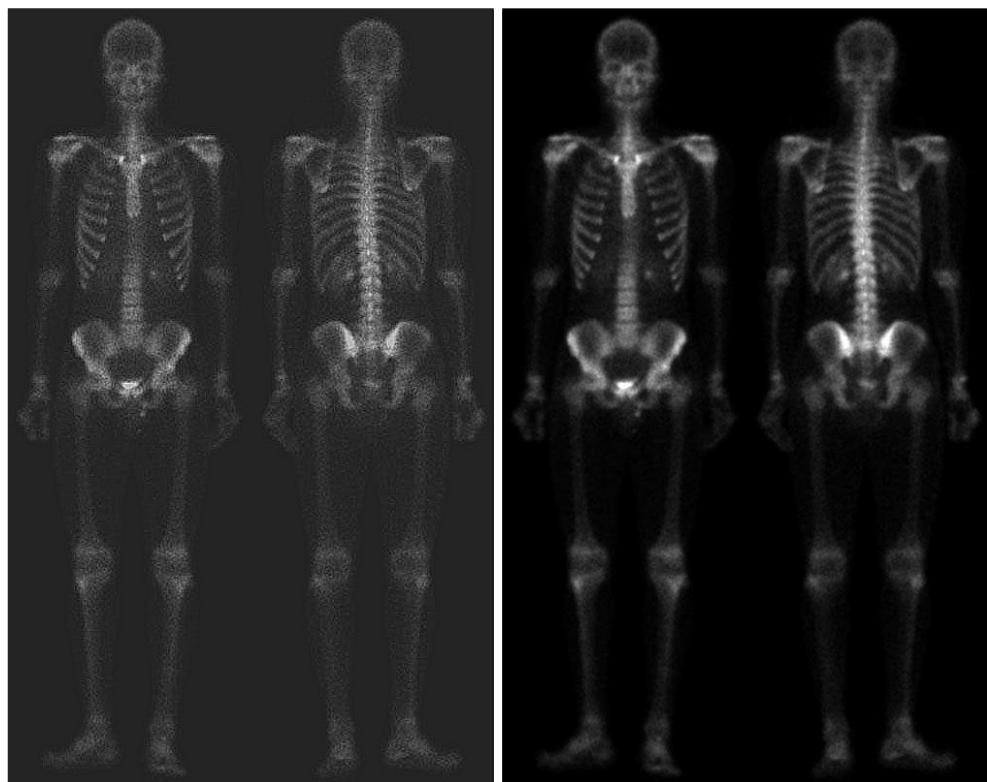


FIGURE B.3 – Sharpened image

FIGURE B.4 – Original image

B.3.3 3x3 Laplacian ($A = 1$)



FIGURE B.5 – Laplacian ($A=1$)

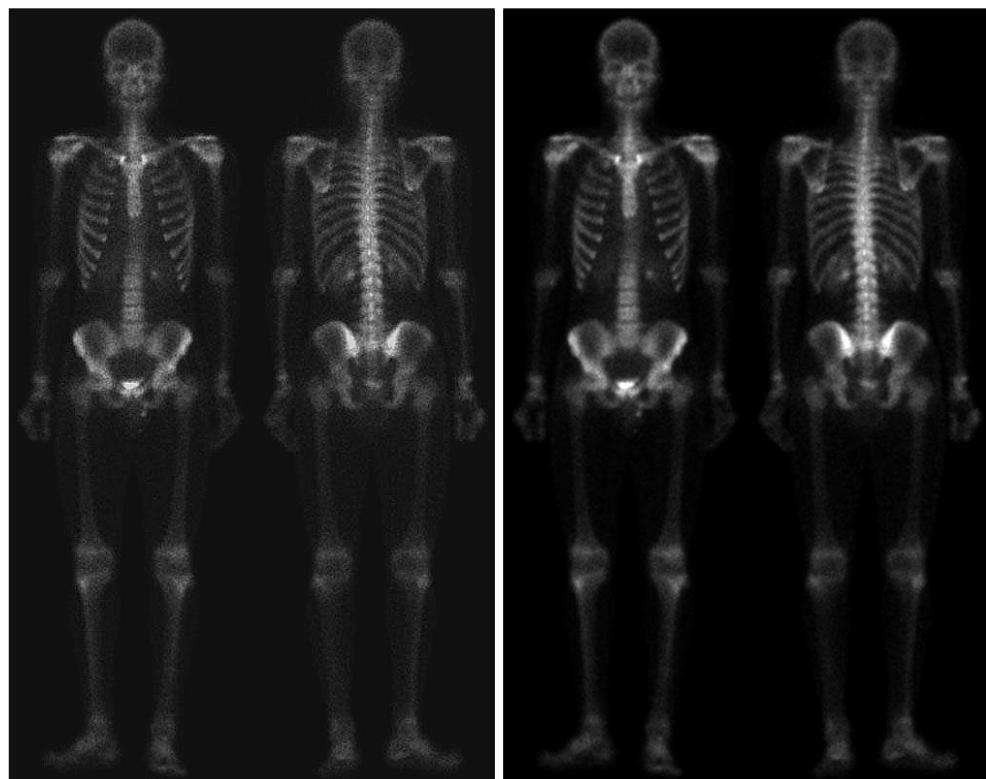


FIGURE B.6 – Sharpened image

FIGURE B.7 – Original image

B.3.4 3x3 Laplacian ($A = 1.7$)

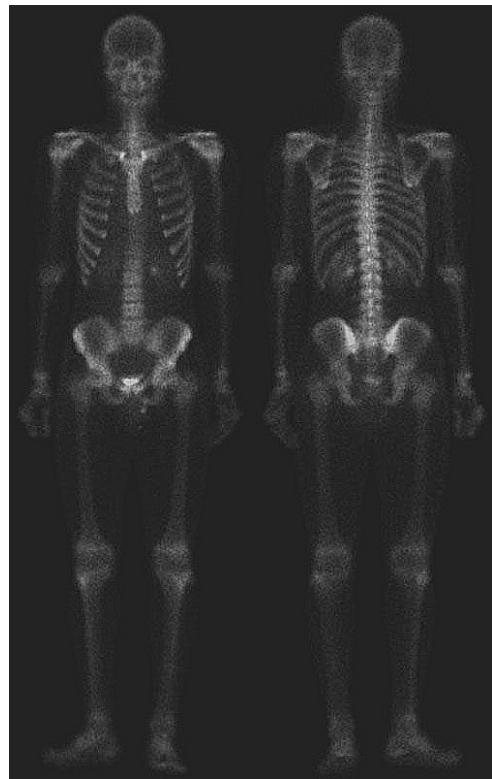


FIGURE B.8 – Laplacian ($A=1.7$)

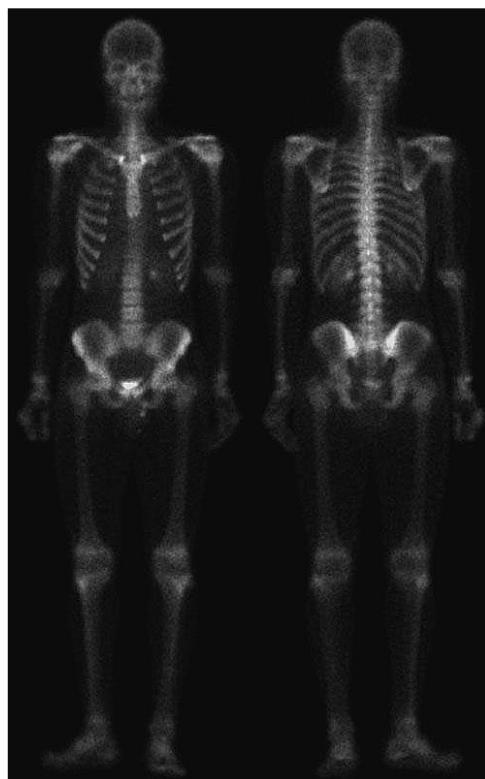


FIGURE B.9 – Sharpened image

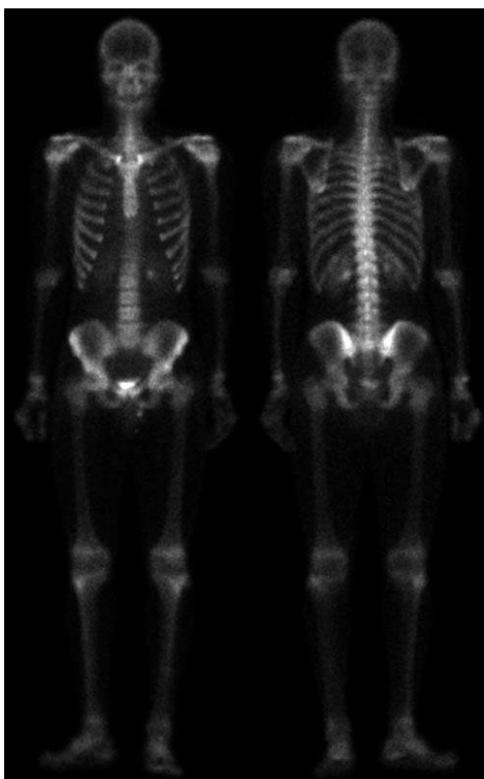


FIGURE B.10 – Original image

B.3.5 Sobel

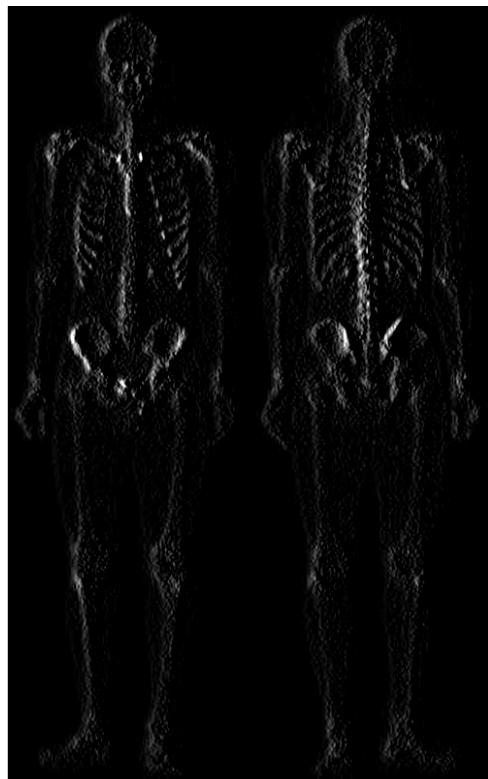


FIGURE B.11 – Sobel x-gradient

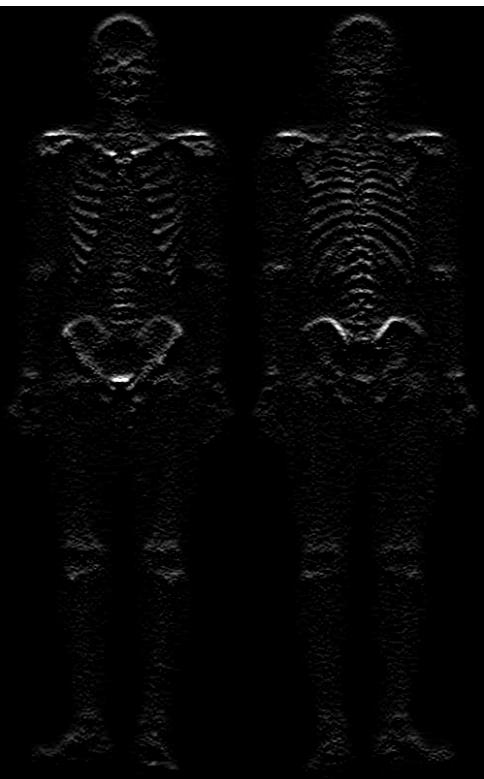


FIGURE B.12 – Sobel y-gradient



FIGURE B.13 – Full Sobel



FIGURE B.14 – Original image

B.3.6 Smoothing, Sharpening and Power-Law transformation

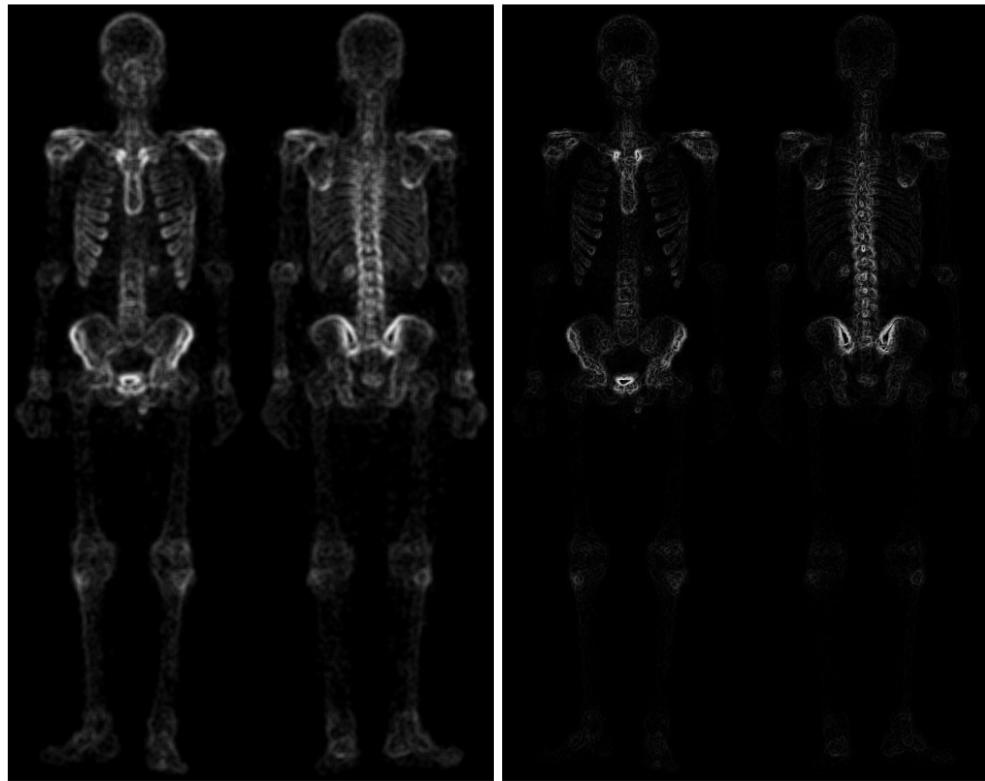


FIGURE B.15 – Smooth Sobel

FIGURE B.16 – Laplacian x Sobel

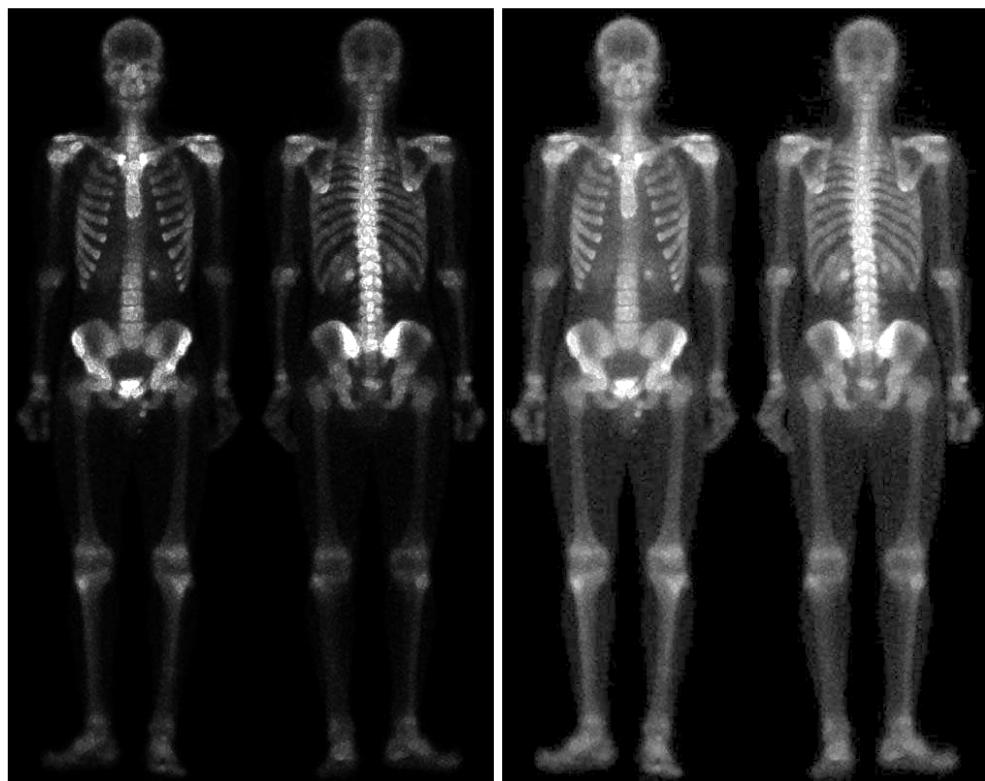


FIGURE B.17 – Original + Laplacian x Smooth Sobel

FIGURE B.18 – Final image (after Power-Law ($c=1, g=0.5$)))

B.3.7 Comparison

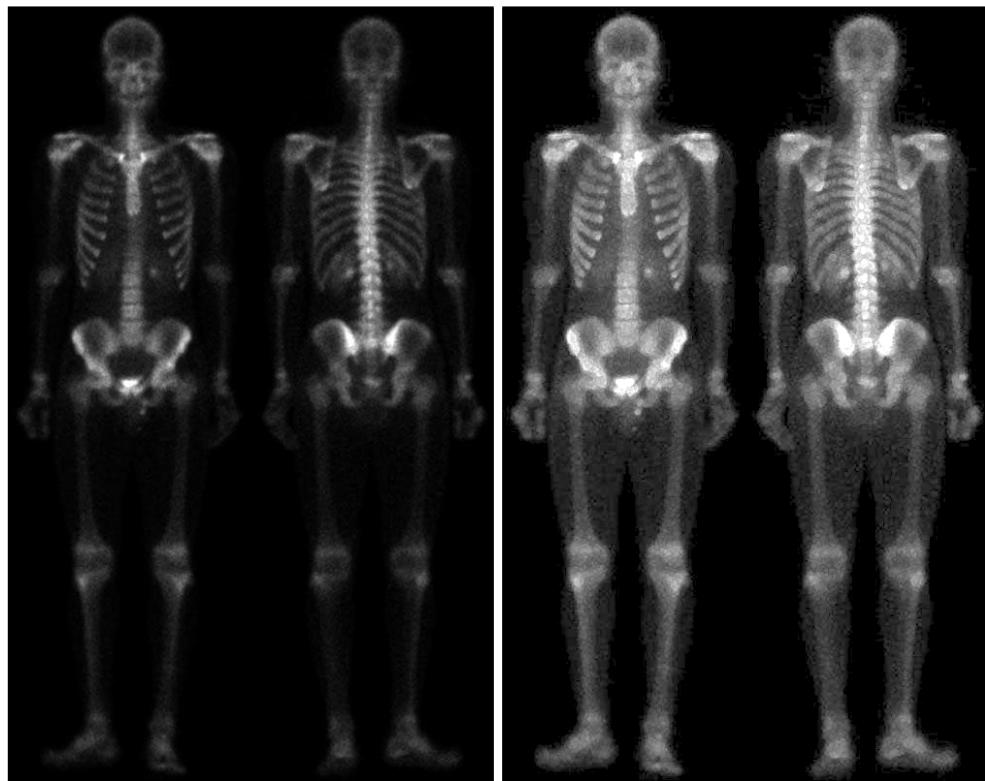


FIGURE B.19 – Original

FIGURE B.20 – Final image

C. Filtering in frequency domain

C.1 Problem statement

Implement the ideal, Butterworth and Gaussian lowpass and highpass filters and test them under different parameters using *characters_test_pattern.tif*.

C.2 Python implementation

```
Usage : python problem3.py [-h] [-ideal] [-butterworth] [-gaussian]
(-low | -high) [-d D] [-n N] image_path
```

Use `python problem3.py -h` to see the help.

C.3 Results

C.3.1 Original image

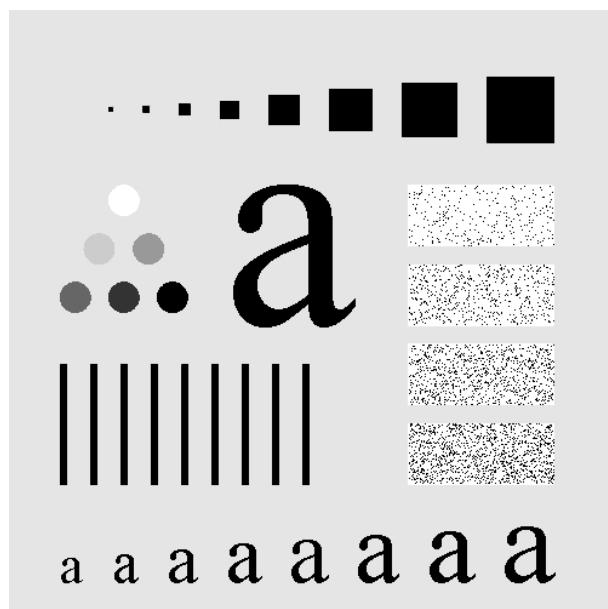


FIGURE C.1 – Original *characters_test_pattern.tif*

C.3.2 Ideal filter

Low pass

```
python problem3.py -ideal -d 5 -low characters_test_pattern.tif
```

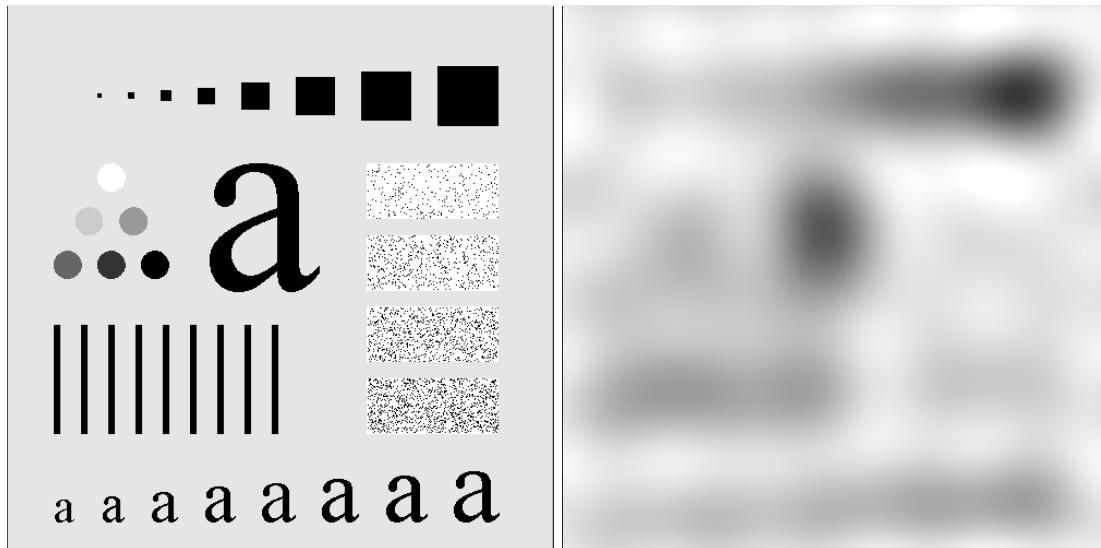


FIGURE C.2 – Original image

FIGURE C.3 – Ideal low 5

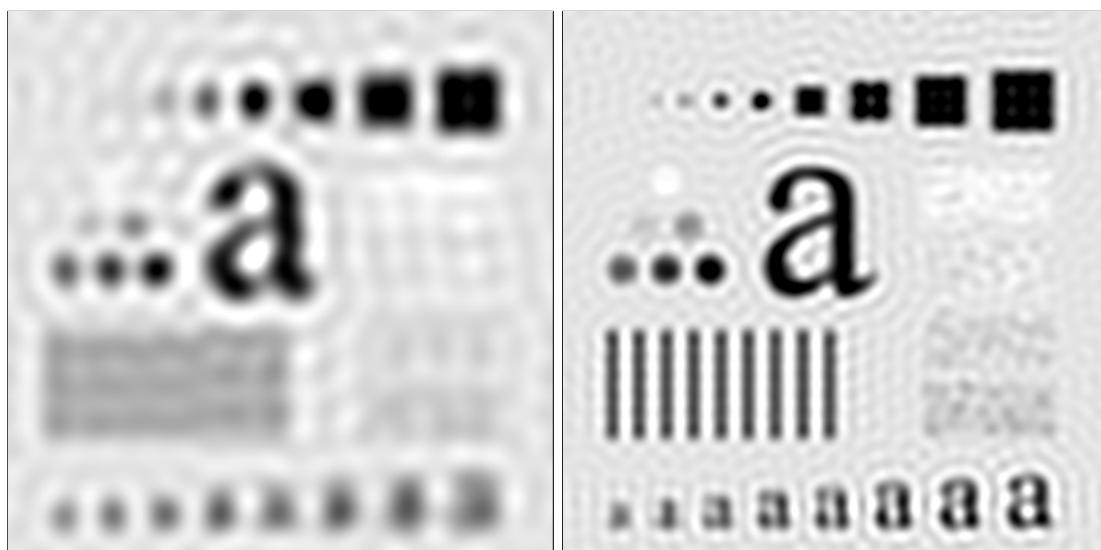


FIGURE C.4 – Ideal low 15

FIGURE C.5 – Ideal low 30

High pass

```
python problem3.py -ideal -d 5 -high characters_test_pattern.tif
```

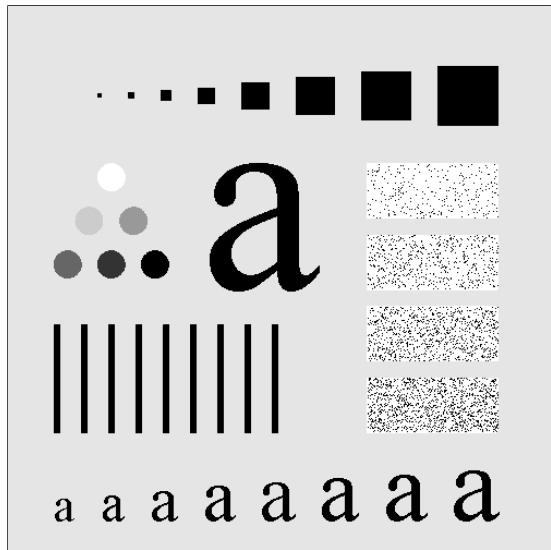


FIGURE C.6 – Original image



FIGURE C.7 – Ideal high 5

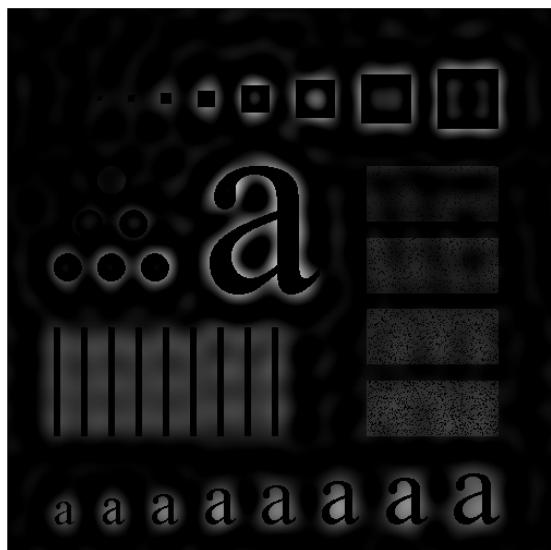


FIGURE C.8 – Ideal high 15



FIGURE C.9 – Ideal high 30

C.3.3 Butterworth order 2 filter

Low pass

```
python problem3.py -butterworth -d 5 -n 2 -low characters_test_pattern.tif
```



FIGURE C.10 – Original image

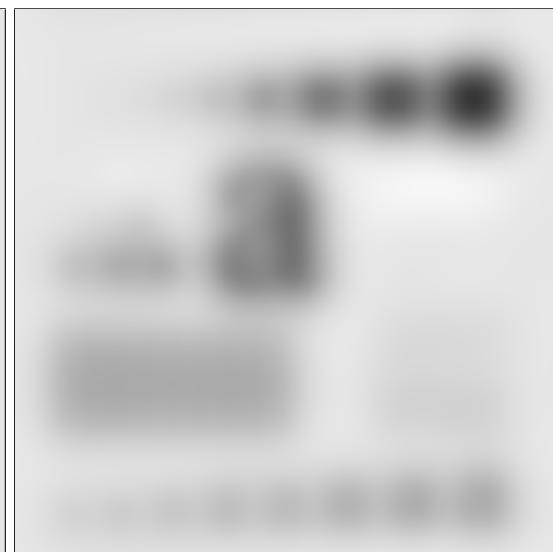


FIGURE C.11 – Butterworth low 5



FIGURE C.12 – Butterworth low 15



FIGURE C.13 – Butterworth low 30

High pass

```
python problem3.py -butterworth -d 5 -n 2 -high characters_test_pattern.tif
```

