

[디지털 영상 처리] 과제2. Area processing

컴퓨터공학부 201911228 홍지우

Contents

1. Smoothing 및 noise filtering
 - Gaussian filtering
 - median filtering
 - average filtering
2. Sharpening : Highboost filter
3. Color space에서 Smoothing
4. Edge detection
 - Prewitt, Sobel (1차 미분 연산자)
 - LoG filter
5. Canny edge operator

Smoothing 및 noise filtering

Gaussian filtering – 소스코드

```
import cv2
import numpy as np
from math import acos, pi, sqrt
from PIL import Image, ImageFilter
```

```
def gaussian1D(sigma):
    a = sigma

    if a % 2 == 0: #mask size가 짝수일 경우 1을 더한다
        a+=1

    a2 = a/2
    filter_size = np.array(range(-int(a/2),int(a/2)+1))
    result = np.zeros(len(filter_size))

    for i in range(len(filter_size)):
        x = i-a2
        result[i] = float(np.exp(-(x**2)/(2*sigma**2)) / (2*3.14*sigma**2)) #가우시안 공식

    total = np.sum(result)
    result/=total

    return result
```

```
def gaussian2D(sigma):
    # 두 