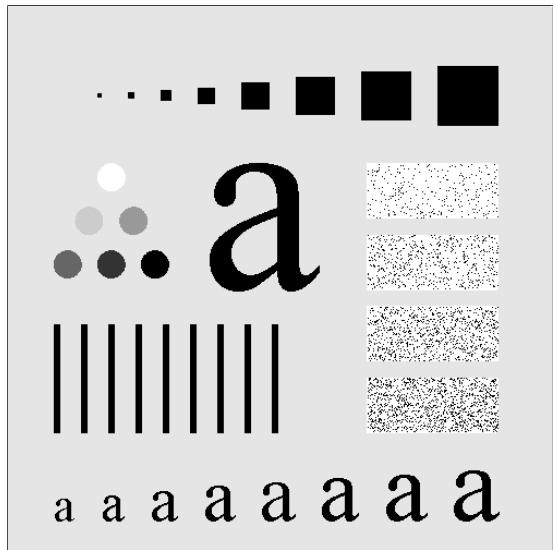


FIGURE C.14 – Original image

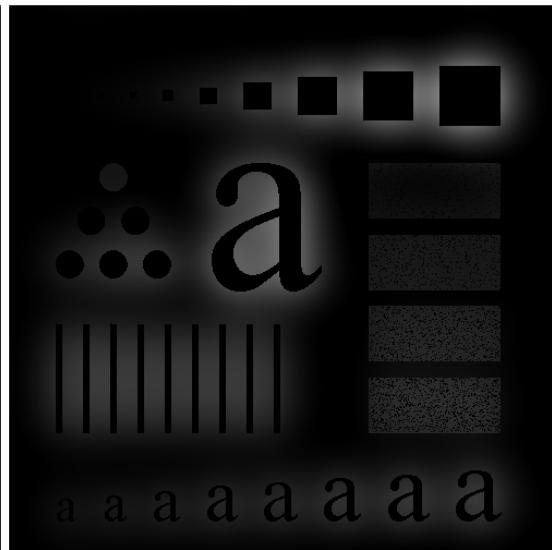


FIGURE C.15 – Butterworth high 5

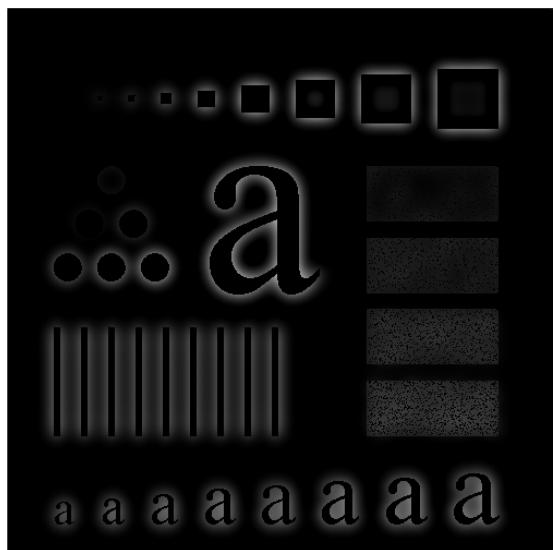


FIGURE C.16 – Butterworth high 15

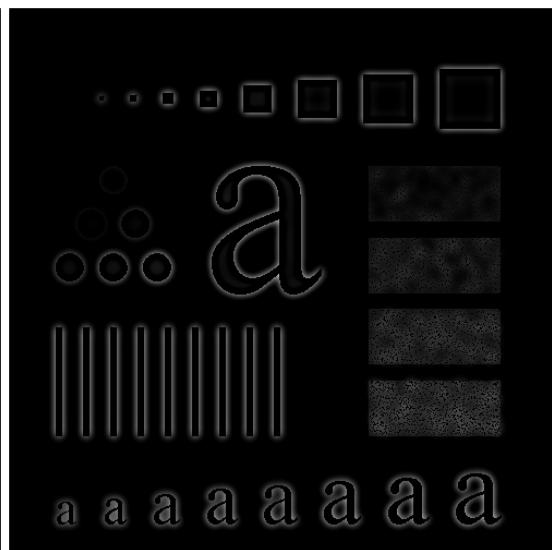


FIGURE C.17 – Butterworth high 30

C.3.4 Butterworth order 5 filter

Low pass

```
python problem3.py -butterworth -d 5 -n 5 -low characters_test_pattern.tif
```

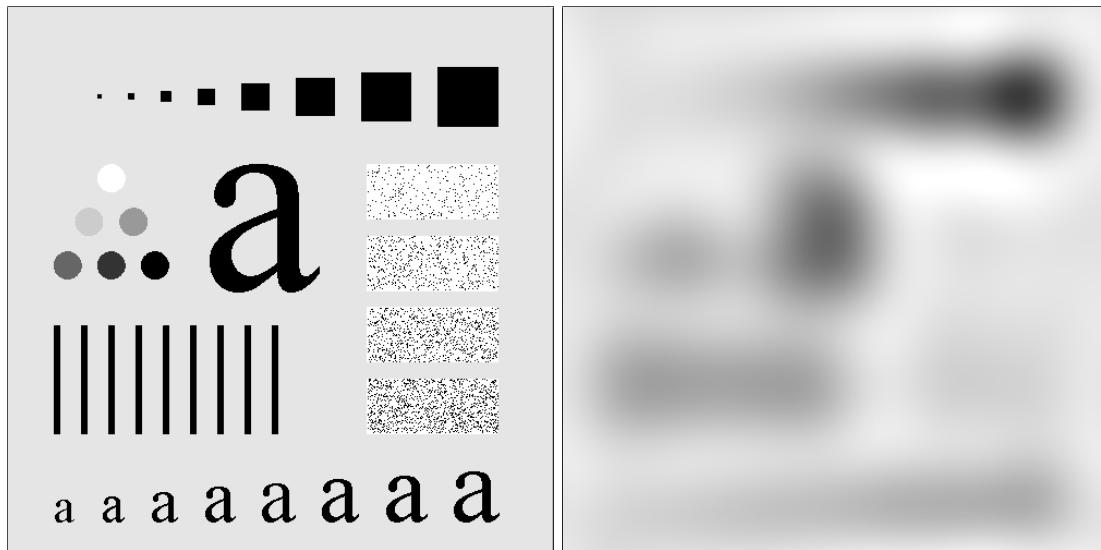


FIGURE C.18 – Original image

FIGURE C.19 – Butterworth low 5



FIGURE C.20 – Butterworth low 15

FIGURE C.21 – Butterworth low 30

High pass

```
python problem3.py -butterworth -d 5 -n 5 -high characters_test_pattern.tif
```

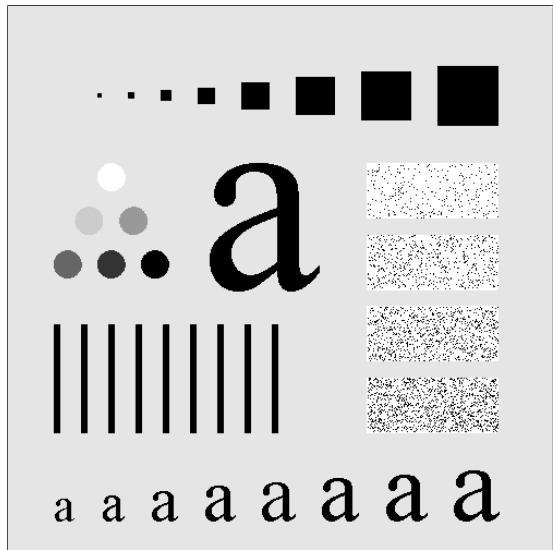


FIGURE C.22 – Original image



FIGURE C.23 – Butterworth high 5

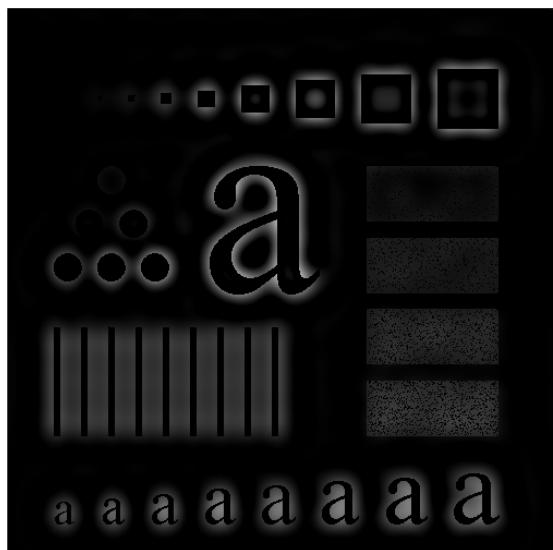


FIGURE C.24 – Butterworth high 15

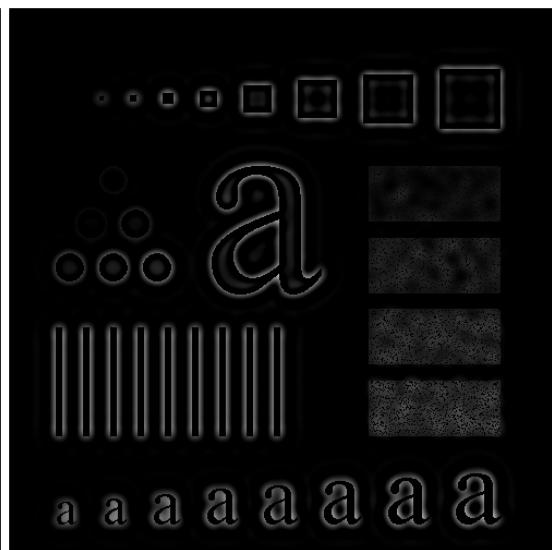


FIGURE C.25 – Butterworth high 30

C.3.5 Gaussian filter

Low pass

```
python problem3.py -gaussian -d 5 -low characters_test_pattern.tif
```

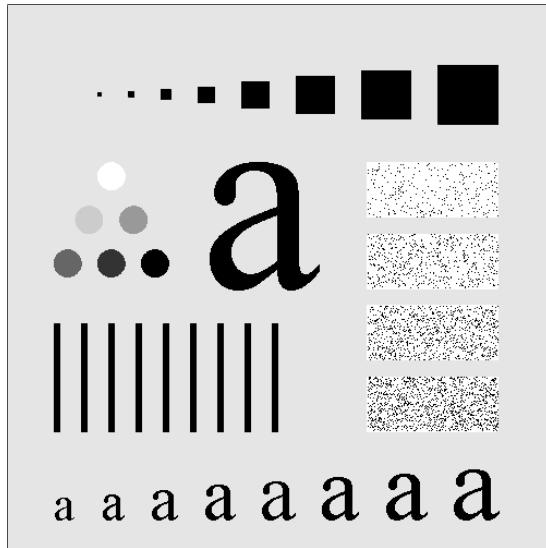


FIGURE C.26 – Original image

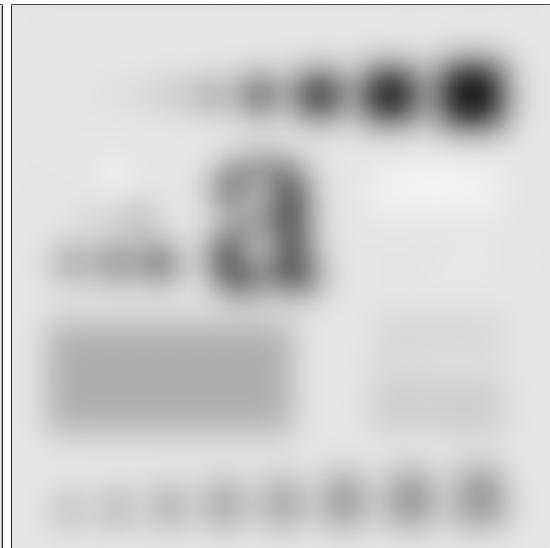


FIGURE C.27 – Gaussian low 5

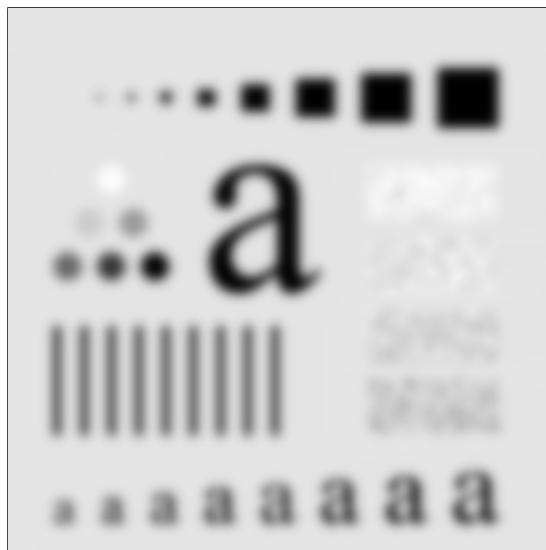


FIGURE C.28 – Gaussian low 15



FIGURE C.29 – Gaussian low 30

High pass

```
python problem3.py -gaussian -d 5 -high characters_test_pattern.tif
```

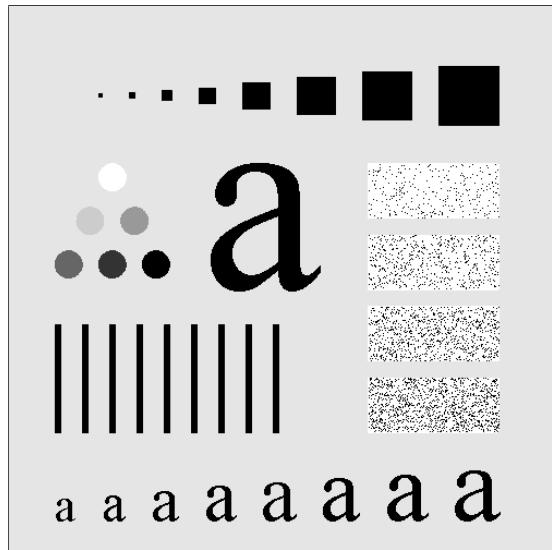


FIGURE C.30 – Original image

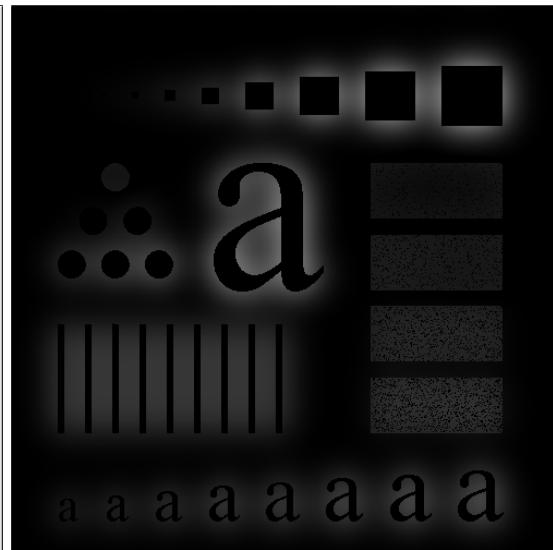


FIGURE C.31 – Gaussian high 5

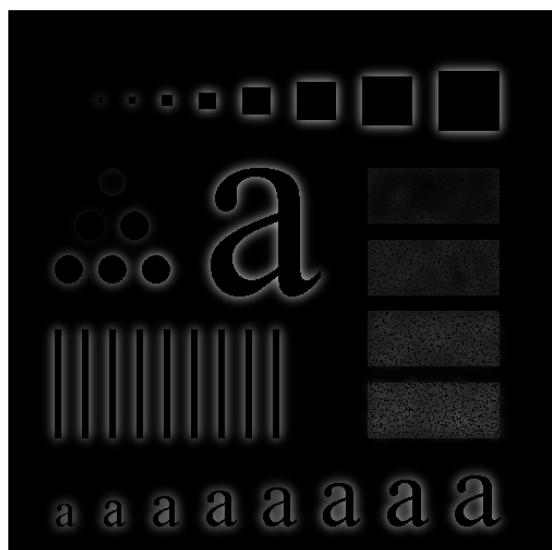


FIGURE C.32 – Gaussian high 15

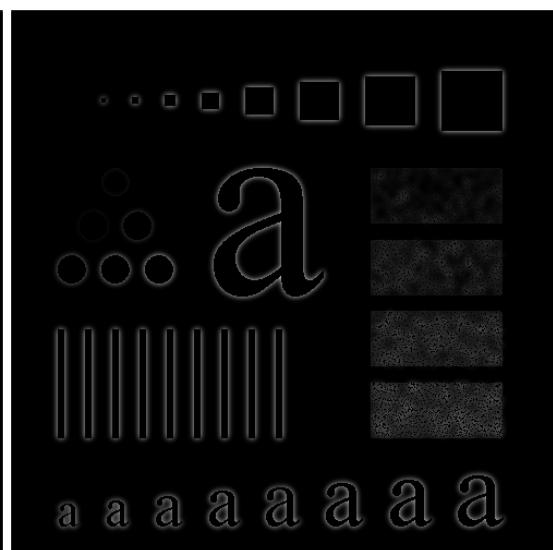


FIGURE C.33 – Gaussian high 30

D. Noise generation and noise reduction

D.1 Problem statement

In this problem, you are required to write a program to generate different types of random noises started from the Uniform noise and Gaussian noise.

And then add some of these noises to the circuit image and investigate the different mean filters and order statistics as the textbook did at pages 344-352.

D.2 Python implementation

```
Usage : problem4.py [-h] (-uniform | -gaussian) [-histogram]
[-arithmetic] [-geometric] [-harmonic] [-contraharmonic] [-q Q]
[-median] [-max] [-min] [-midpoint] [-alpha] [-d D]
image_path
```

Use **python problem4.py -h** to see the help.

D.3 Gaussian noise

D.3.1 Noise generation and histogram

```
python problem4.py -gaussian circuit.tif
```

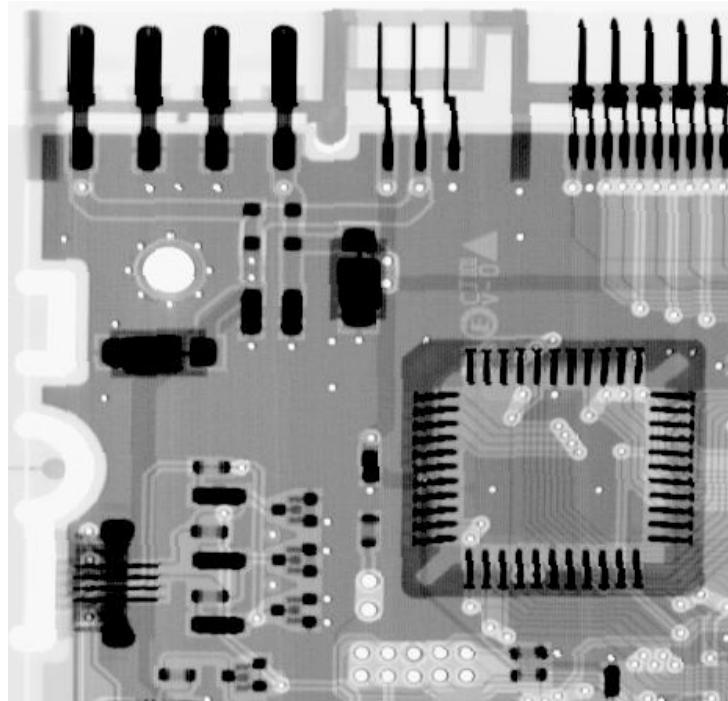


FIGURE D.1 – Original image

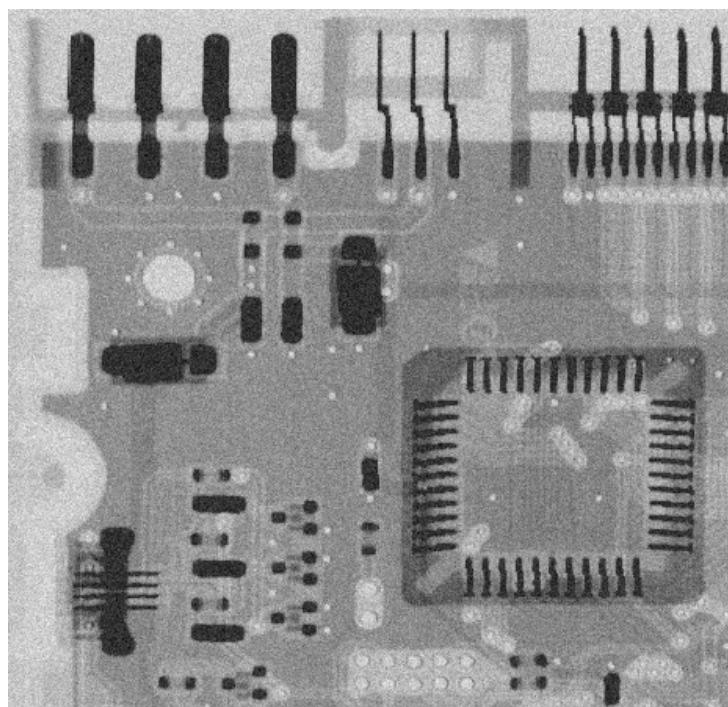


FIGURE D.2 – Image + Gaussian noise ($\mu = 0, \sigma = 20$)

```
python problem4.py -gaussian -histogram circuit.tif
```

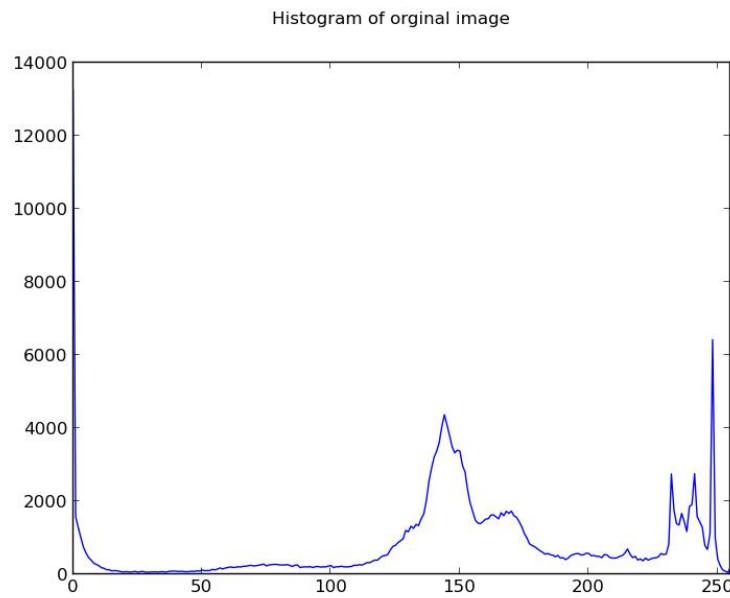


FIGURE D.3 – Histogram of original image

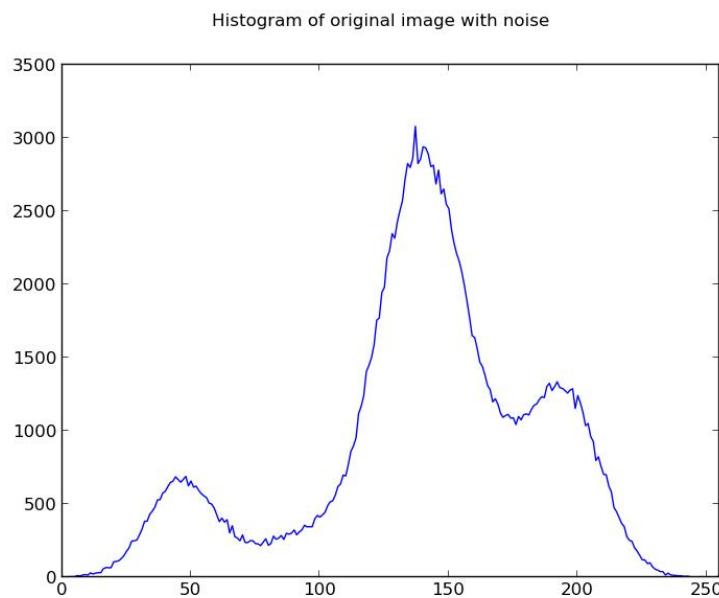


FIGURE D.4 – Histogram of image with Gaussian noise

D.3.2 Noise reduction

Mean filters

```
python problem4.py -gaussian -arithmetic circuit.tif  
python problem4.py -gaussian -geometric circuit.tif
```

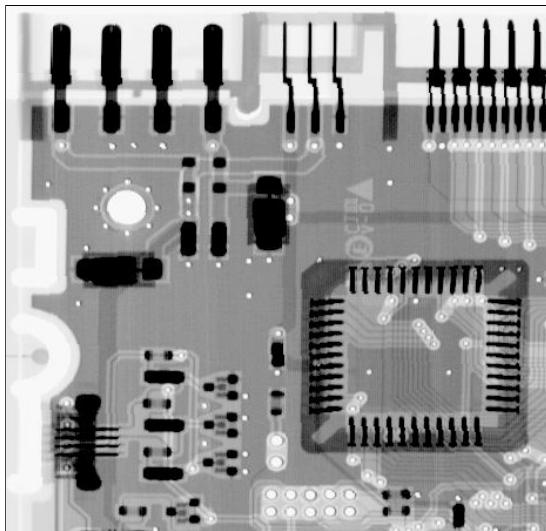


FIGURE D.5 – Original image

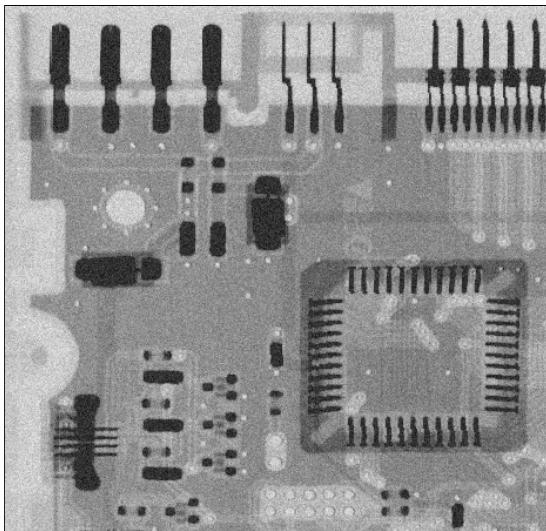


FIGURE D.6 – Image + Gaussian

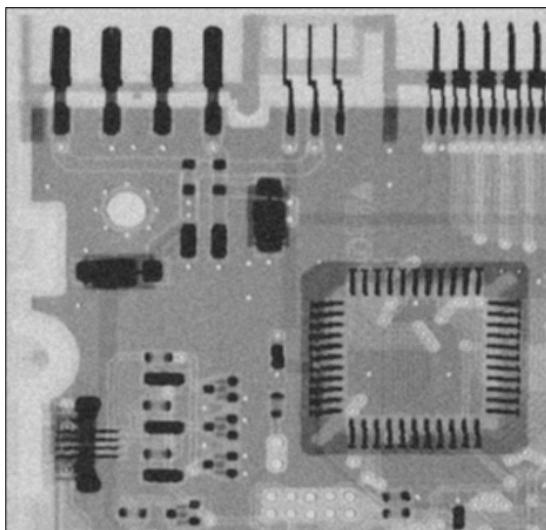


FIGURE D.7 – Arithmetic filter

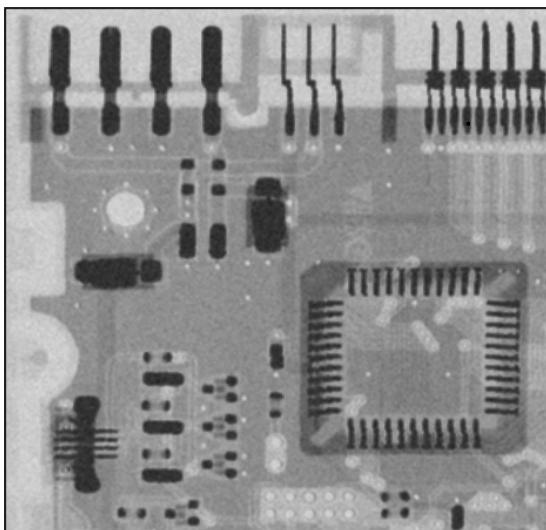


FIGURE D.8 – Geometric filter

```
python problem4.py -gaussian -harmonic circuit.tif
```

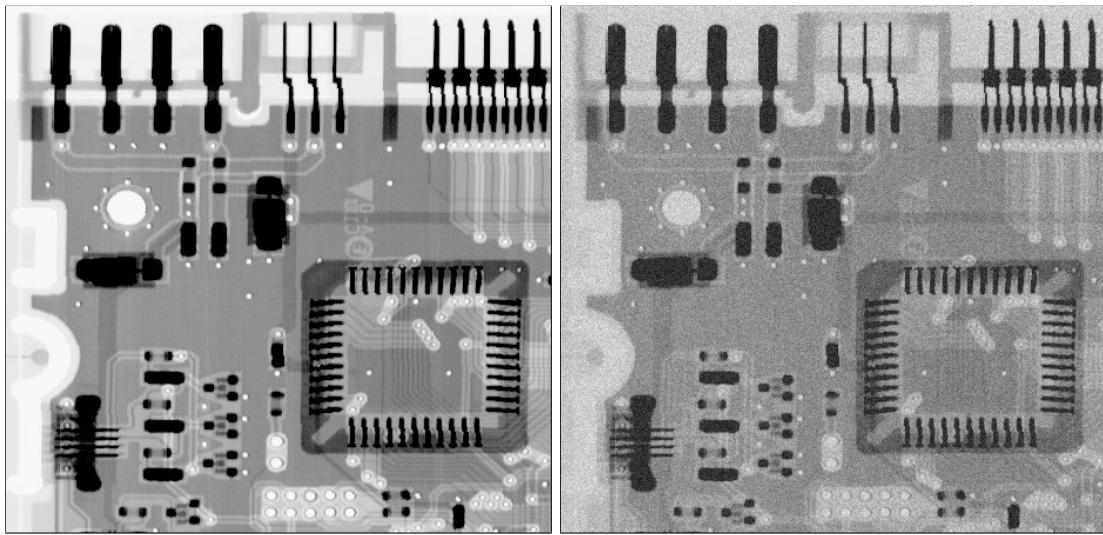


FIGURE D.9 – Original image

FIGURE D.10 – Image + Gaussian

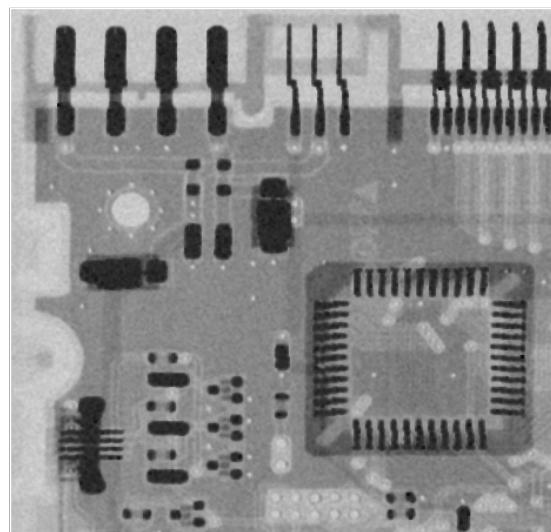


FIGURE D.11 – Harmonic filter

```
python problem4.py -gaussian -contraharmonic -q 1.5 circuit.tif  
python problem4.py -gaussian -contraharmonic -q -1.5 circuit.tif
```

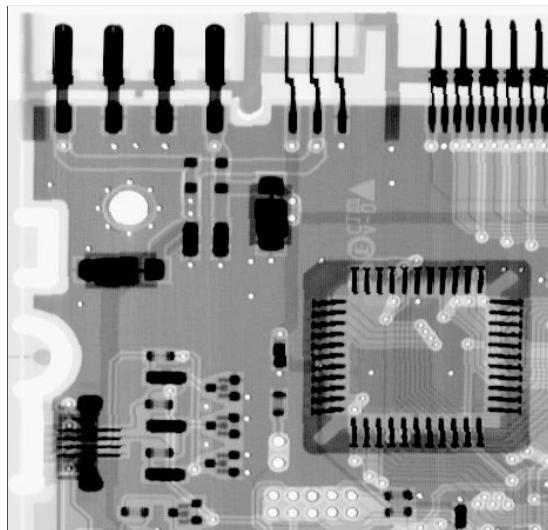


FIGURE D.12 – Original image

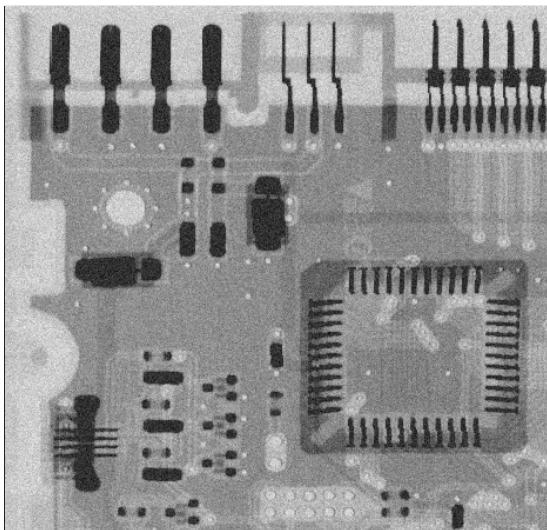


FIGURE D.13 – Image + Gaussian

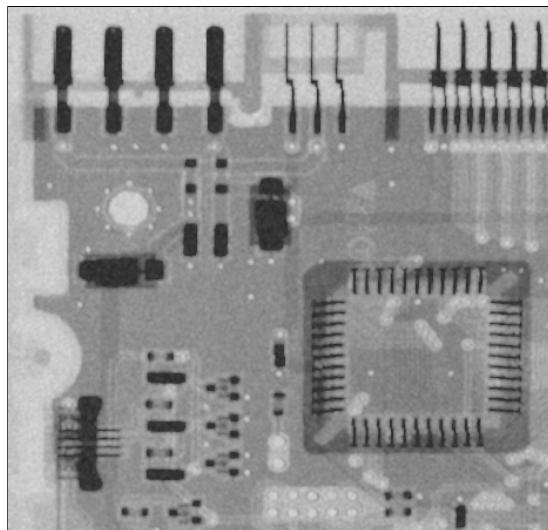


FIGURE D.14 – Contra-harmonic filter (Q=1.5)

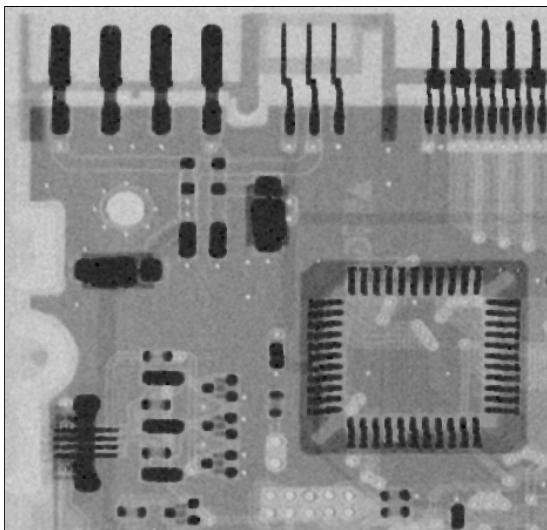


FIGURE D.15 – Contra-harmonic filter (Q=-1.5)

Order-Statistic filters

```
python problem4.py -gaussian -median circuit.tif
```

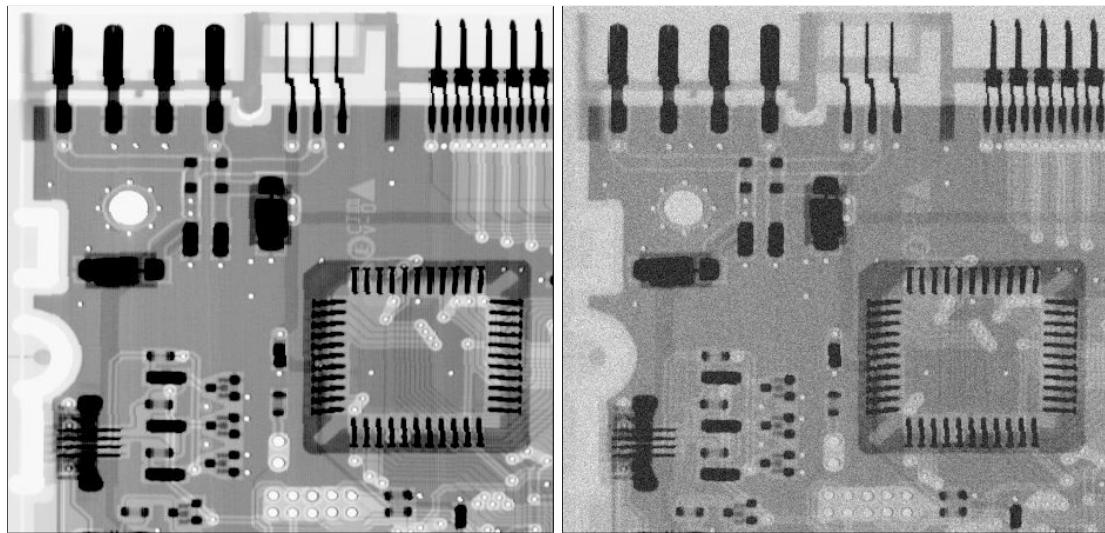


FIGURE D.16 – Original image

FIGURE D.17 – Image + Gaussian

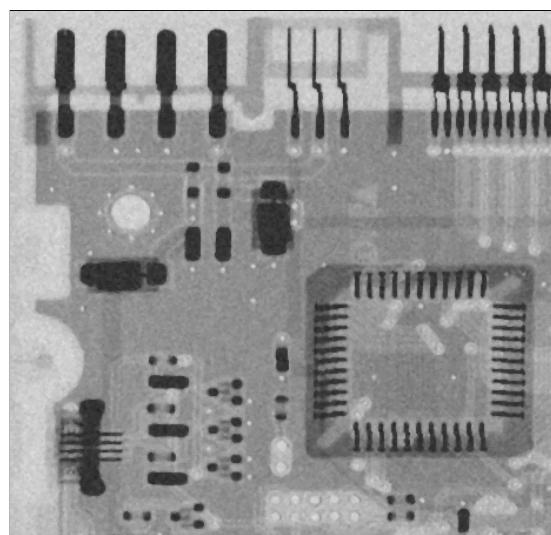


FIGURE D.18 – Median filter

```
python problem4.py -gaussian -max circuit.tif  
python problem4.py -gaussian -min circuit.tif
```

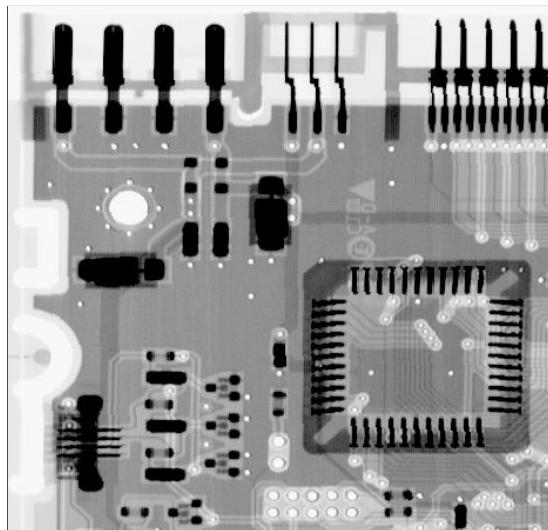


FIGURE D.19 – Original image

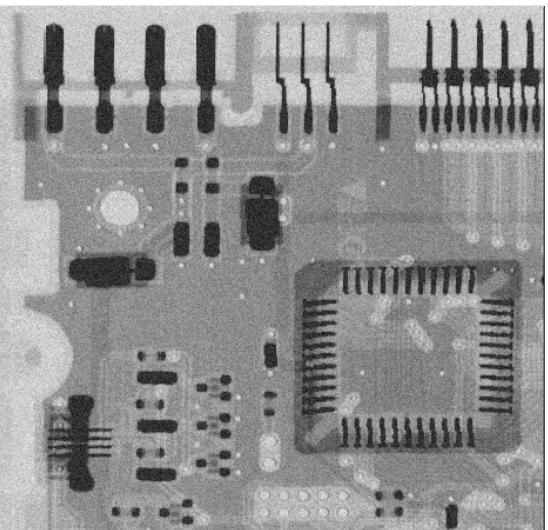


FIGURE D.20 – Image + Gaussian

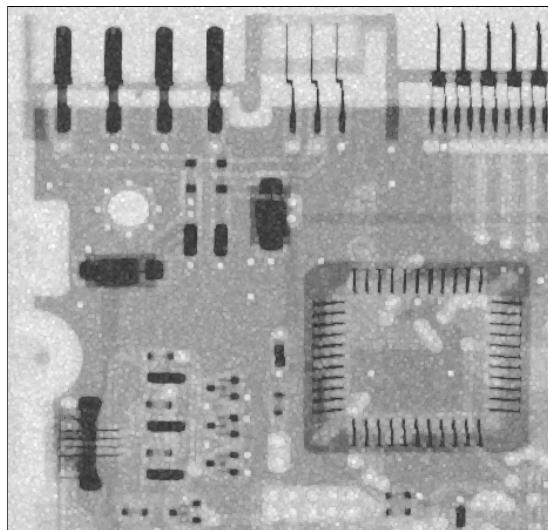


FIGURE D.21 – Max filter

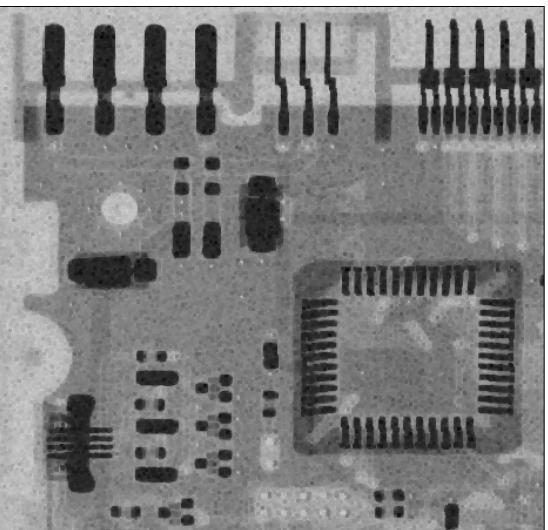


FIGURE D.22 – Min filter

```
python problem4.py -gaussian -midpoint circuit.tif
```

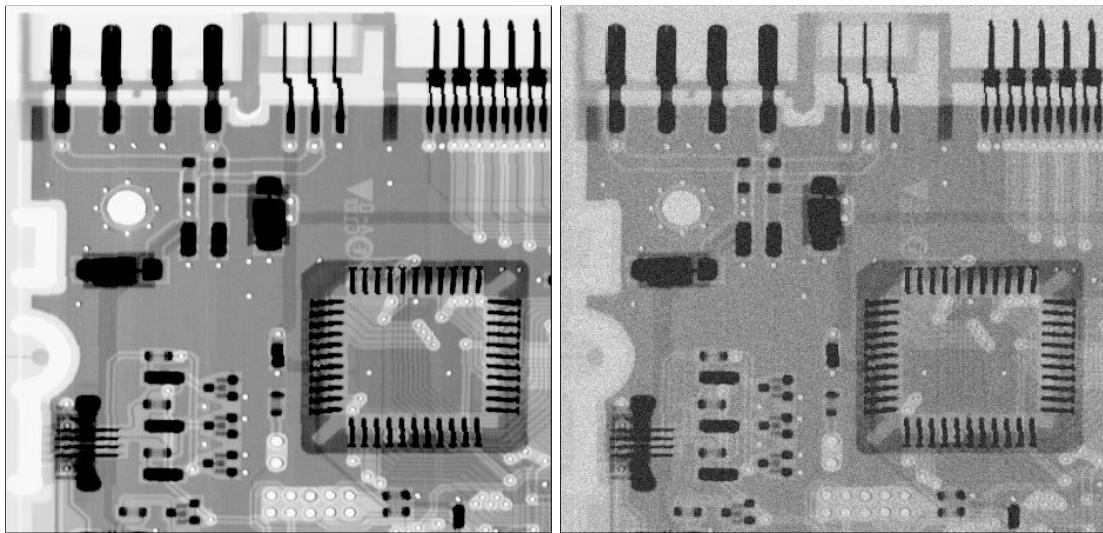


FIGURE D.23 – Original image

FIGURE D.24 – Image + Gaussian

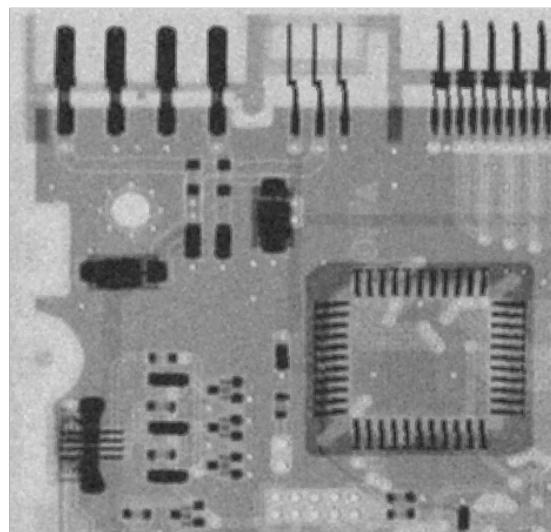


FIGURE D.25 – Midpoint filter

```
python problem4.py -gaussian -alpha -d 2 circuit.tif  
python problem4.py -gaussian -alpha -d 4 circuit.tif
```

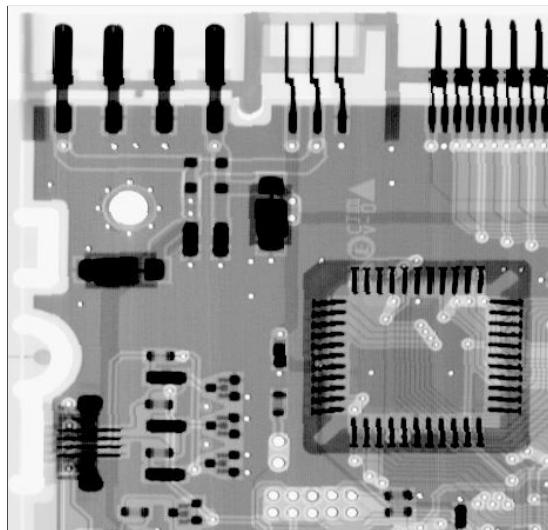


FIGURE D.26 – Original image

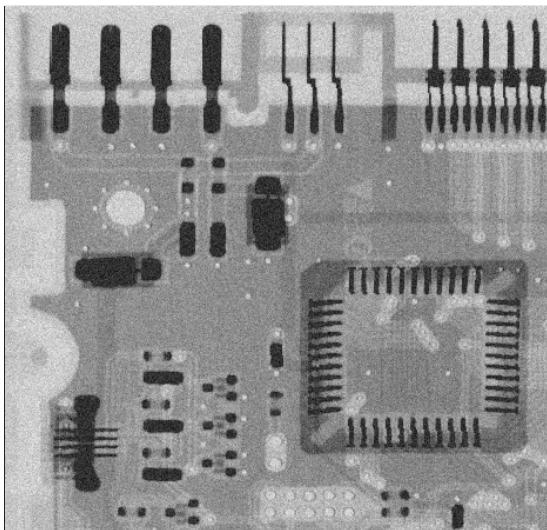


FIGURE D.27 – Image + Gaussian

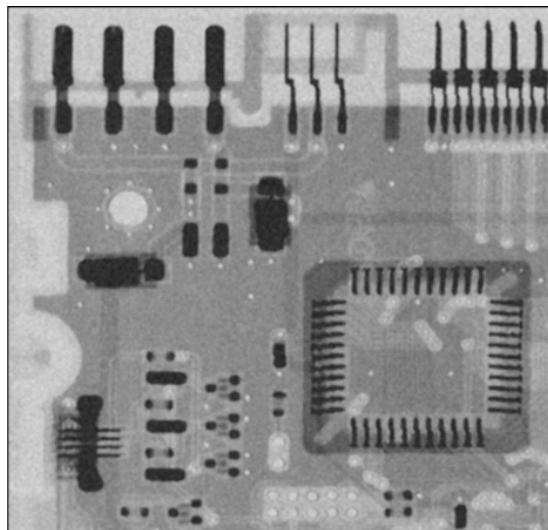


FIGURE D.28 – Alpha-trimmed filter
($d = 2$)

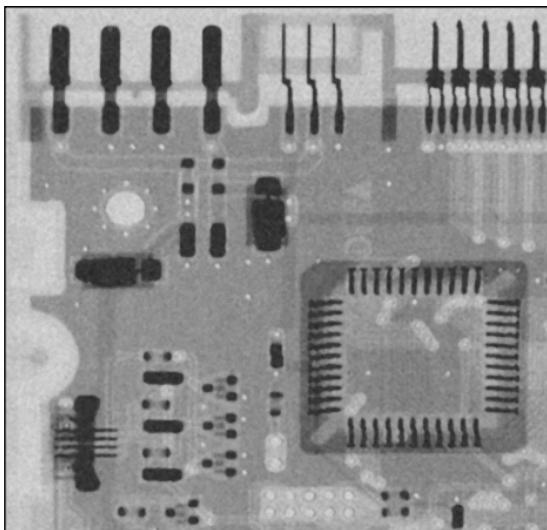


FIGURE D.29 – Alpha-trimmed filter
($d = 4$)

D.4 Uniform noise

D.4.1 Noise generation and histogram

```
python problem4.py -uniform circuit.tif
```

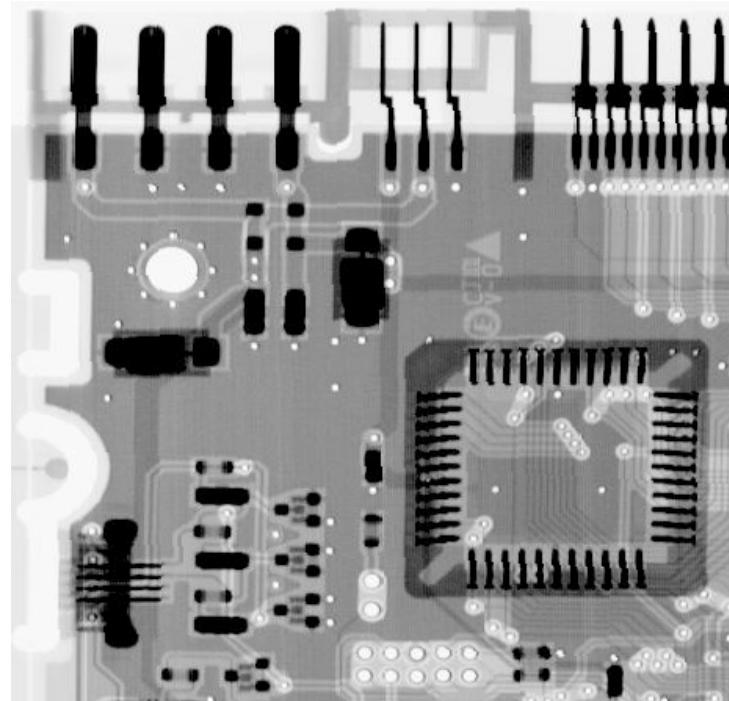


FIGURE D.30 – Original image

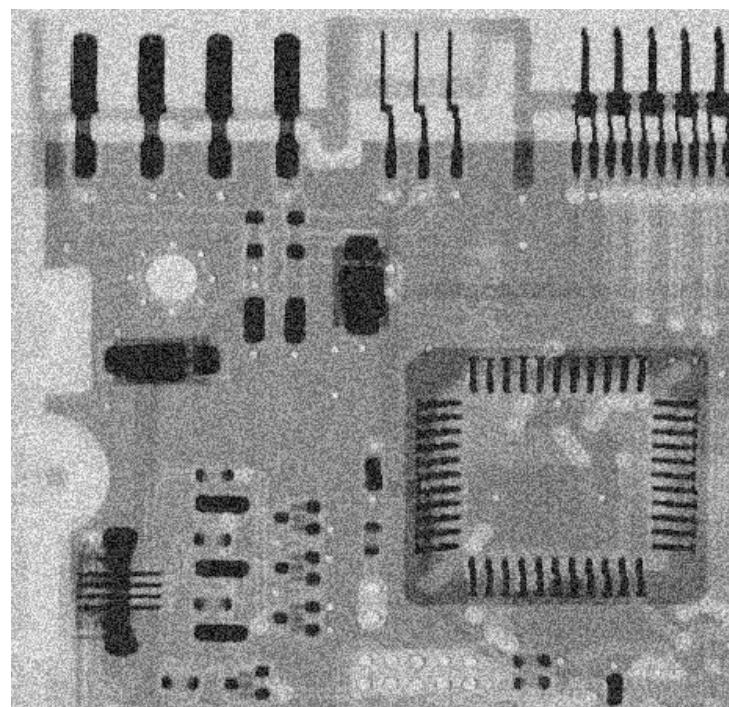


FIGURE D.31 – Image + Uniform noise (min = 0, max = 128)

```
python problem4.py -uniform -histogram circuit.tif
```

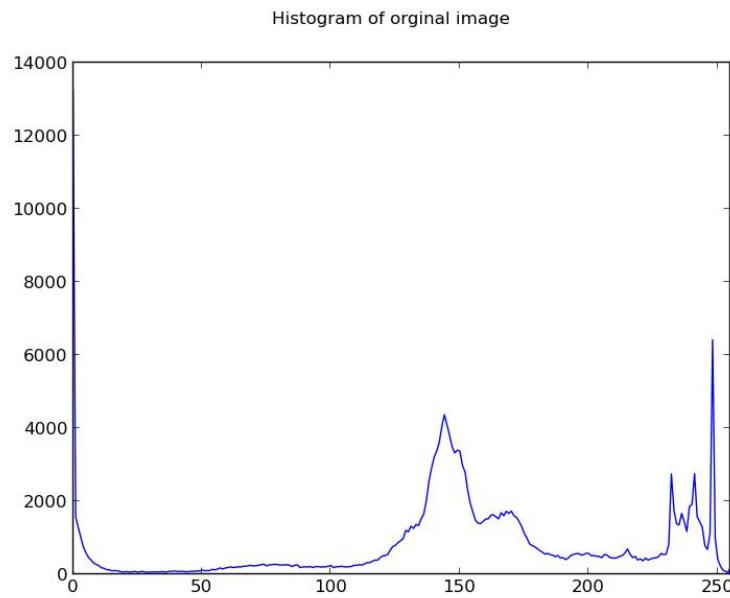


FIGURE D.32 – Histogram of original image

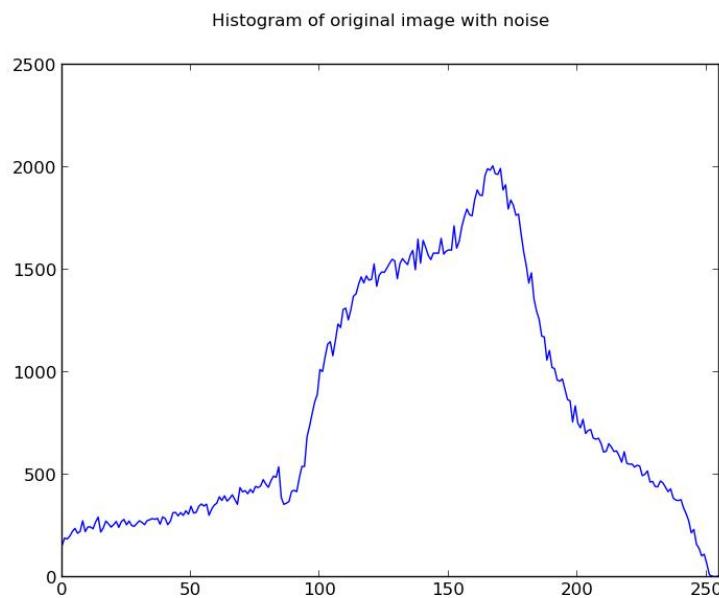


FIGURE D.33 – Histogram of image with Uniform noise

D.4.2 Noise reduction

Mean filters

```
python problem4.py -uniform -arithmetic circuit.tif  
python problem4.py -uniform -geometric circuit.tif
```

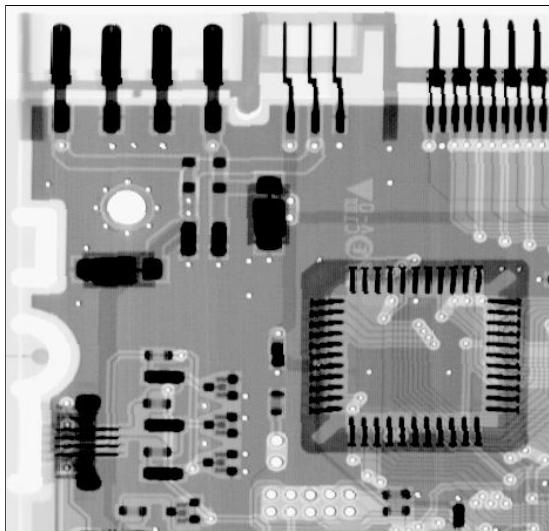


FIGURE D.34 – Original image

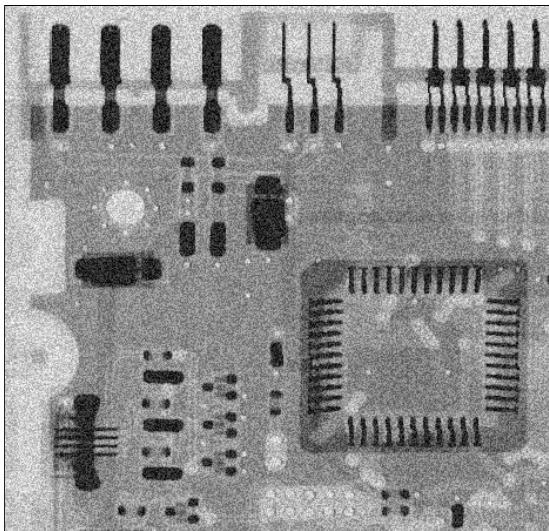


FIGURE D.35 – Image + Uniform

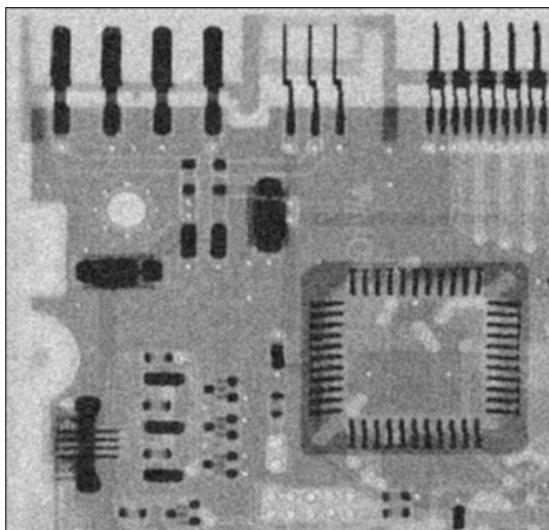


FIGURE D.36 – Arithmetic filter

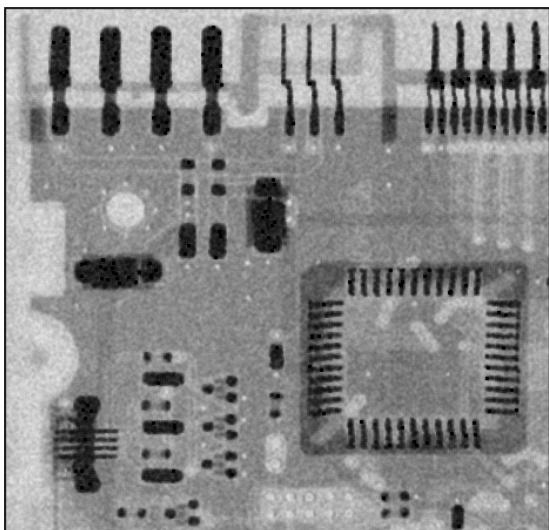


FIGURE D.37 – Geometric filter

```
python problem4.py -uniform -harmonic circuit.tif
```

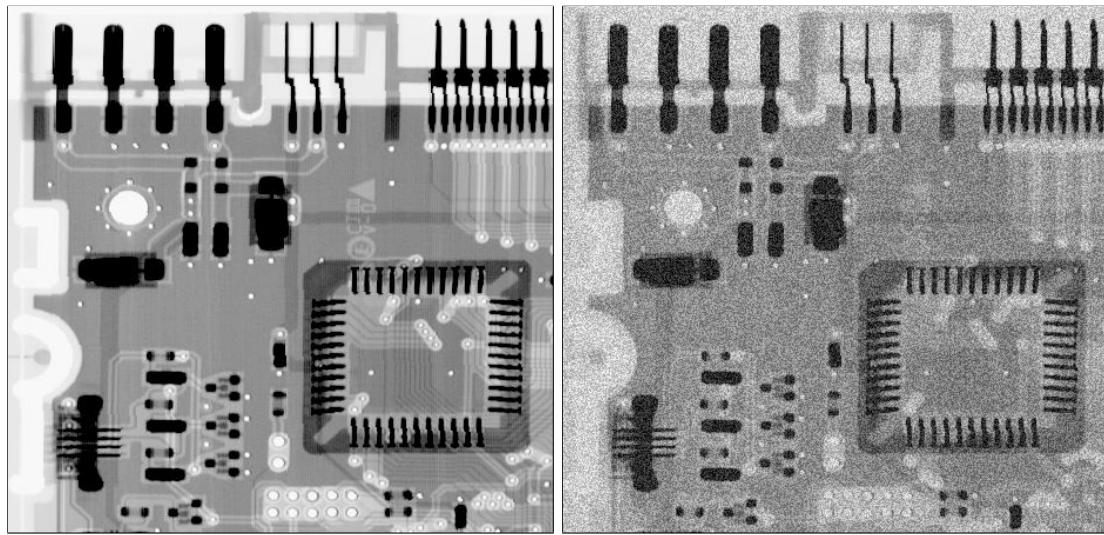


FIGURE D.38 – Original image

FIGURE D.39 – Image + Uniform

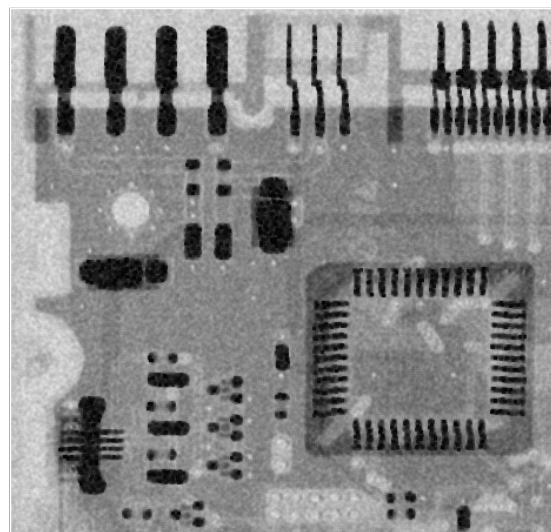


FIGURE D.40 – Harmonic filter

```
python problem4.py -uniform -contraharmonic -q 1.5 circuit.tif  
python problem4.py -uniform -contraharmonic -q -1.5 circuit.tif
```

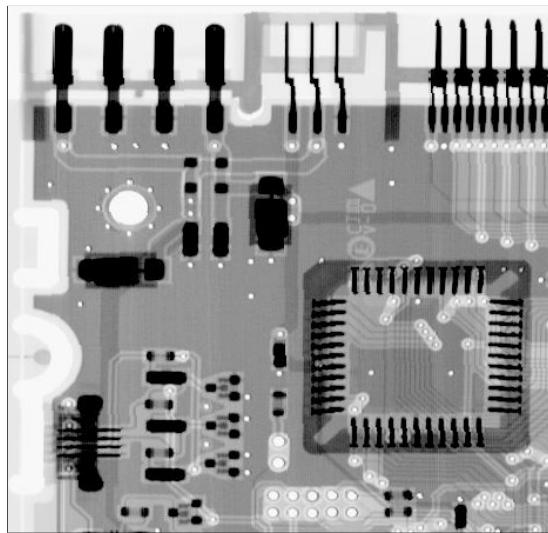


FIGURE D.41 – Original image

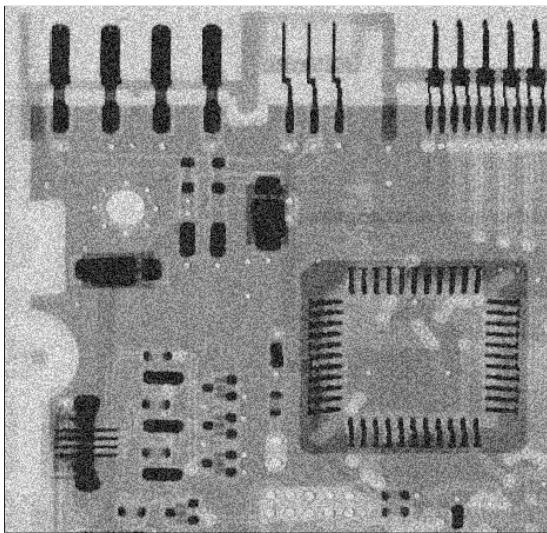


FIGURE D.42 – Image + Uniform

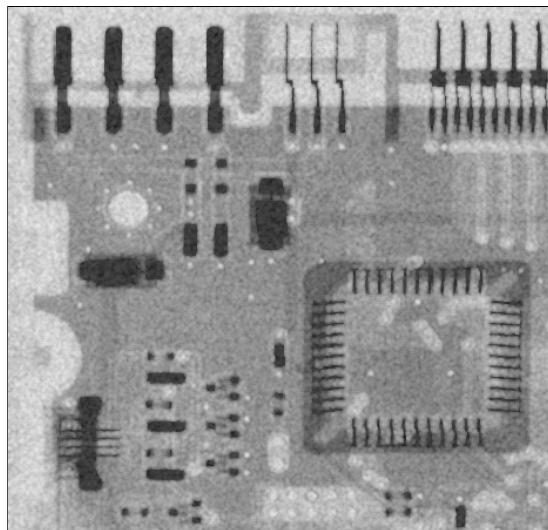


FIGURE D.43 – Contra-harmonic filter (Q=1.5)

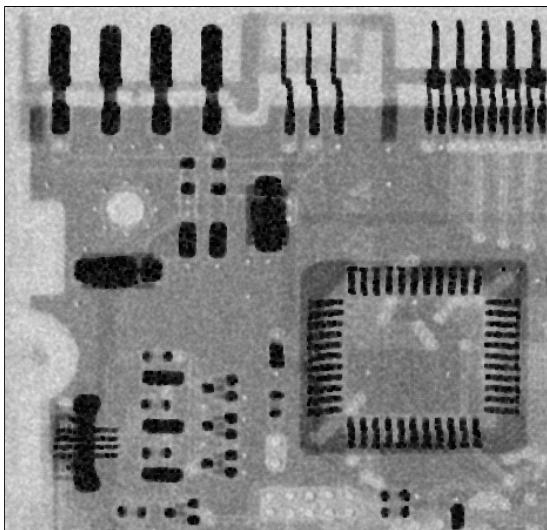


FIGURE D.44 – Contra-harmonic filter (Q=-1.5)

Order-Statistic filters

```
python problem4.py -uniform -median circuit.tif
```

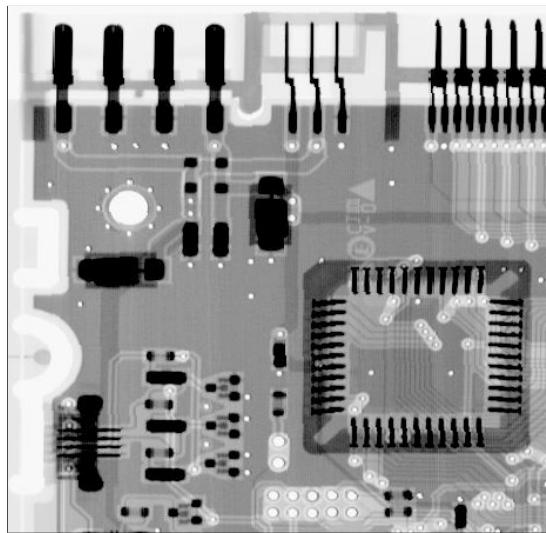


FIGURE D.45 – Original image

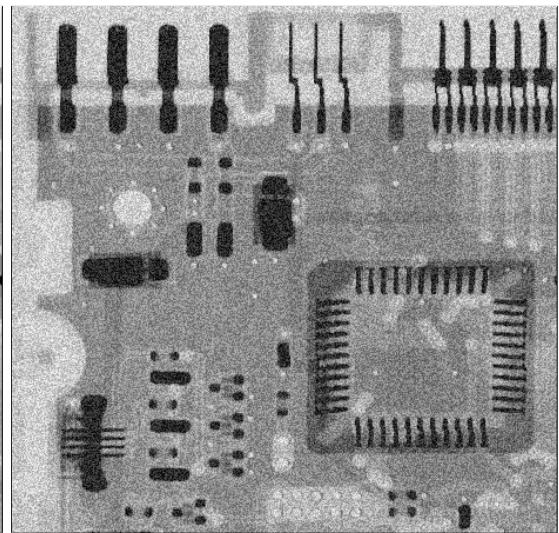


FIGURE D.46 – Image + Uniform

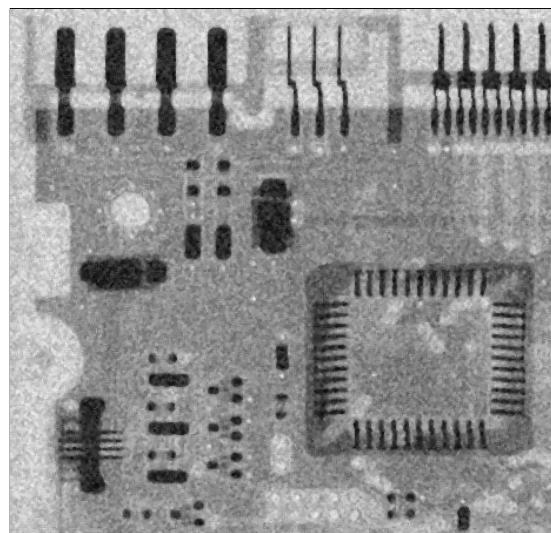


FIGURE D.47 – Median filter

```
python problem4.py -uniform -max circuit.tif  
python problem4.py -uniform -min circuit.tif
```

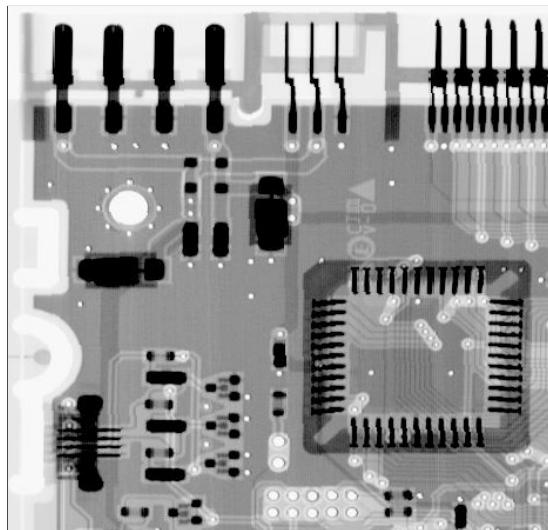


FIGURE D.48 – Original image

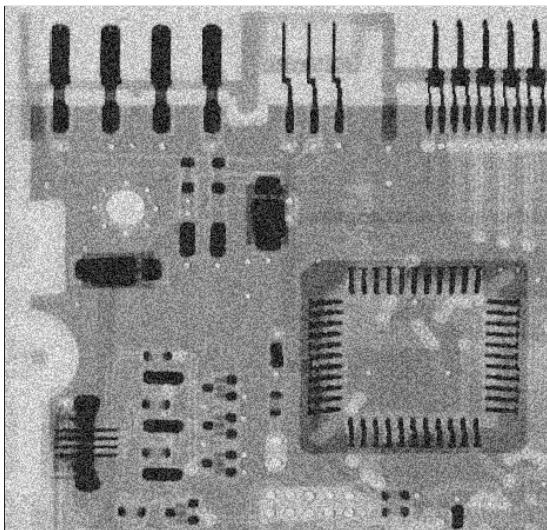


FIGURE D.49 – Image + Uniform

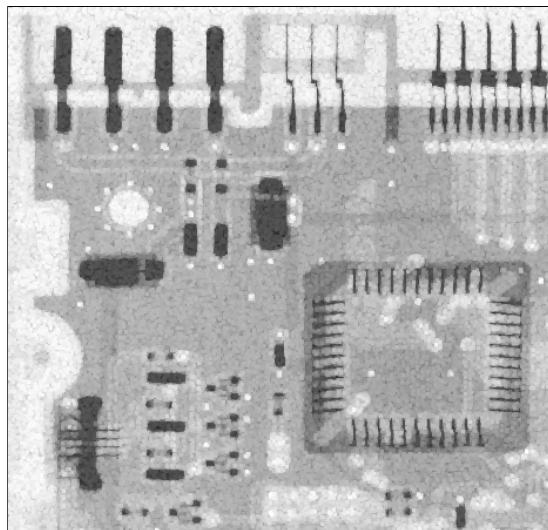


FIGURE D.50 – Max filter

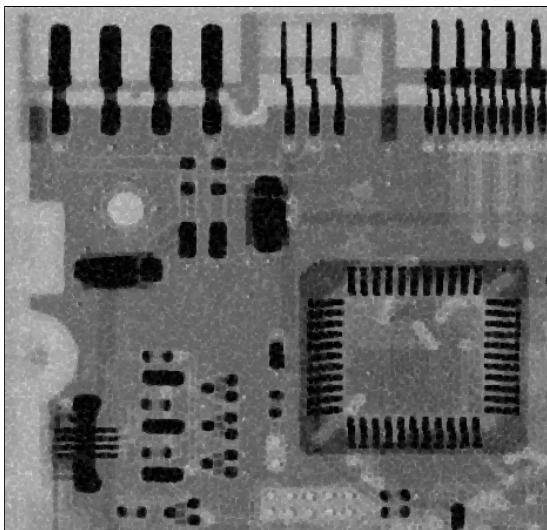


FIGURE D.51 – Min filter

```
python problem4.py -uniform -midpoint circuit.tif
```

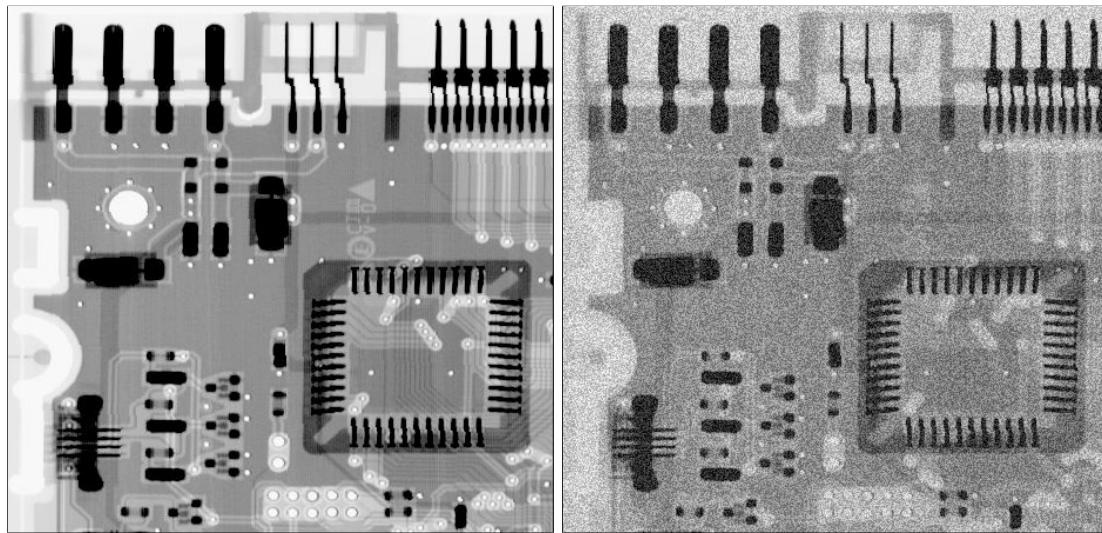


FIGURE D.52 – Original image

FIGURE D.53 – Image + Uniform

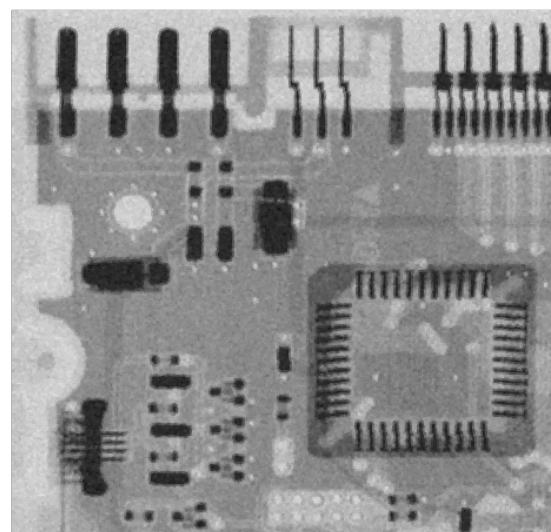


FIGURE D.54 – Midpoint filter

```
python problem4.py -uniform -alpha -d 2 circuit.tif  
python problem4.py -uniform -alpha -d 4 circuit.tif
```

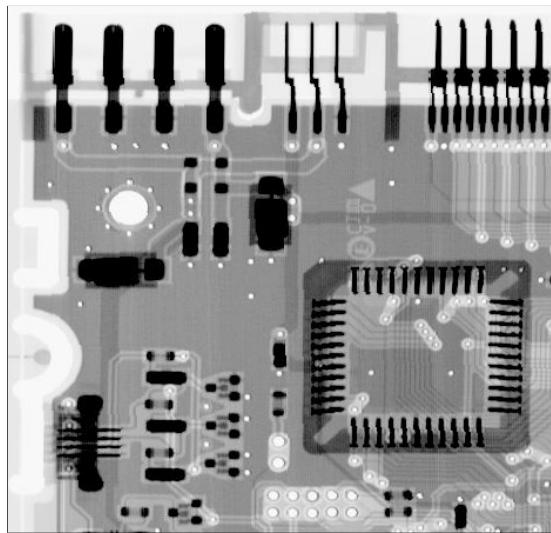


FIGURE D.55 – Original image

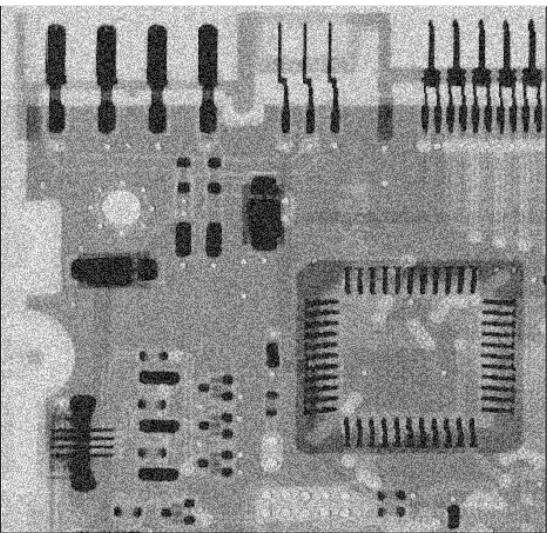


FIGURE D.56 – Image + Uniform

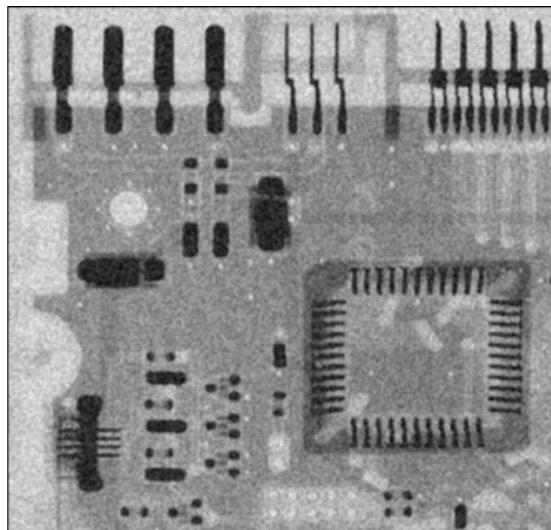


FIGURE D.57 – Alpha-trimmed filter
($d = 2$)

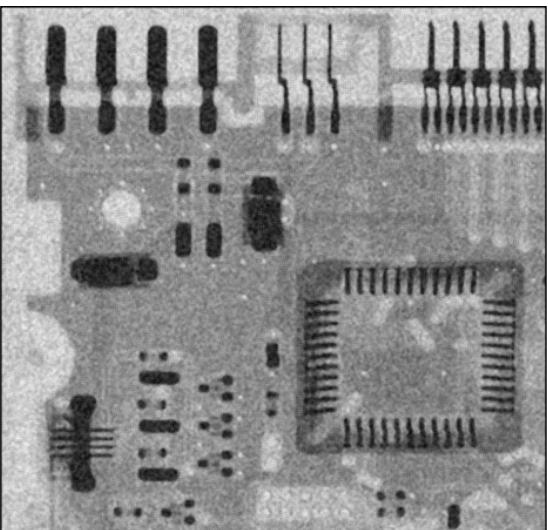


FIGURE D.58 – Alpha-trimmed filter
($d = 4$)

E. Image restoration

E.1 Problem statement

Suppose a blurring degradation function as

$$H(u, v) = T \times \text{sinc}(ua + vb) \times e^{-j\pi(ua+vb)} \quad (\text{E.1})$$

- (a) Implement a blurring filter using the equation above.
- (b) Blur the test image book_cover.jpg using parameters $a = b = 0.1$ and $T = 1$.
- (c) Add Gaussian noise of 0 mean and variance of 650 to the blurred image.
- (d) Restore the blurred image and the blurred noisy image using the inverse filter, Wiener deconvolution filter and the parametric Wiener filter, respectively.
- (e) Add Gaussian noise of 0 and different variances to the blurred image and repeat (d), investigate the performance of the Wiener deconvolution filter.

E.2 Python implementation

```
Usage : problem5.py [-h] [-blurred BLURRED] [-noise] [-s S] [-inv]
[-wiener] [-a A] [-b B] [-T T] [-K K] image_path
```

Use **python problem5.py -h** to see the help.

E.3 Blurring and noising

`python problem5.py book_cover.jpg`

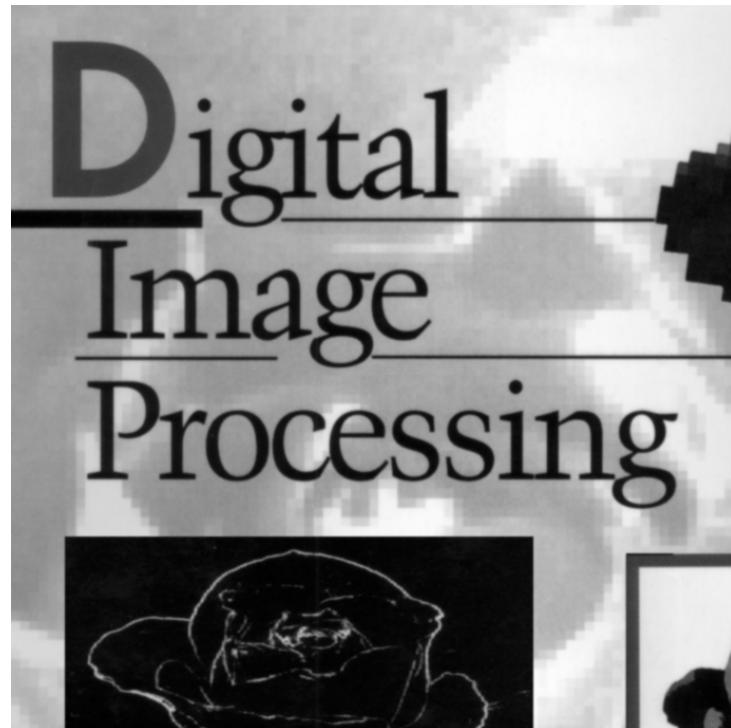


FIGURE E.1 – Original image



FIGURE E.2 – Blurred image ($a = b = 0.1; T = 1$)

```
python problem5.py book_cover.jpg --noise -s 650
```



FIGURE E.3 – Blur + Gaussian noise ($\mu = 0, \sigma^2 = 650$)

E.4 Restoration

E.4.1 Inverse filter

```
python problem5.py -inv book_cover.jpg  
python problem5.py -noise -s 650 -inv book_cover.jpg
```

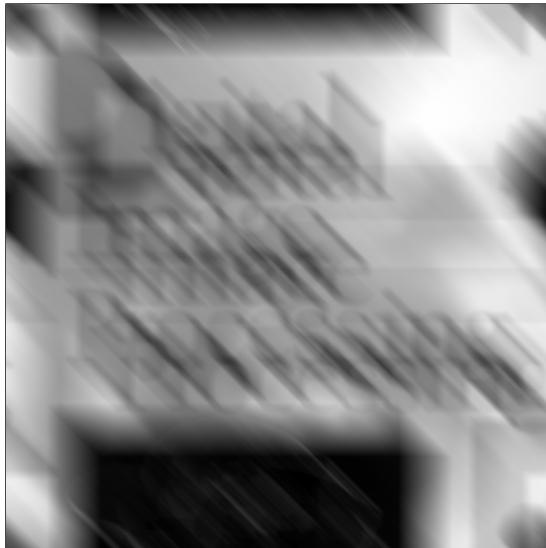


FIGURE E.4 – Blurred image

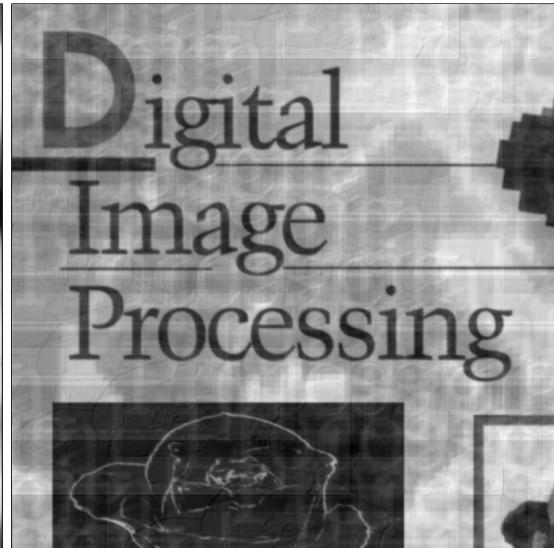


FIGURE E.5 – Inverse blurred image



FIGURE E.6 – Blur + noise

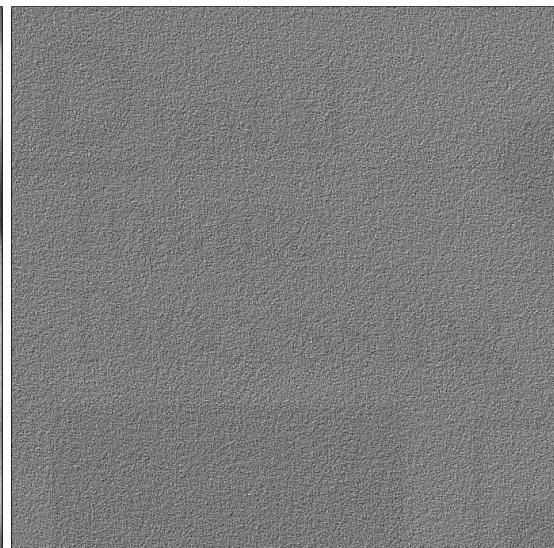


FIGURE E.7 – Inverse 'blur + noise'

The inverse filter is really affected by the noise because it ignores it (does not inverse it).

E.4.2 Wiener filter

```
python problem5.py -wiener book_cover.jpg  
python problem5.py -noise -s 650 -wiener book_cover.jpg
```



FIGURE E.8 – Blurred image

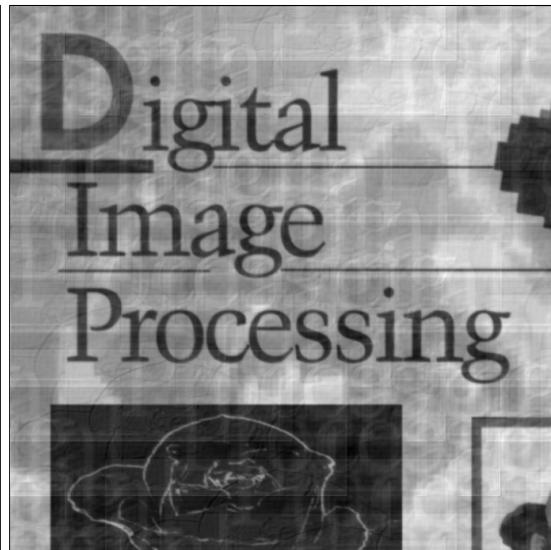


FIGURE E.9 – Wiener blurred image



FIGURE E.10 – Blur + noise



FIGURE E.11 – Wiener 'blur + noise'

The Wiener filter on the blurred image without noise is equivalent to the inverse filter (power spectrum of noise equals to 0).

The Wiener filter is more efficient to remove the noise as it takes it into consideration during the restoration process.

E.4.3 Experimentations

Gaussian noise ($\mu = 0, \sigma^2 = 300$)

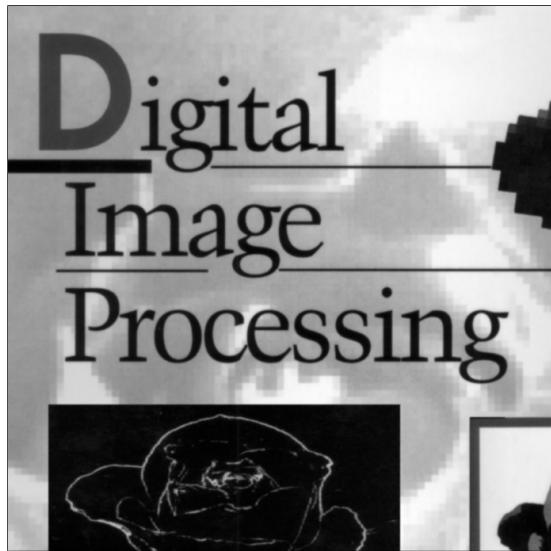


FIGURE E.12 – Original image



FIGURE E.13 – Blur + noise

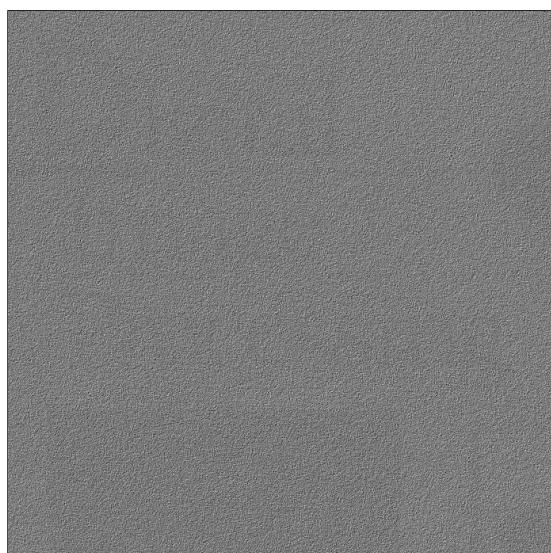


FIGURE E.14 – Inverse 'blur + noise'



FIGURE E.15 – Wiener 'blur + noise'

Gaussian noise ($\mu = 0, \sigma^2 = 100$)

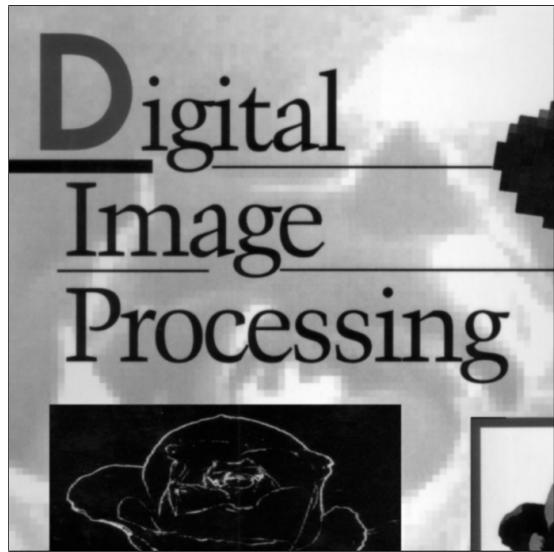


FIGURE E.16 – Original image



FIGURE E.17 – Blur + noise

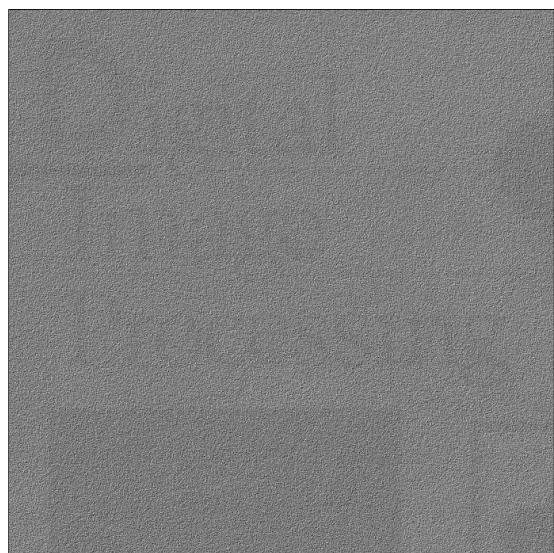


FIGURE E.18 – Inverse 'blur + noise'



FIGURE E.19 – Wiener 'blur + noise'

Gaussian noise ($\mu = 0, \sigma^2 = 30$)

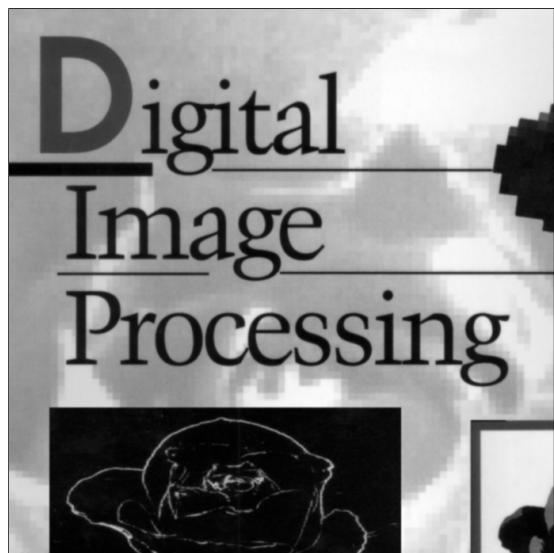


FIGURE E.20 – Original image



FIGURE E.21 – Blur + noise

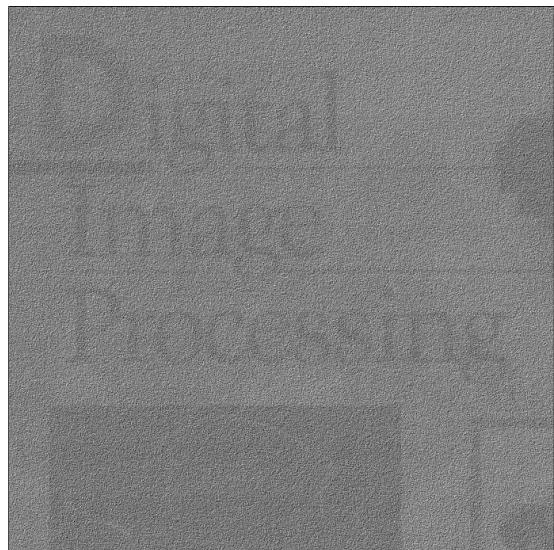


FIGURE E.22 – Inverse 'blur + noise'



FIGURE E.23 – Wiener 'blur + noise'

Wiener performance

FIGURE E.24 – Wiener 'blur + noise ($\sigma^2 = 30$)'



FIGURE E.25 – Wiener 'blur + noise ($\sigma^2 = 100$)'



FIGURE E.26 – Inverse 'blur + noise ($\sigma^2 = 300$)'



FIGURE E.27 – Wiener 'blur + noise ($\sigma^2 = 650$)'

F. Geometric transforms

F.1 Problem statement

Develop a geometric transform program that will rotate, translate, and scale an image by specified amounts, using the nearest neighbor and bilinear interpolation methods, respectively.

F.2 Python implementation

```
Usage : problem6.py [-h] -i INPUT  
(-t TRANSLATE [TRANSLATE ...] | -r ROTATE | -s SCALE) (-nearest | -bilinear)  
[-ntruncate] [-debug]
```

Use `python problem6.py -h` to see the help.

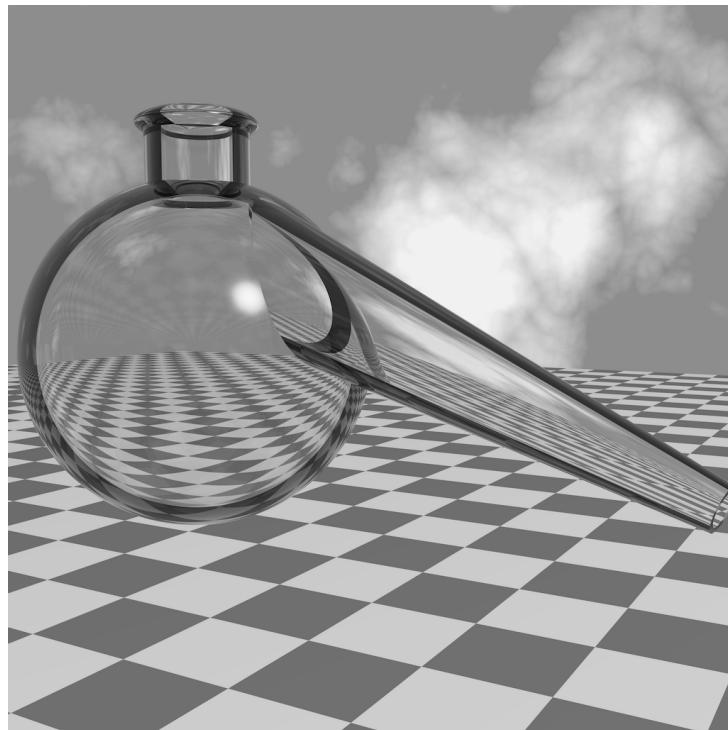


FIGURE F.1 – Original image

F.3 Translation

`python problem6.py -i ray_trace_bottle.tif -bilinear -t tx ty`, where (tx, ty) is the 2D translation vector.

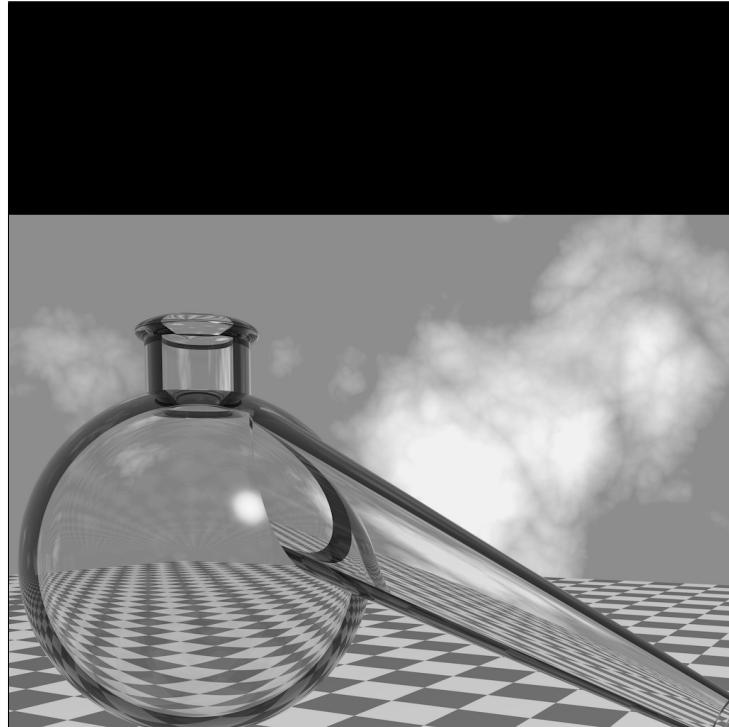


FIGURE F.2 – Translation $(0, 300)$ nearest

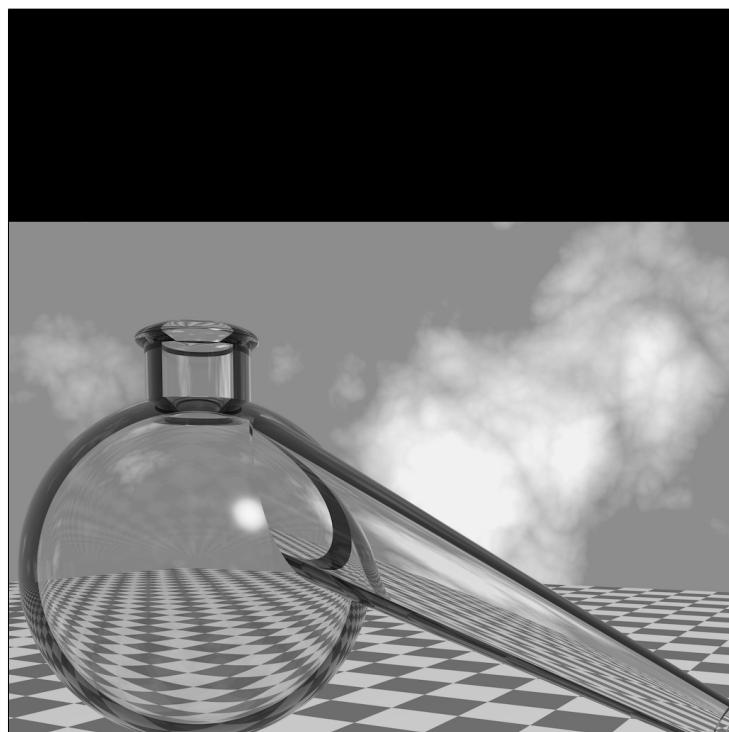


FIGURE F.3 – Translate $(0, 300)$ bilinear

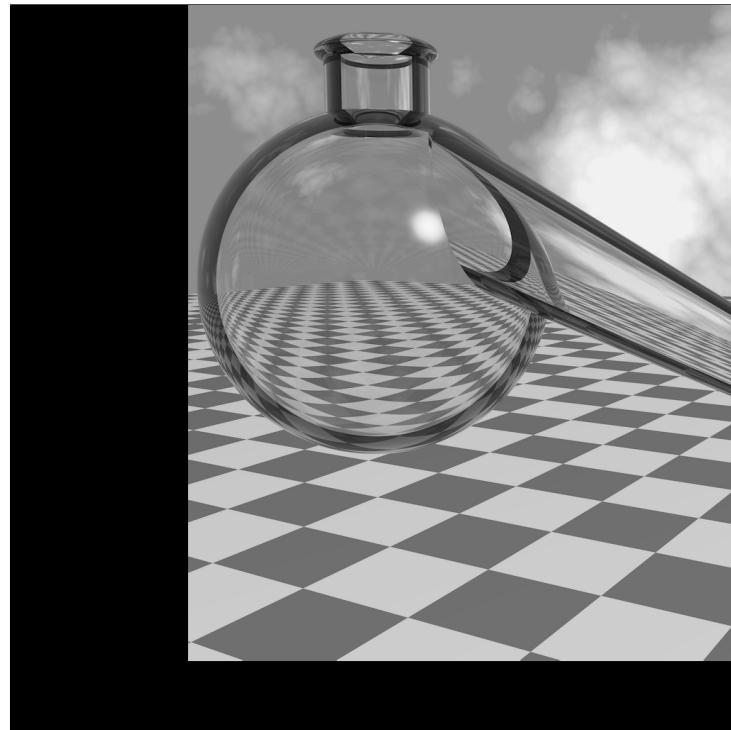


FIGURE F.4 – Translation (250, -100) nearest

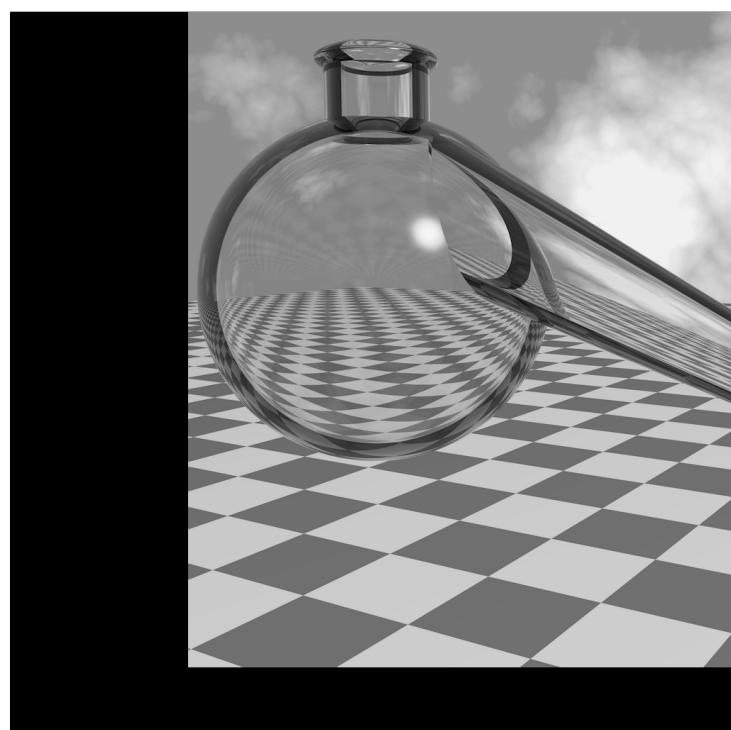


FIGURE F.5 – Translate (250, -100) bilinear

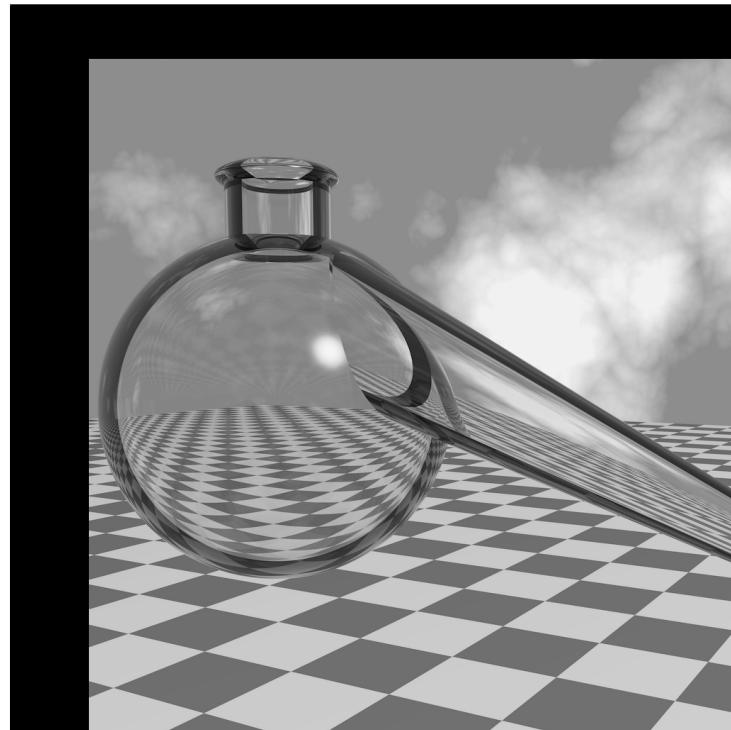


FIGURE F.6 – Translation (109.8, 75.5) nearest

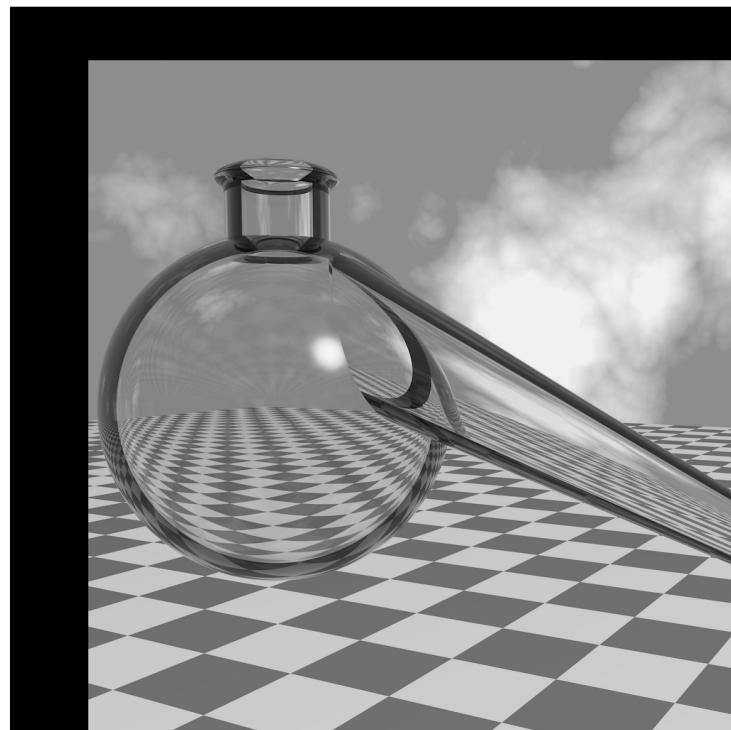


FIGURE F.7 – Translation (109.8, 75.5) bilinear

F.4 Rotation

`python problem6.py -i ray_trace_bottle.tif -nearest -r theta`, where theta is the angle of rotation.

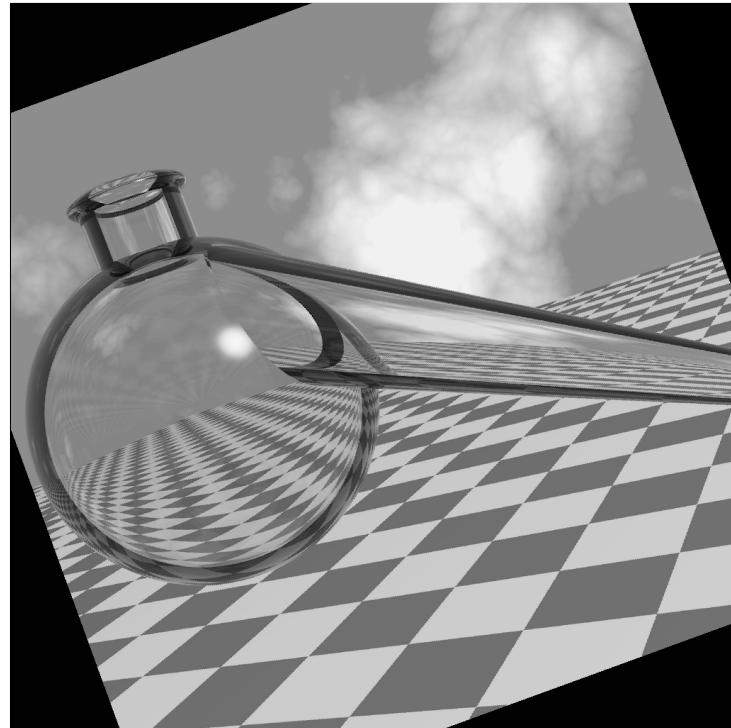


FIGURE F.8 – Rotate 20° nearest

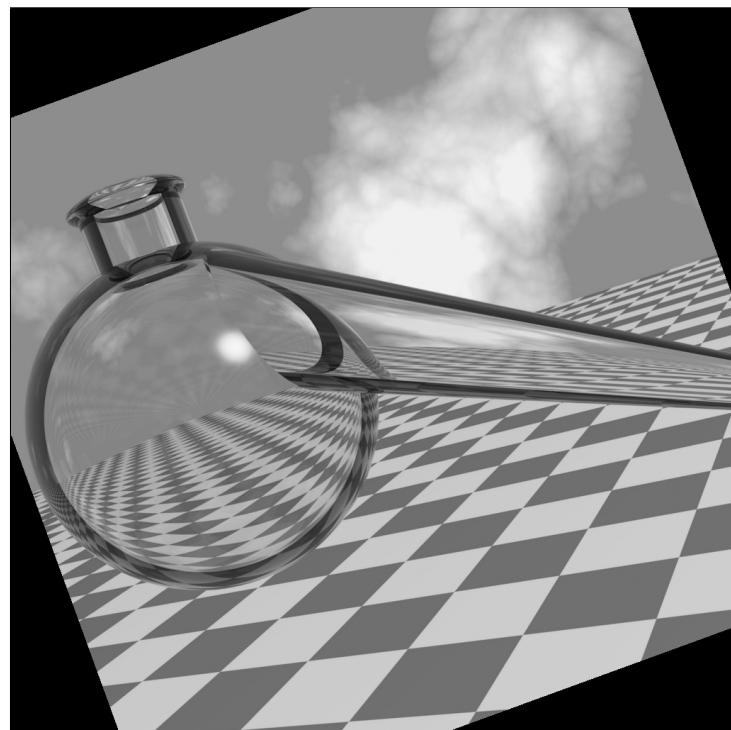


FIGURE F.9 – Rotate 20° bilinear

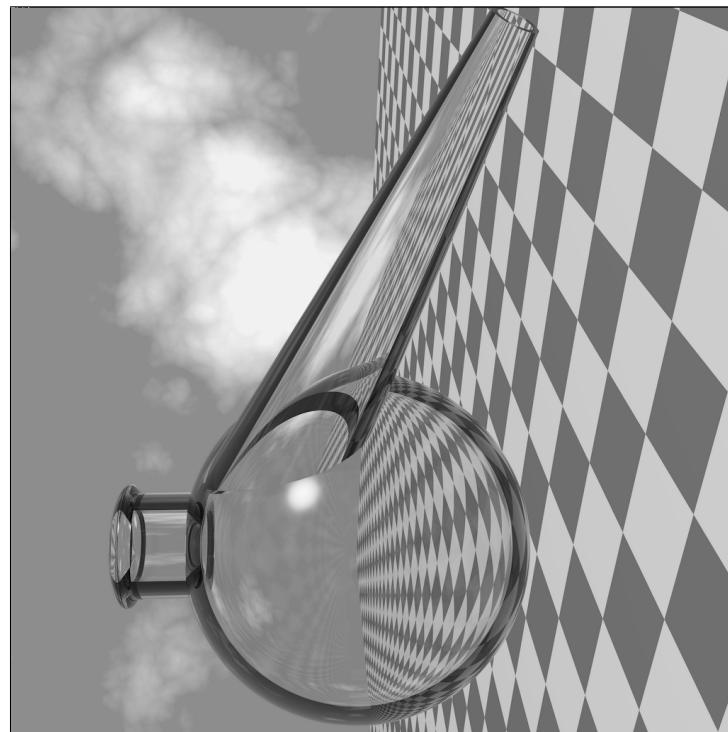


FIGURE F.10 – Rotate 90° nearest

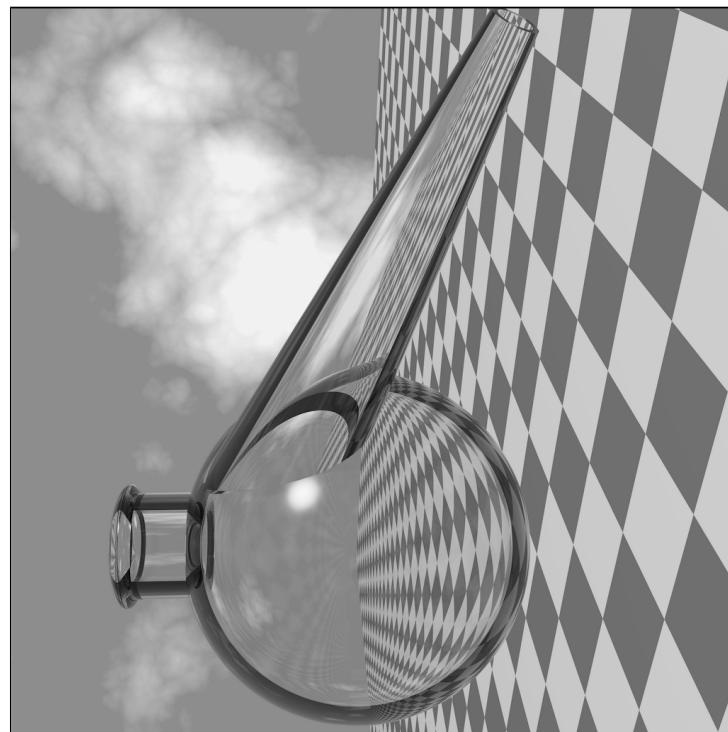


FIGURE F.11 – Rotate 90° bilinear

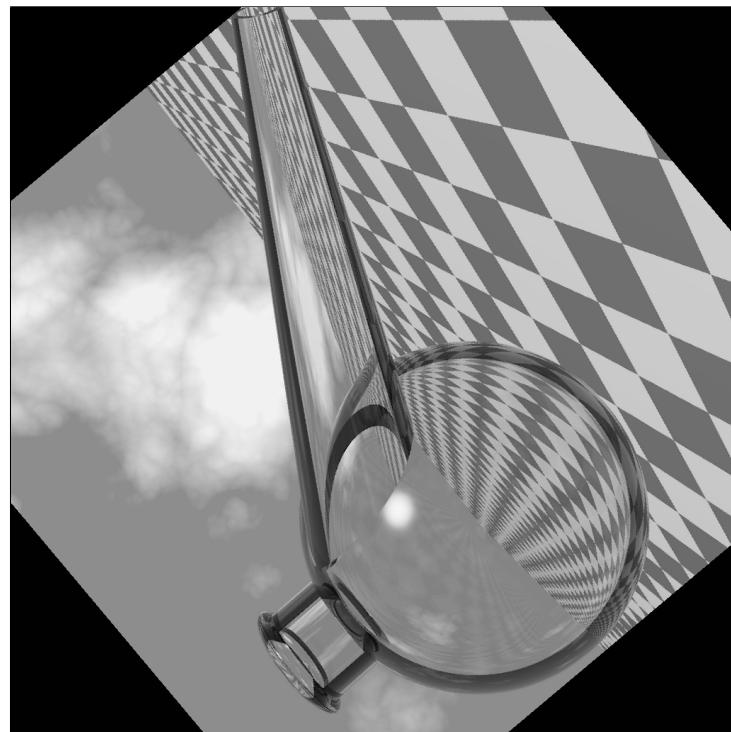


FIGURE F.12 – Rotate 130° nearest

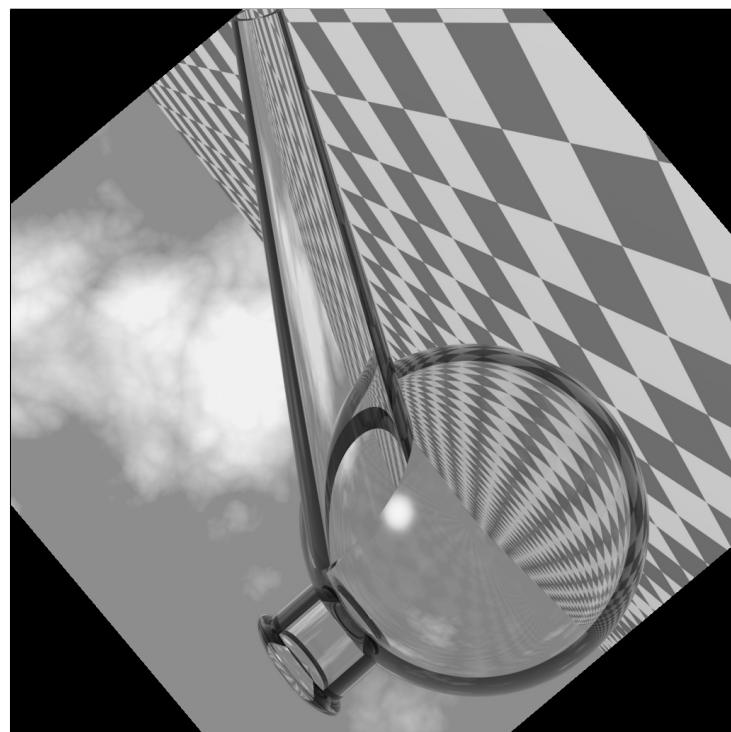


FIGURE F.13 – Rotate 130° bilinear

F.5 Scaling

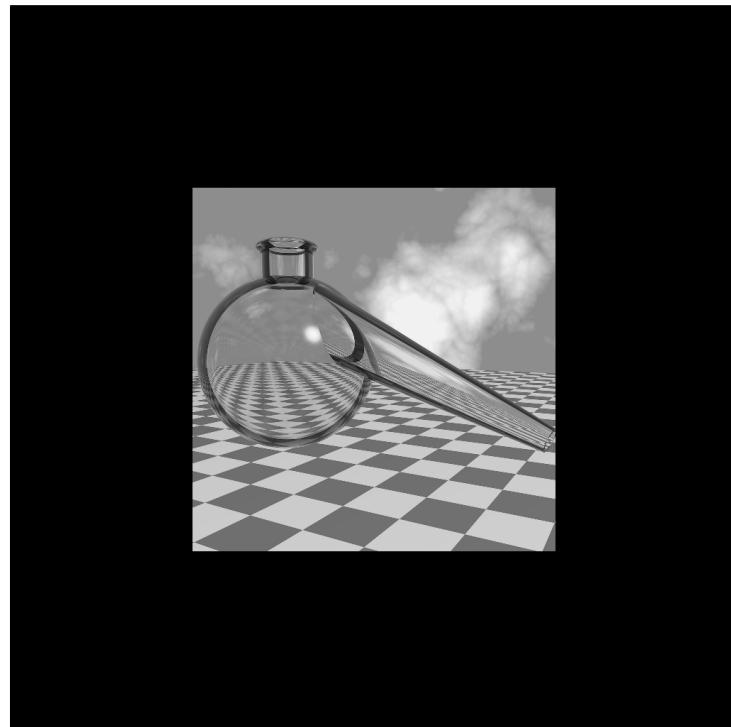


FIGURE F.14 – Scaling 0.5 nearest

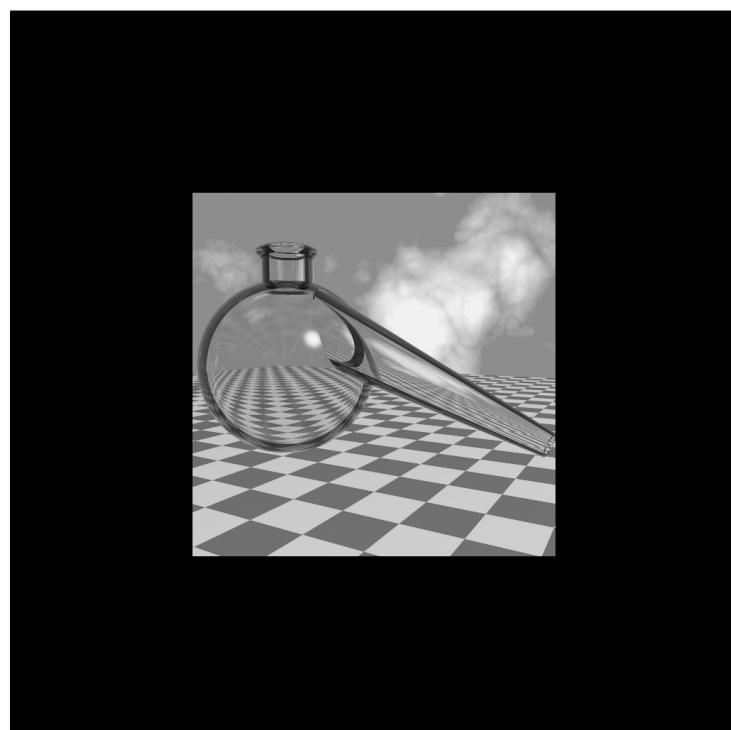


FIGURE F.15 – Scaling 0.5 bilinear

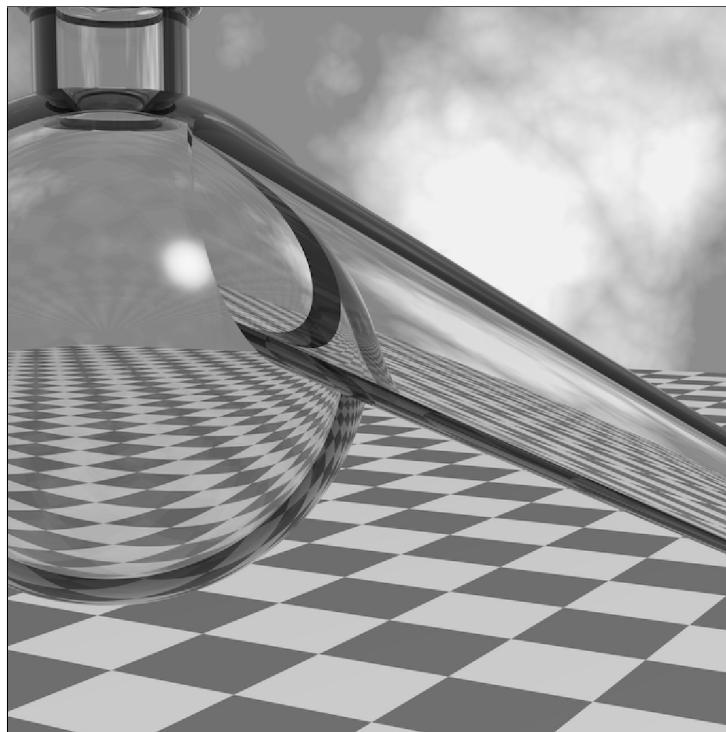


FIGURE F.16 – Scaling 1.5 nearest

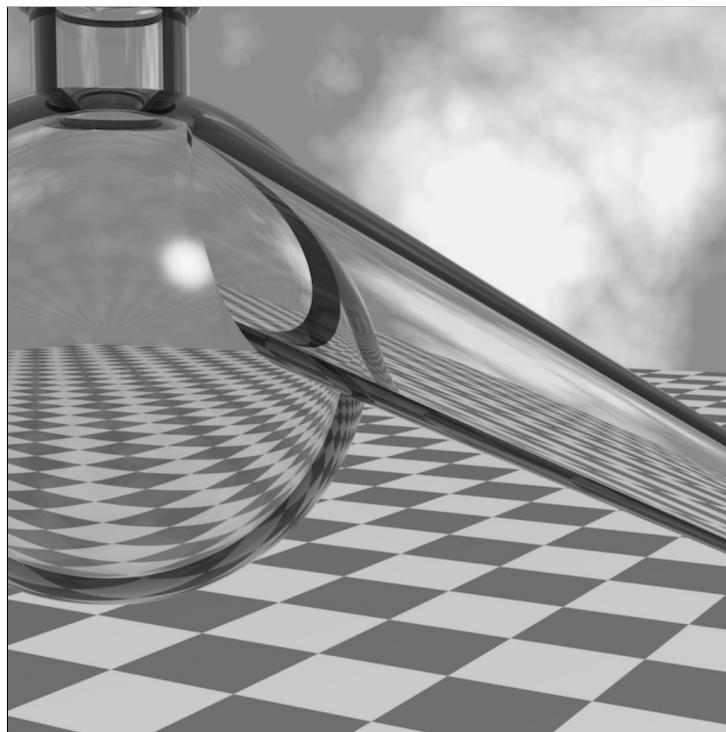


FIGURE F.17 – Scaling 1.5 bilinear

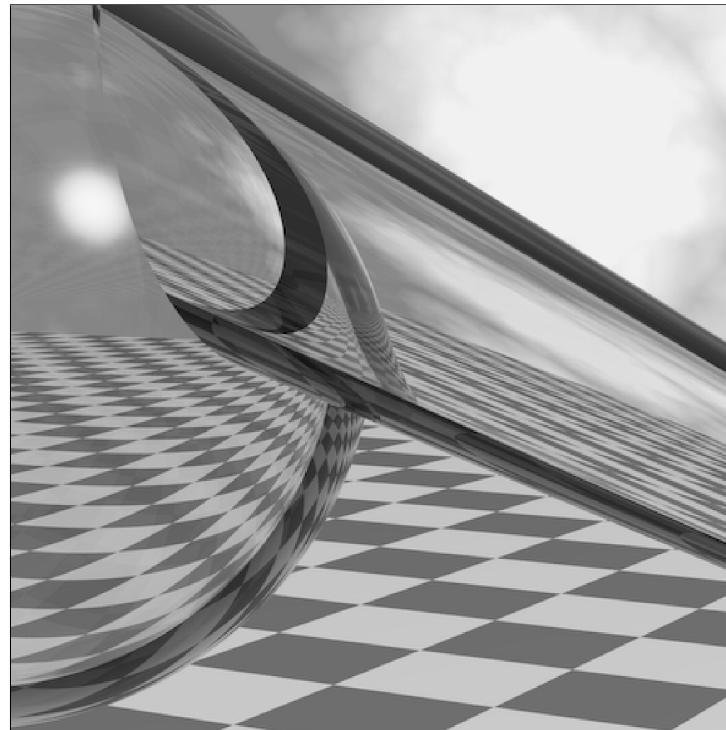


FIGURE F.18 – Scaling 2.3 nearest

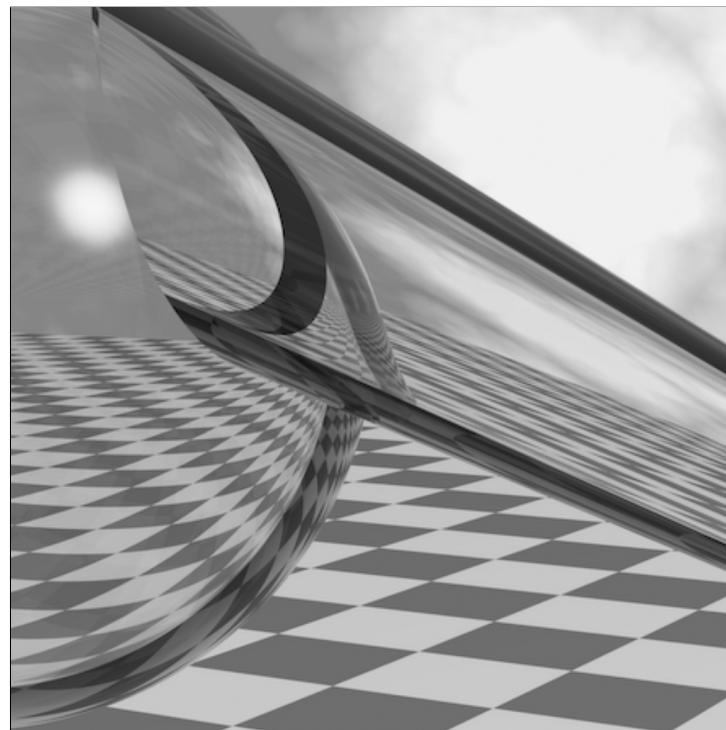


FIGURE F.19 – Scaling 2.3 bilinear

G. Transform image compression

G.1 Problem statement

- (a) Investigate image compression based on DCT. Divide the image into 8-by-8 subimages, compute the two-dimensional discrete cosine transform of each subimage, compress the test image to different qualities by discarding some DCT coefficients based on zonal mask and threshold mask and using the inverse discrete cosine transform with fewer transform coefficients. Display the original image, the reconstructed images and the difference images.
- (b) Investigate image compression based on wavelets.
Consider four types of wavelets :
 - (a) Haar (2x2)
 - (b) Daubechies (8-tap)
 - (c) Symlet (8-tap)
 - (d) Cohen-Daubechies-Feauveau (17-tap)

G.2 Python implementation

Two programs :

- DCT compression : **dct.py** Usage : **dct.py [-h] [-zonal | -threshold] [-z Z] image_path**
Use **python dct.py -h** to see the help.
- Wavelet compression : **wavelet.py** Usage : **wavelets.py [-h] [-haar | -daub | -symlet | -cohen] [-l LEVEL] [-t THRESHOLD] image_path** Use **python wavelets.py -h** to see the help.



FIGURE G.1 – Original image

G.3 Image compression based on DCT

G.3.1 Zonal mask

`python dct.py -zonal -z Z lenna.tif` where $Z = 1, 4$ or 7 .

Example : Zonal 5

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (\text{G.1})$$

Example : Zonal 2

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (\text{G.2})$$



FIGURE G.2 – Zonal 7



FIGURE G.3 – Zonal 7 - Diff



FIGURE G.4 – Zonal 4

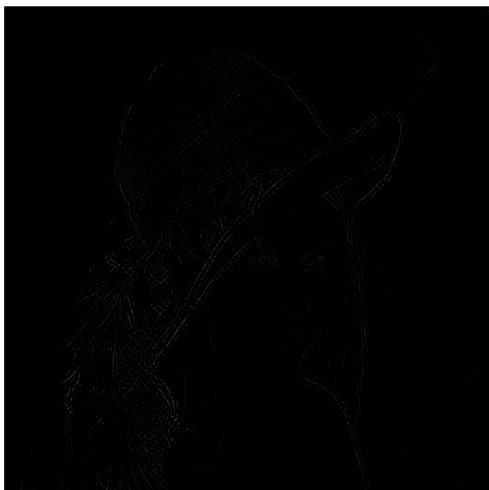


FIGURE G.5 – Zonal 4 - Diff



FIGURE G.6 – Zonal 2



FIGURE G.7 – Zonal 2 - Diff



FIGURE G.8 – Zonal 1



FIGURE G.9 – Zonal 1 - Diff

G.4 Image compression based on Wavelets

Decompose the image by wavelets, truncate some coefficients to 0 below a threshold (0 in this case) and reconstructed the image.

```
python wavelets.py -haar lenna.tif -l 1 -t 0.
```

G.4.1 Haar

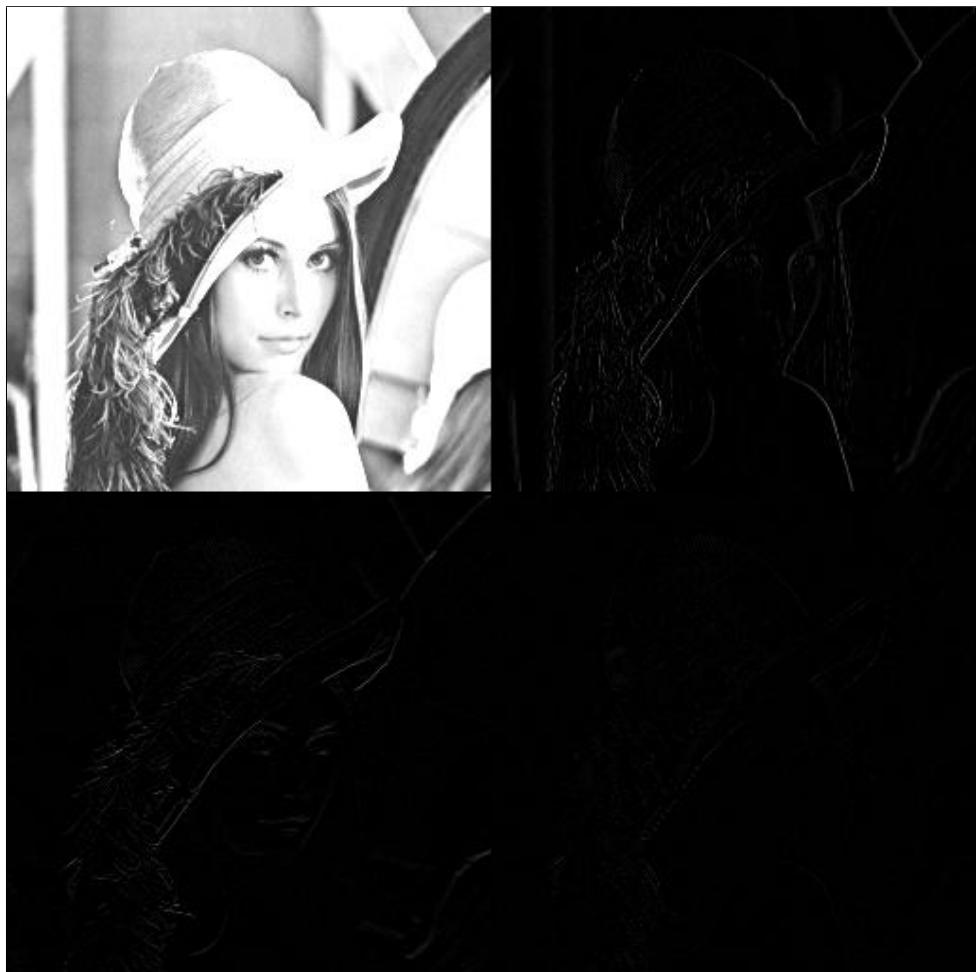


FIGURE G.10 – Haar



FIGURE G.11 – Haar reconstructed

FIGURE G.12 – Haar difference

G.4.2 Daubechies



FIGURE G.13 – Daubechies



FIGURE G.14 – Daubechies reconstructed **FIGURE G.15** – Daubechies difference

G.4.3 Symlet



FIGURE G.16 – Symlet



FIGURE G.17 – Symlet reconstructed

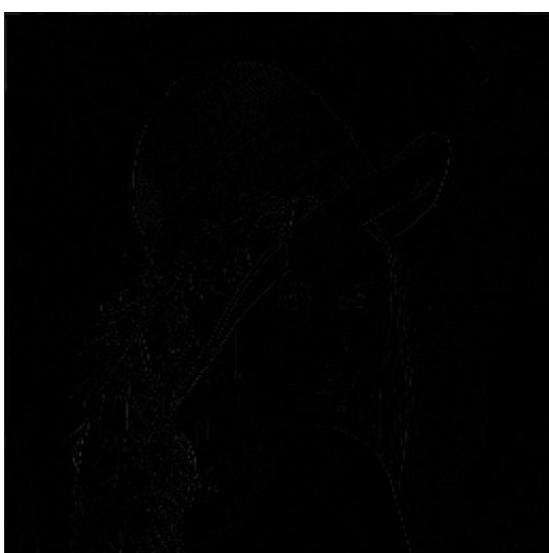


FIGURE G.18 – Symlet difference

G.4.4 Cohen-Daubechies-Feauveau

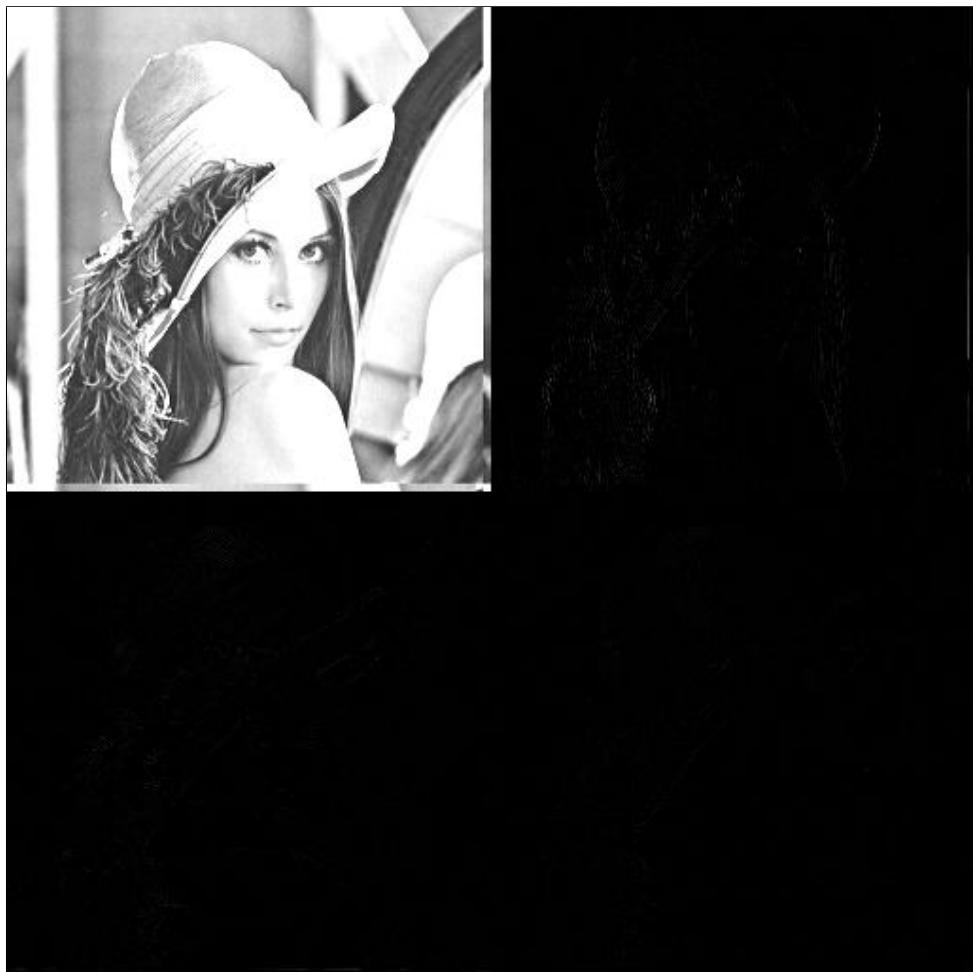


FIGURE G.19 – Cohen



FIGURE G.20 – Cohen reconstructed

FIGURE G.21 – Cohen difference

H. Morphological Processing

H.1 Problem statement

- (a) Implement the morphological operations : erosion, dilation, opening and closing, and use the noisy_fingerprint.tif to check your implementation.
- (b) Implement boundary extraction, hole filling, connected component extraction. Using lincoln_from_penny.tif, region_filling_reflection.tif and chickenfilet_with_bones.tif to the results, respectively.

H.2 Python implementation

```
Usage : problem8.py [-h] [-debug] [-test] [-erosion] [-dilation]
[-opening] [-closing] [-boundary] [-filling]
[-connected]
image_path
```

Use **python problem8.py -h** to see the help.

H.3 Erosion

```
python problem8.py -erosion noisy_fingerprint.tif
```



FIGURE H.1 – Original image



FIGURE H.2 – Erosion

H.4 Dilation

```
python problem8.py -dilation noisy_fingerprint.tif
```



FIGURE H.3 – Original image



FIGURE H.4 – Dilation

H.5 Opening

```
python problem8.py --opening noisy_fingerprint.tif
```



FIGURE H.5 – Original image

FIGURE H.6 – Opening

H.6 Closing

```
python problem8.py --closing noisy_fingerprint.tif
```



FIGURE H.7 – Original image

FIGURE H.8 – Closing

H.7 Boundary extraction

```
python problem8.py -boundary lincoln_from_penny.tif
```

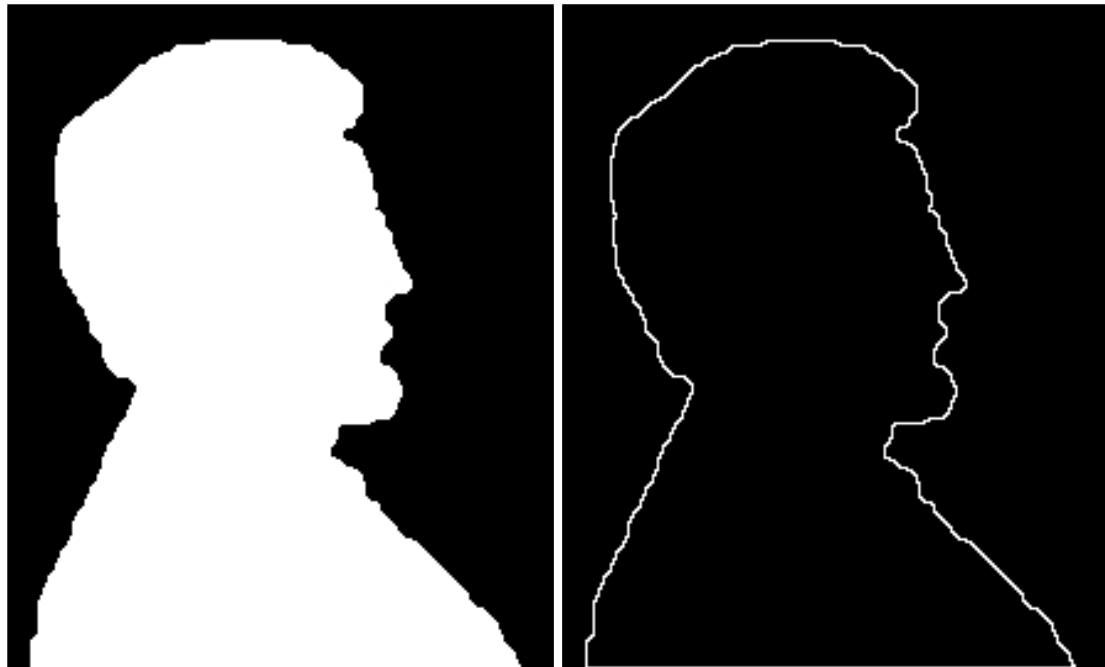


FIGURE H.9 – Original image

FIGURE H.10 – Boundary extraction

H.8 Hole filling

```
python problem8.py -filling region_filling_reflections.tif
```

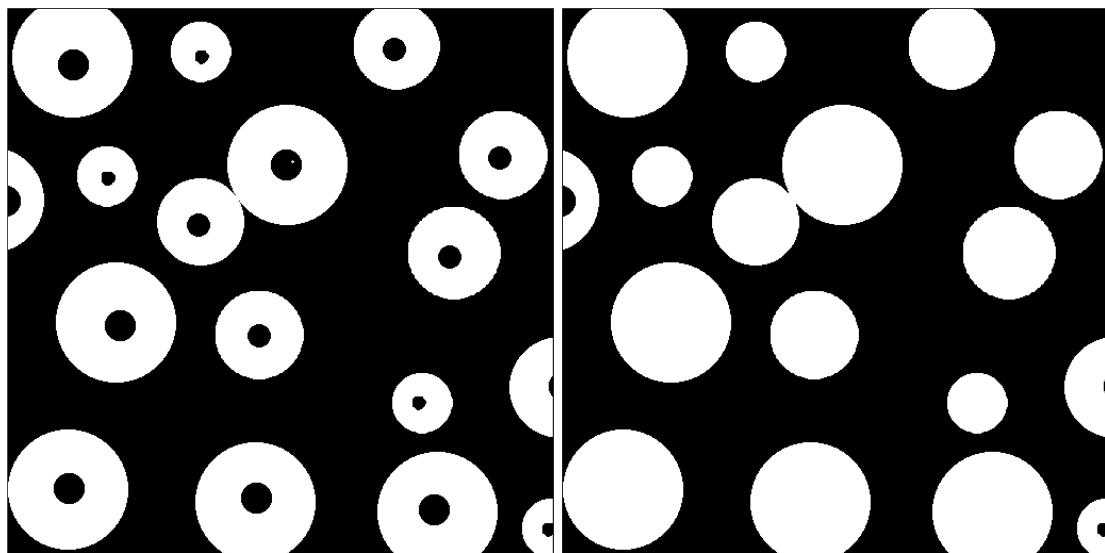


FIGURE H.11 – Original image

FIGURE H.12 – Hole filling

H.9 Connected component extraction

```
python problem8.py -connected chickenfilet_with_bones.tif
```

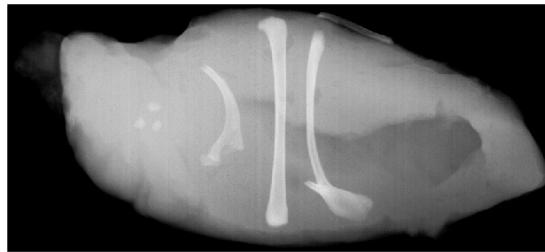


FIGURE H.13 – Original image

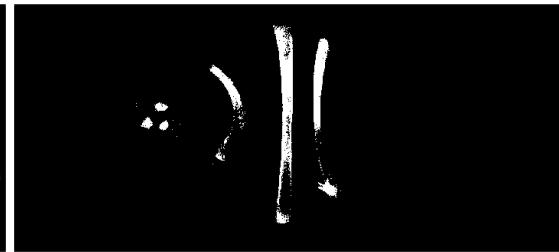


FIGURE H.14 – Thresholding



FIGURE H.15 – Erosion 5×5

Connected components :

Connected component	No of pixels
01	836
02	690
03	44
04	148
05	1
06	18
07	19
08	26
09	2
10	14
11	11
12	13
13	28
14	87
15	18

I. Image segmentation

I.1 Problem statement

- (a) Develop a program to implement the Roberts, Prewitt, Sobel, the Marr-Hildreth and the Canny edge detectors. Use the image ‘building.tif’ to test your detectors. (For technique details of Marr-Hildreth and Canny, please refer to pp.736-747 (3rd edition, Gonzalez DIP) or MH-Canny.pdf at the same address of the slides.)
- (b) Develop a program to implement the Otsu’s method of thresholding segementation, and compare the results with the global thresholding method using test image ‘polymersomes.tif’. (For technique details, please refer to pp.763-770 (3rd edition, Gonzalez DIP), or Otsu.pdf at the same ftp address of slides.)

I.2 Python implementation

Three programs :

- Marr-Hildreth edge detector : **marr.py**
Usage : **marr.py [-h] [-s SIGMA] image_path**
Use **python marr.py -h** to see the help.
- Canny edge detector : **canny.py**
Usage : **canny.py [-h] (-roberts | -sobel | -prewitt) [-s SIGMA] [-th TH] [-tl TL] image_path**
Use **python canny.py -h** to see the help.
- Otsu’s method of thresholding segmentation : **otsu.py**
Usage : **otsu.py [-h] [-o] [-g] image_path**
Use **python otsu.py -h** to see the help.

I.3 Marr-Hildreth edge detector

```
python marr.py -s 4 building.tif
```



FIGURE I.1 – Original image

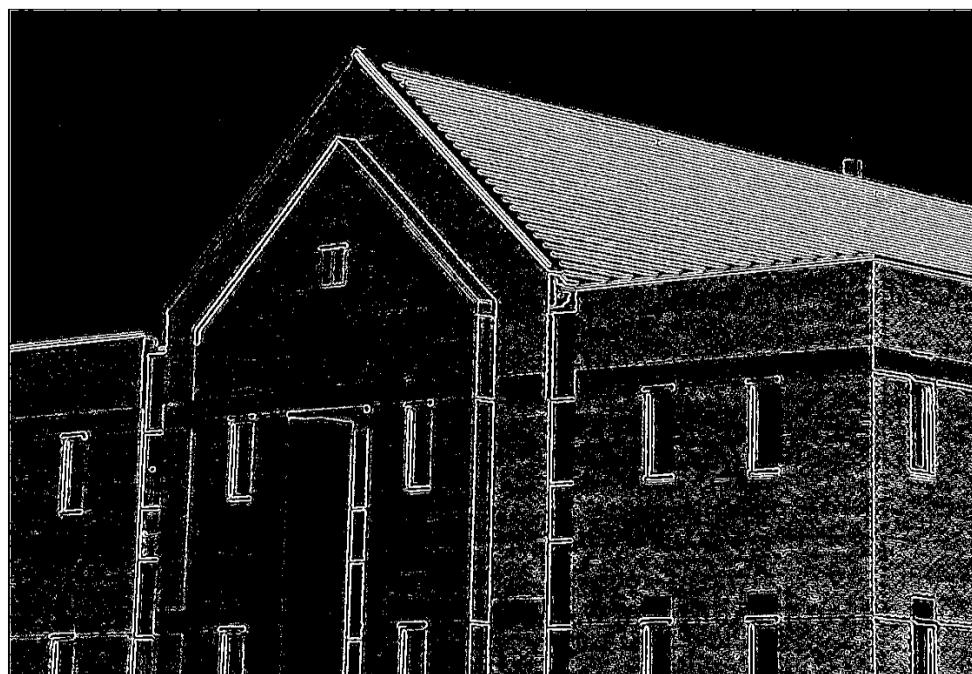


FIGURE I.2 – Marr-Hildreth

I.4 Canny edge detector

I.4.1 Roberts

```
python canny.py -roberts -s 4 -tl 0.04 -th 0.10 building.tif
```



FIGURE I.3 – Original image

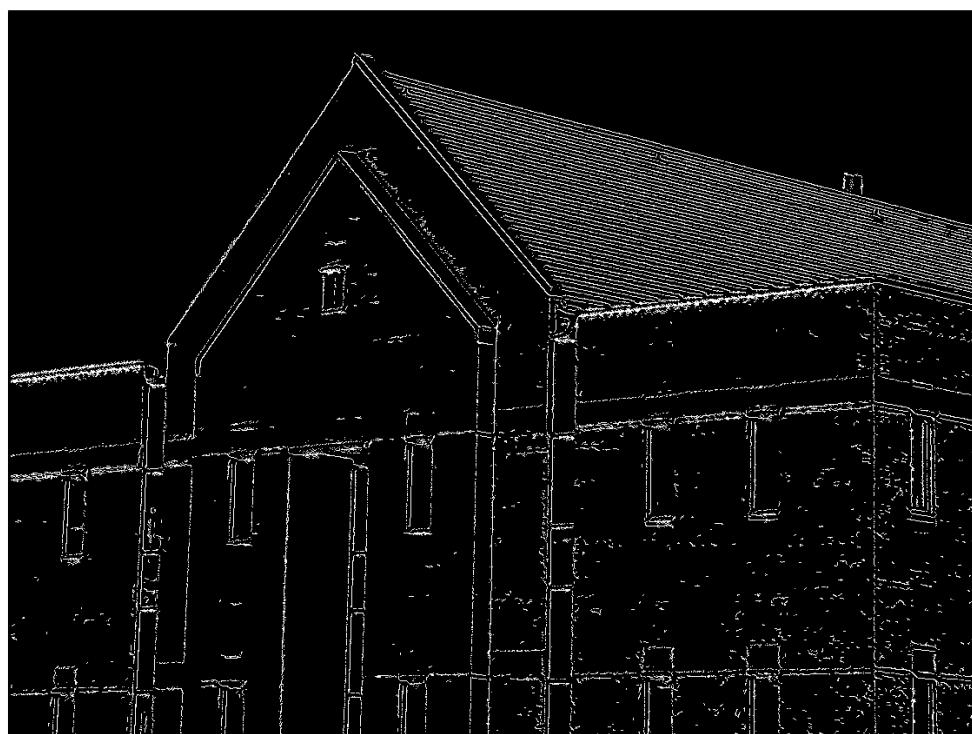


FIGURE I.4 – Roberts

I.4.2 Prewitt

```
python canny.py -prewitt -s 4 -tl 0.04 -th 0.10 building.tif
```



FIGURE I.5 – Original image

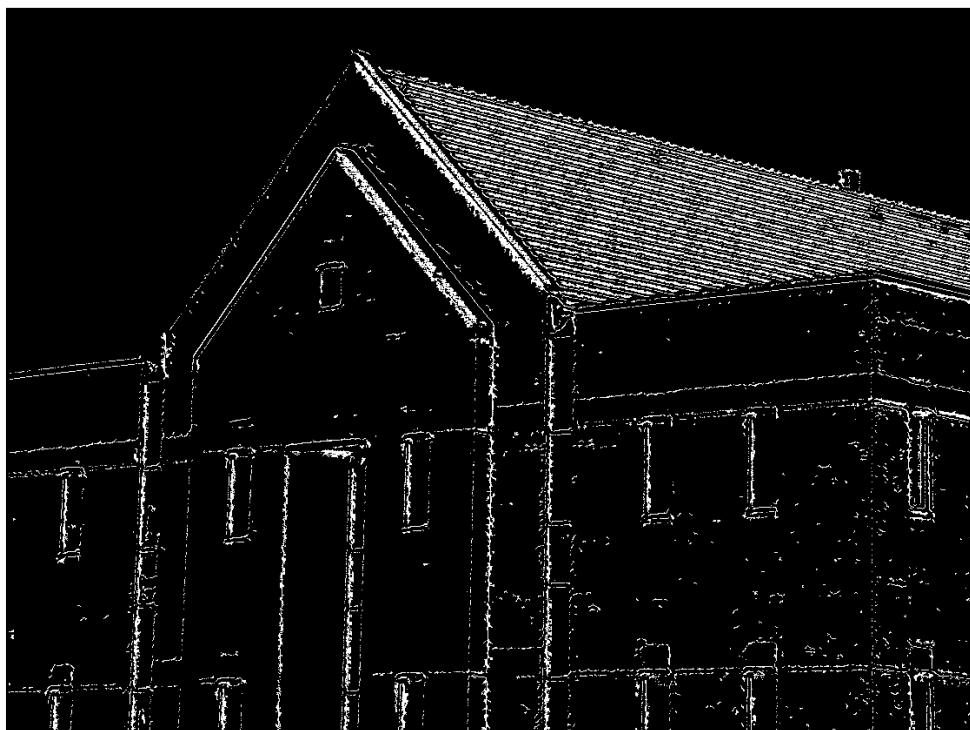


FIGURE I.6 – Prewitt

I.4.3 Sobel

```
python canny.py --sobel -s 4 -tl 0.04 -th 0.10 building.tif
```



FIGURE I.7 – Original image

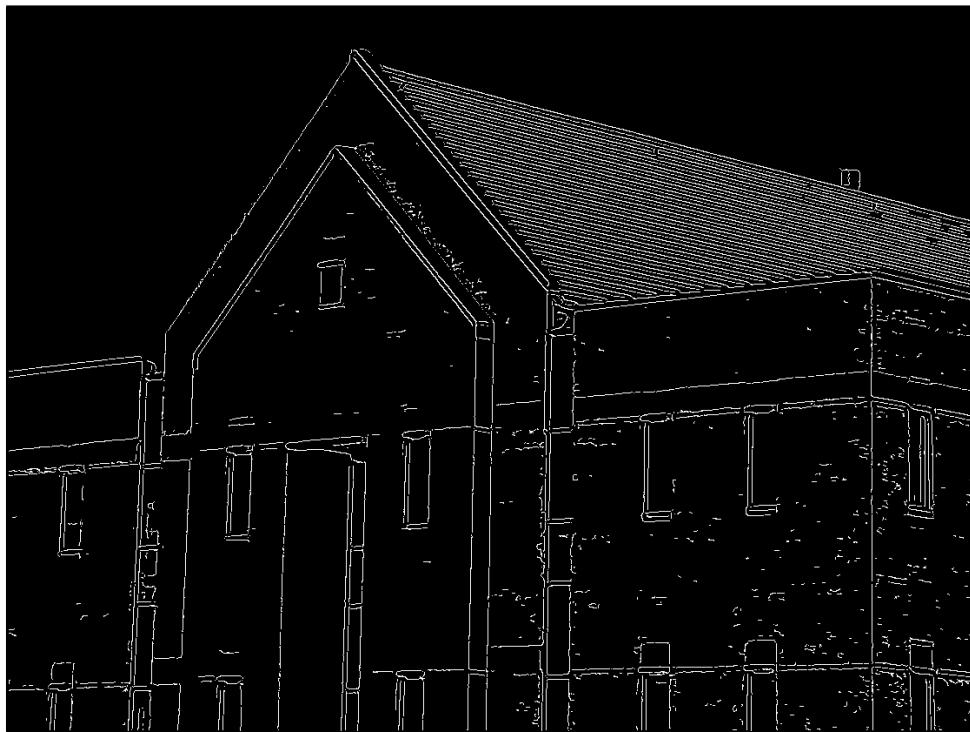


FIGURE I.8 – Sobel

I.5 Thresholding segmentation

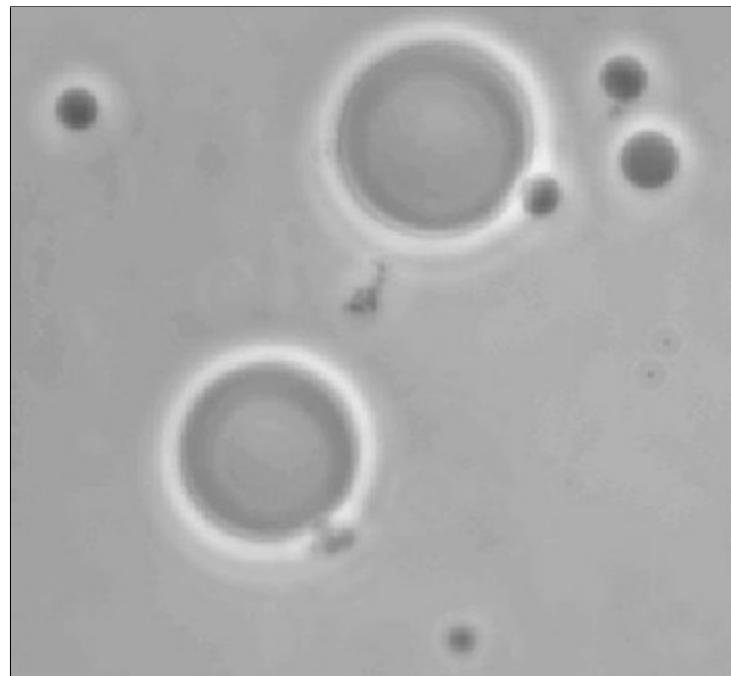


FIGURE I.9 – Original image

I.5.1 Otsu

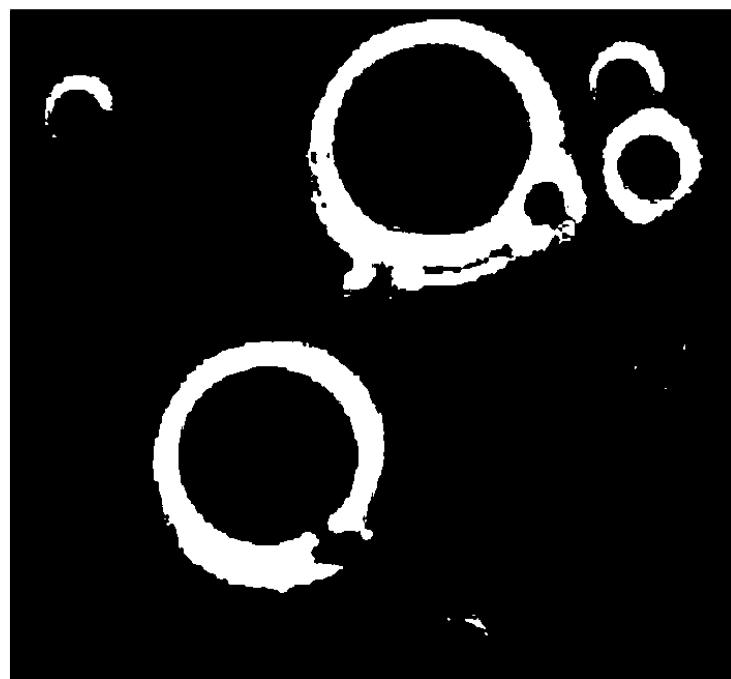


FIGURE I.10 – Otsu

I.5.2 Global thresholding



FIGURE I.11 – Global thresholding

J. Image representation and description

J.1 Problem statement

- (a) Develop a program to implement the boundary following algorithm, the resampling grid and calculate the chain code and the first difference chain code. Use the image ‘noisy_stroke.tif’ for test. (For technique details, please refer to pp.818-822 (3rd edition, Gonzalez DIP) or boundaryfollowing.pdf at the same address of the slides.)
- (b) Develop a program to implement the image description by the principal components (PC). Calculate and display the PC images and the reconstructed images from 2 PCs. Use the six images in ‘washingtonDC.rar’ as the test images.

J.2 Python implementation

Four programs :

- Boundary following : **boundary.py**
Usage : **boundary.py [-h] [-smooth] image_path**
Use **python boundary.py -h** to see the help.
- Resampling grid : **resampling.py**
Usage : **resampling.py [-h] [-s SAMPLING [SAMPLING ...]] boundary_image**
Use **python resampling.py -h** to see the help.
- Chain code : **chaincode.py**
Usage : **chaincode.py [-h] [-s SAMPLING [SAMPLING ...]] boundary_image**
Use **python chaincode.py -h** to see the help.
- Image description by the principal components (PC) : **pc.py**
Usage : **pc.py [-h] [-n N] [-debug] [-diff] [-nshow]**
Use **python pc.py -h** to see the help.

J.3 Boundary following

`python boundary.py noisy_stroke.tif.`

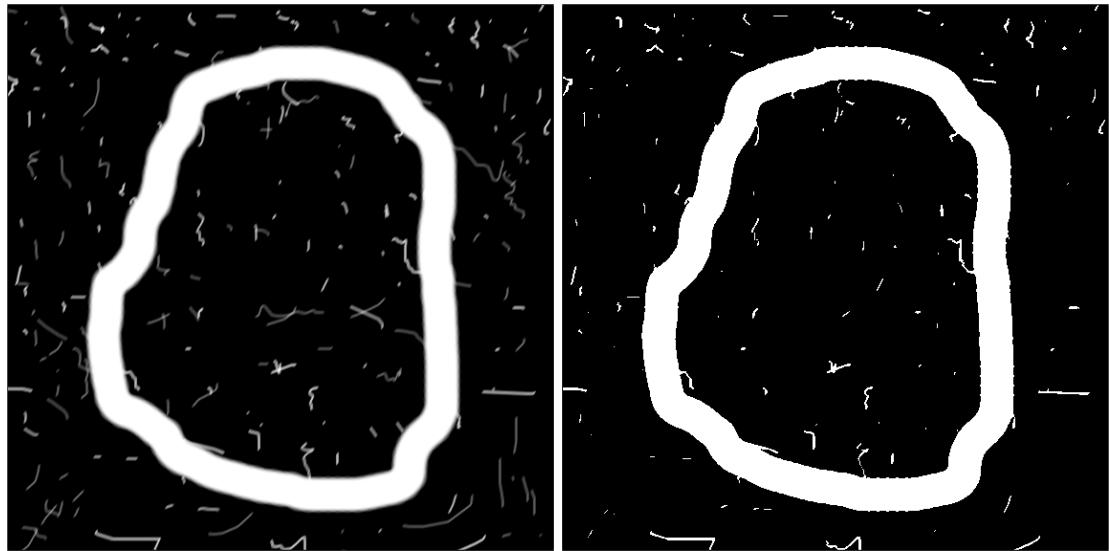


FIGURE J.1 – Original image

FIGURE J.2 – Black & white

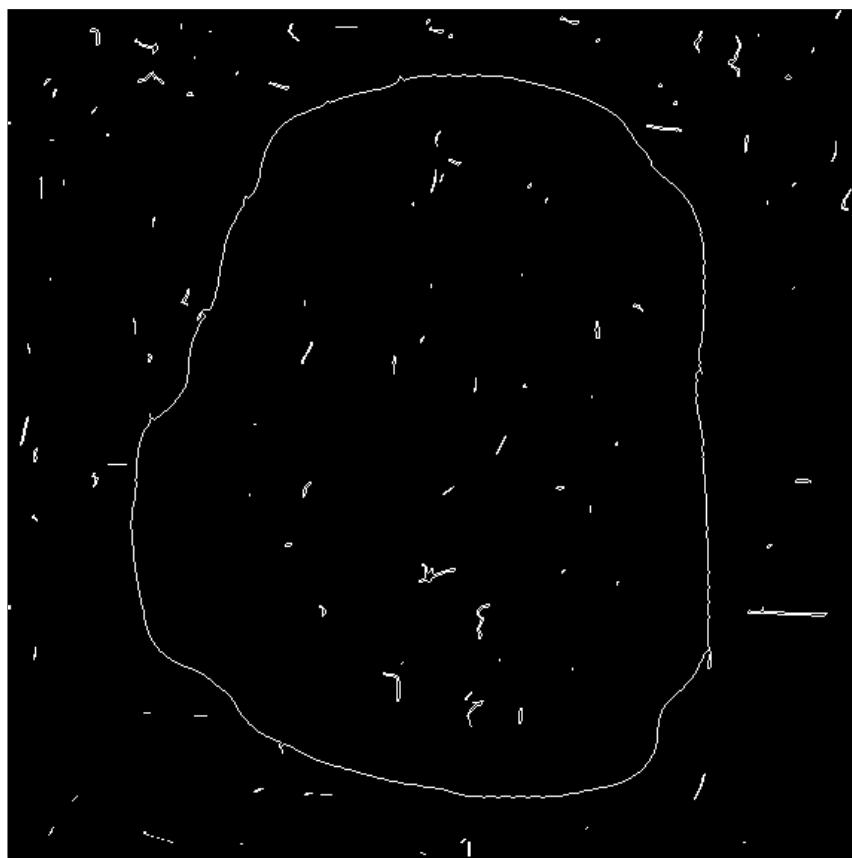


FIGURE J.3 – Boundaries

Even the boundaries of the noise are found... We need to remove the noise beforehand. For that, let use a Gaussian blur of mean 0 and variance 10.

```
python boundary.py noisy_stroke.tif -smooth.
```

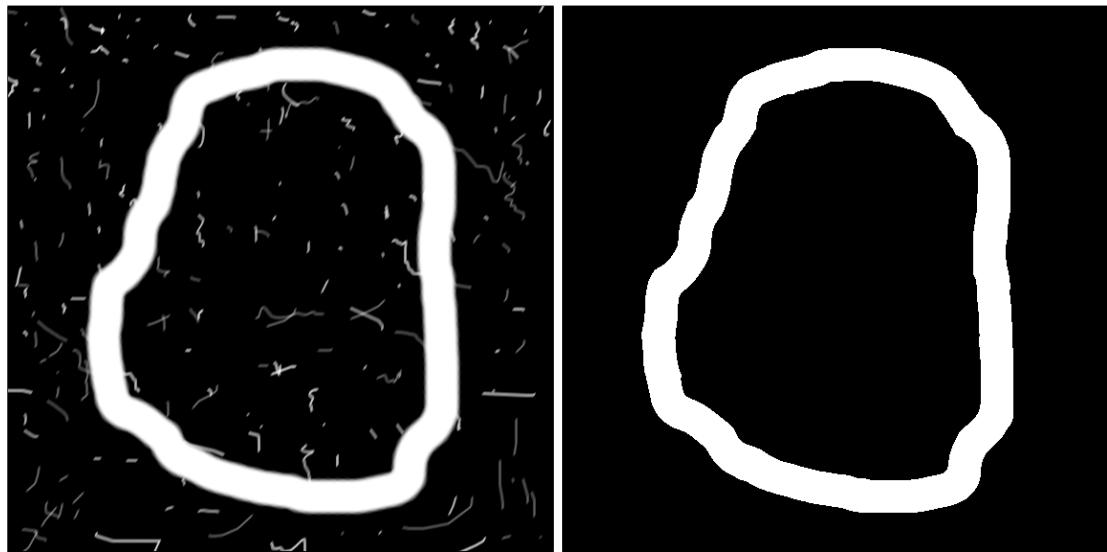


FIGURE J.4 – Original image

FIGURE J.5 – Smoothing + binarisation

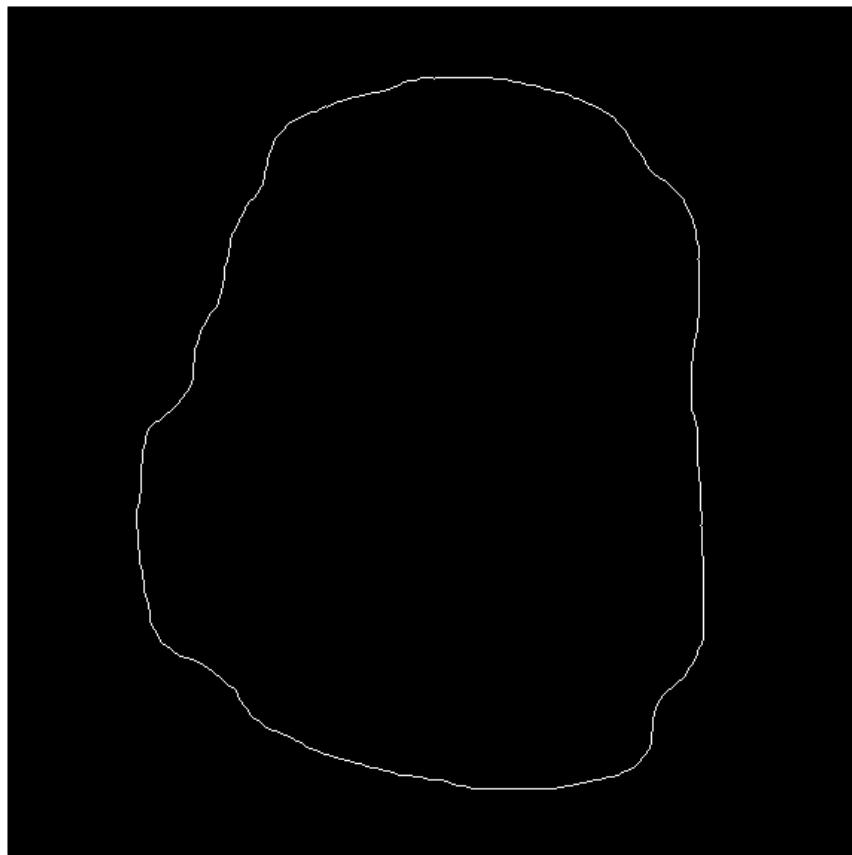


FIGURE J.6 – Boundaries

J.4 Resampling grid

`python resampling.py noisy_stroke_boundary.png -s Sx Sy`, where Sx and Sy are the sampling intervalles along the X and Y axis.

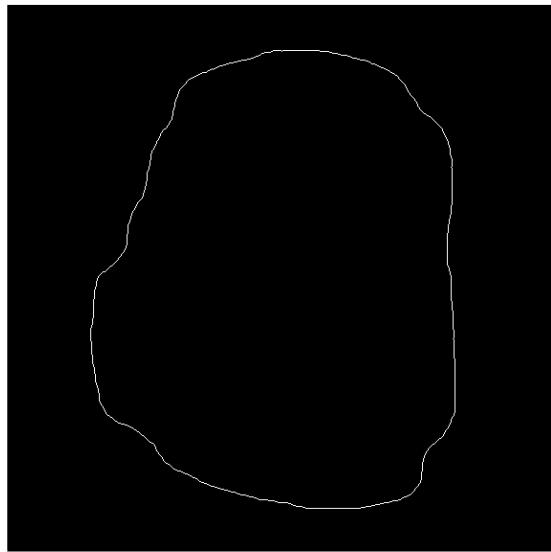


FIGURE J.7 – Original image

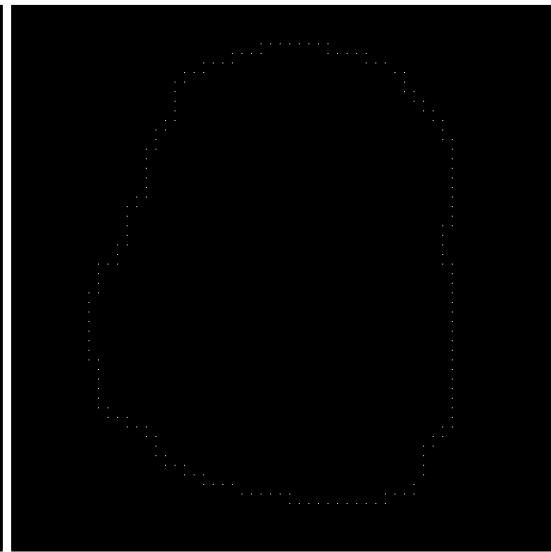


FIGURE J.8 – R-grid ($S = (10, 10)$)

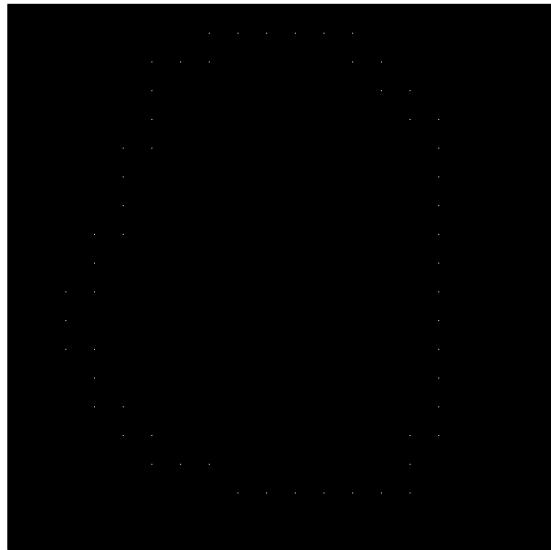


FIGURE J.9 – R-grid ($S = (30, 30)$)

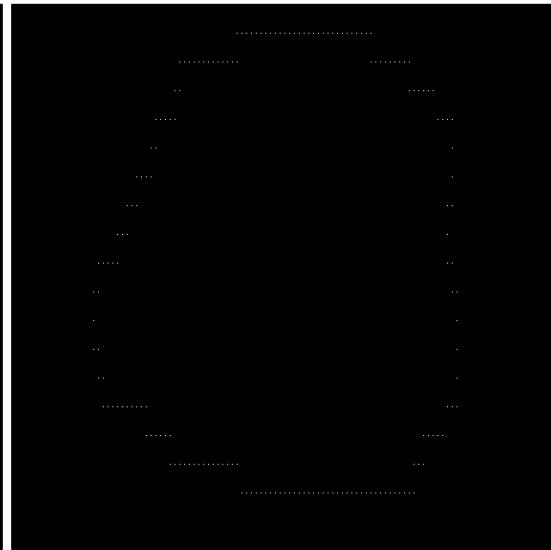


FIGURE J.10 – R-grid ($S = (5, 30)$)

J.5 Chain code and first difference chain code

J.5.1 Chain code - resampling grid (10, 10)

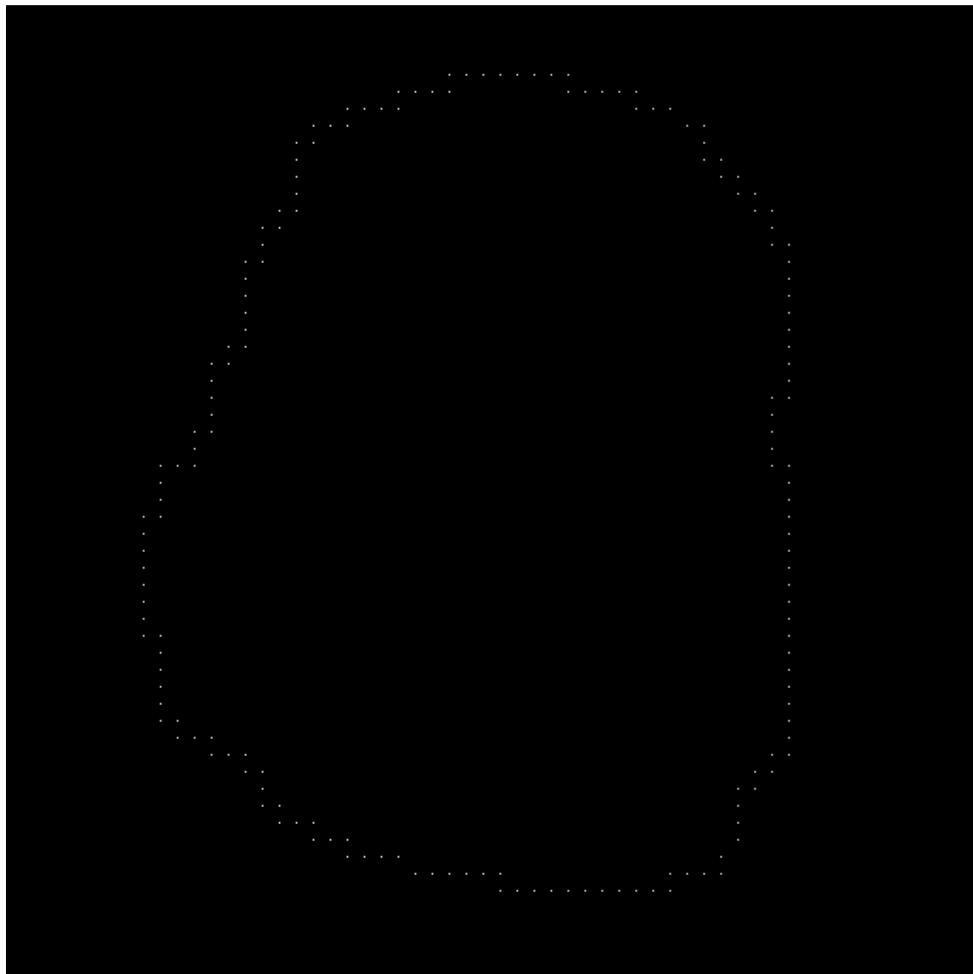


FIGURE J.11 – Resampling grid ($S = (10, 10)$)

Chaincode (length = 170) :

```
00000060000600706606060606606666666  
6646666066666666666666646466656  
4446444444444424444344424424424224  
2442442422222422222202220022022220  
20222220220202220200200020002
```

First difference (length = 169) :

```
000006200062071602626260260000000  
0620002600000000000000062626200716  
0026000000000620007100620620626026  
206206260002600000620060206200062  
62000062062620006260260026002
```

J.5.2 Chain code - resampling grid (30, 30)

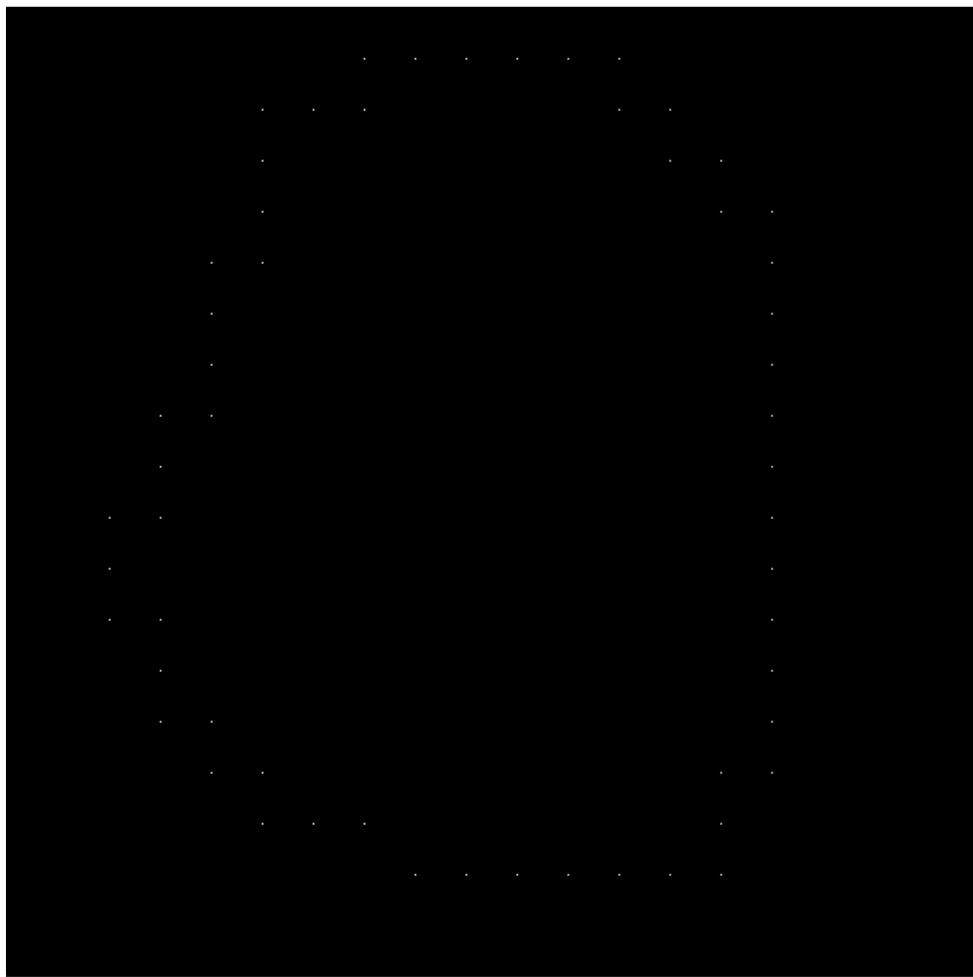


FIGURE J.12 – Resampling grid ($S = (30, 30)$)

Chaincode (length = 56) :

00060606066666666664664444434424242242202202220020

First difference (length = 55) :

0062626260000000000620600000710626260260620620062006026

J.6 Principal components

```
python pc.py -diff -n 2
```

We keep only 2 principal components for the images reconstruction.



FIGURE J.13 – Band 1



FIGURE J.14 – Band 1 reconstructed



FIGURE J.15 – Band 1 difference

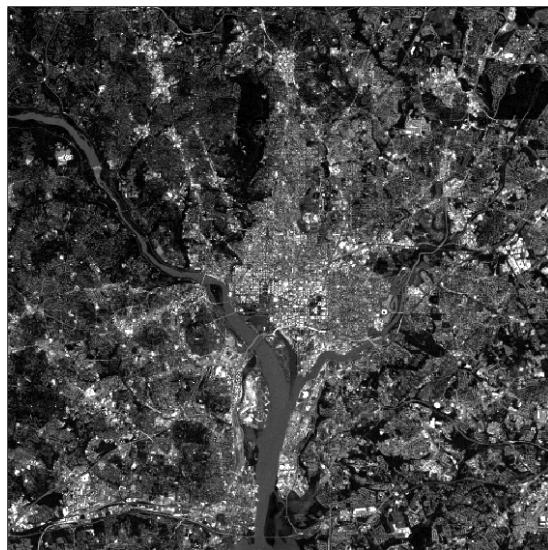


FIGURE J.16 – Band 2

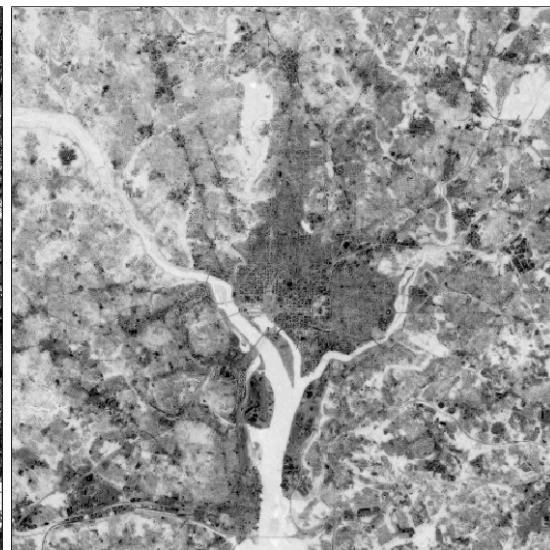


FIGURE J.17 – Band 2 reconstructed

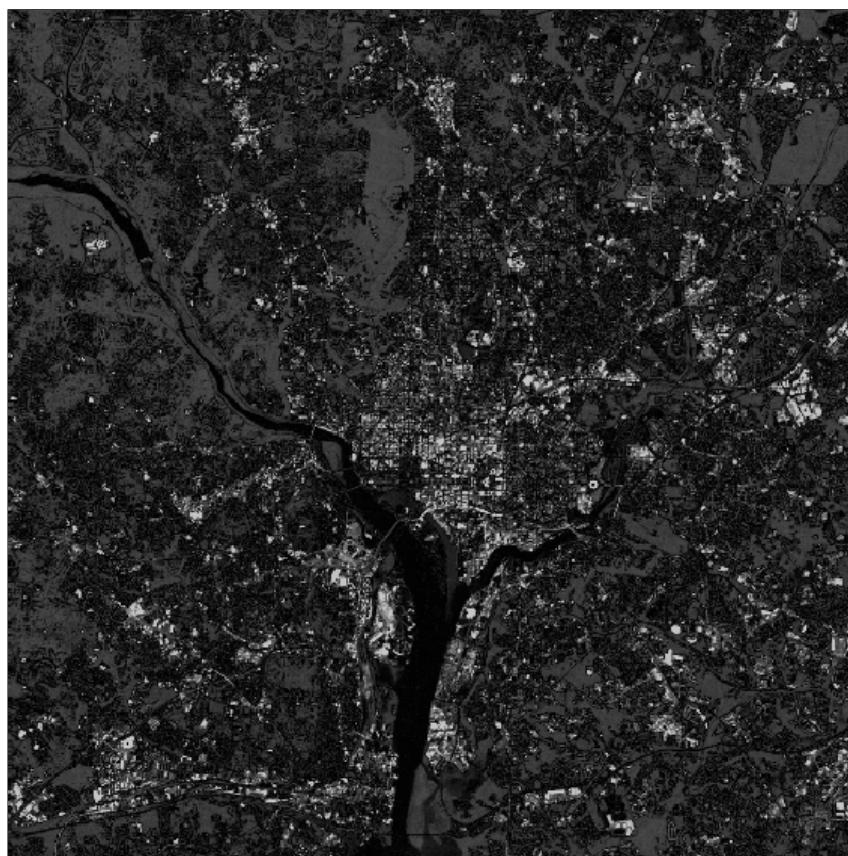


FIGURE J.18 – Band 2 difference



FIGURE J.19 – Band 3



FIGURE J.20 – Band 3 reconstructed

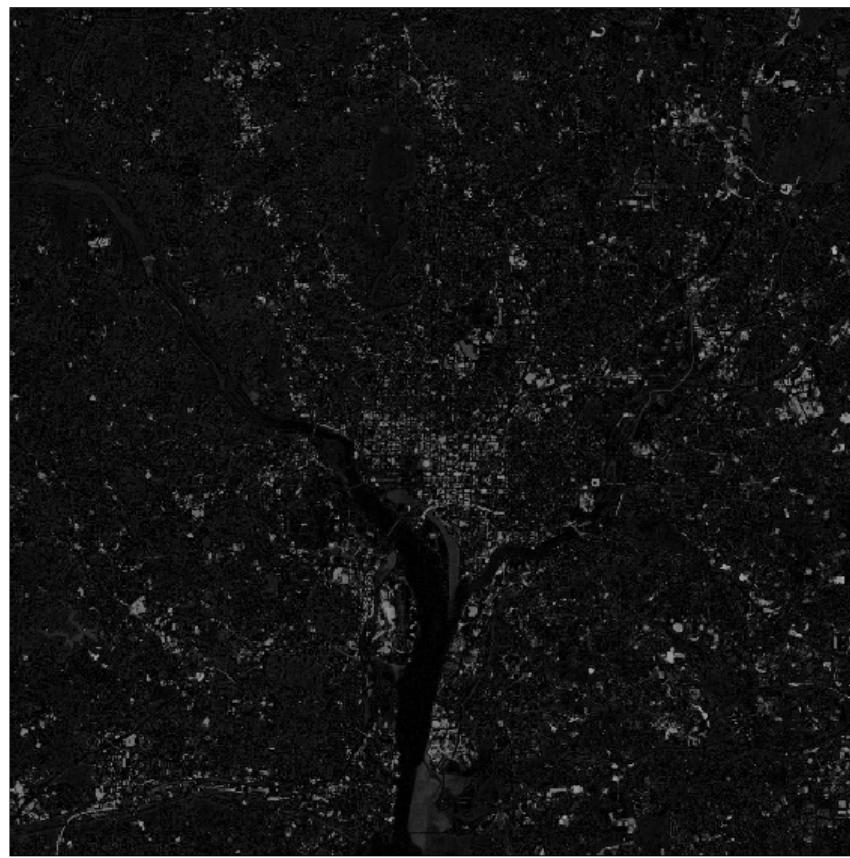


FIGURE J.21 – Band 3 difference

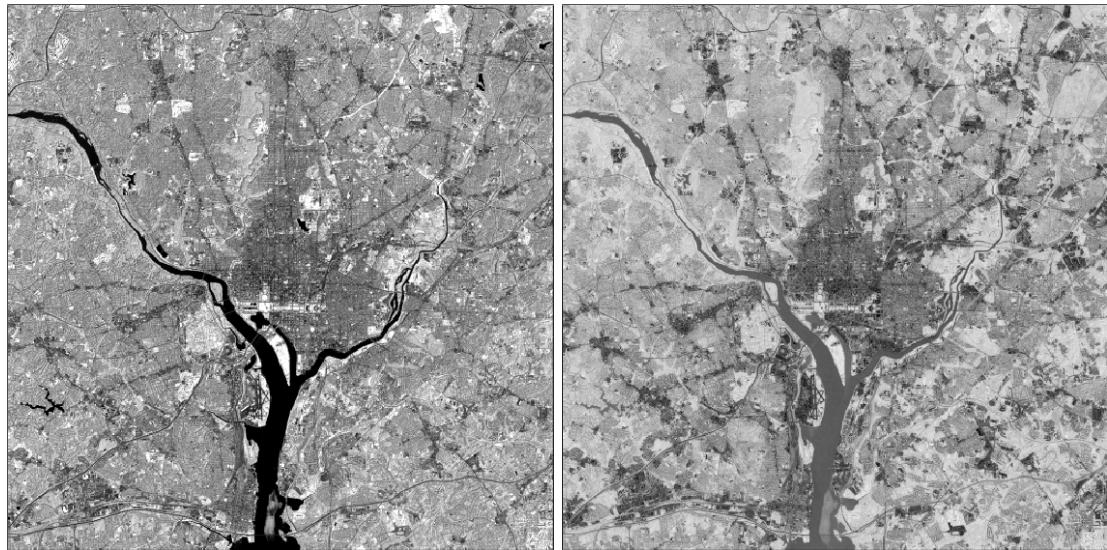


FIGURE J.22 – Band 4

FIGURE J.23 – Band 4 reconstructed



FIGURE J.24 – Band 4 difference

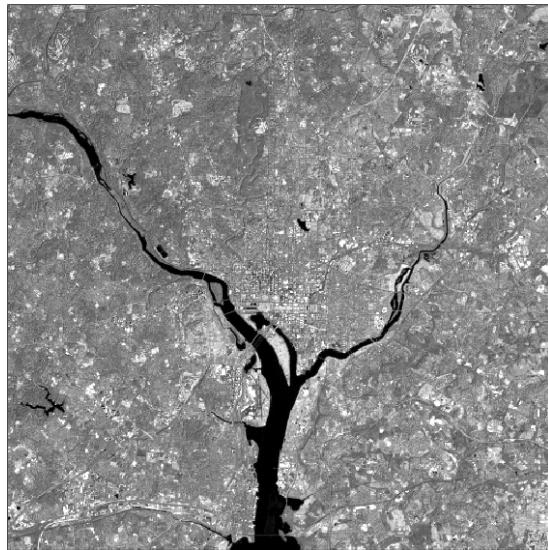


FIGURE J.25 – Band 5

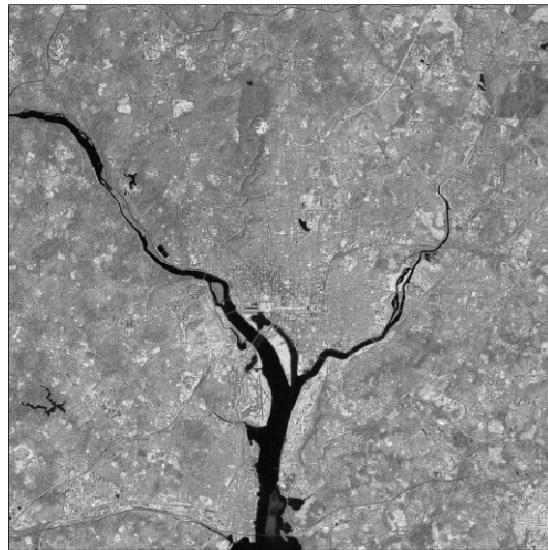


FIGURE J.26 – Band 5 reconstructed

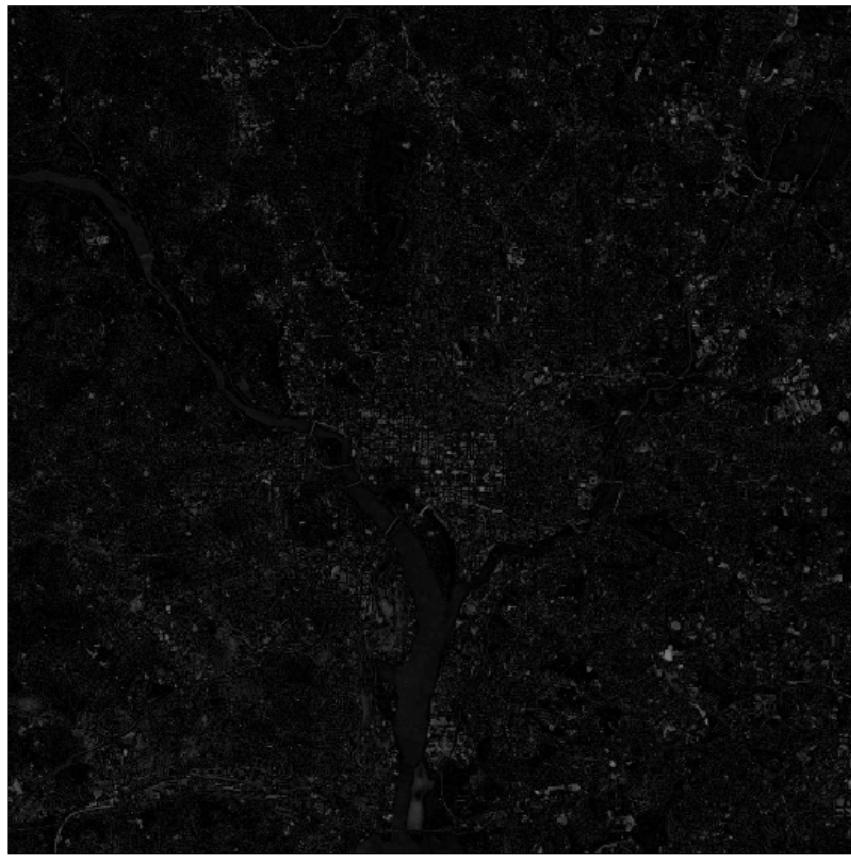


FIGURE J.27 – Band 5 difference

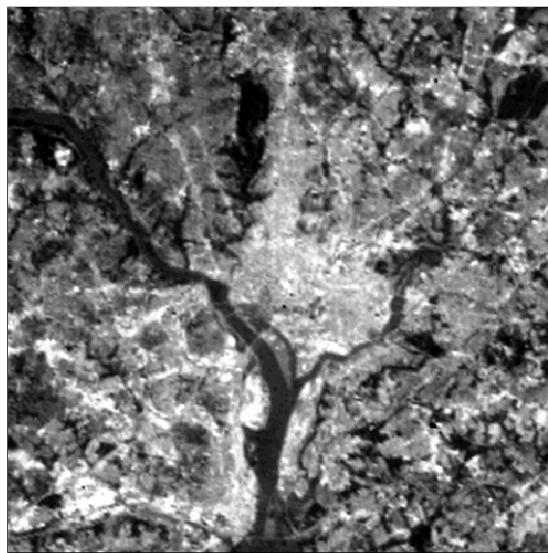


FIGURE J.28 – Band 6



FIGURE J.29 – Band 6 reconstructed

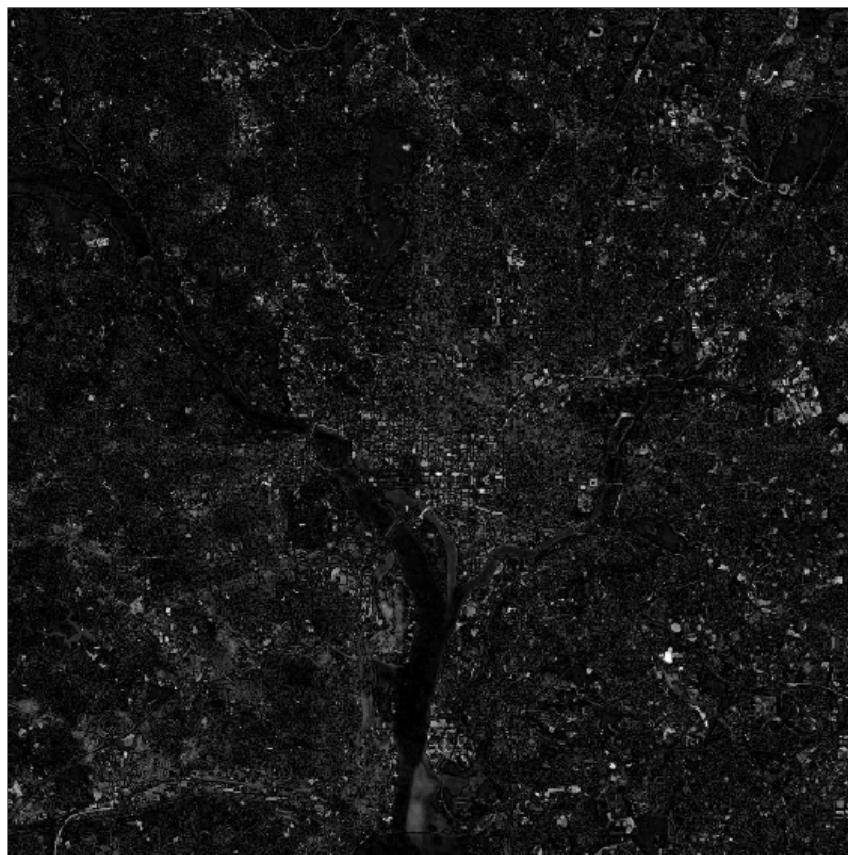


FIGURE J.30 – Band 6 difference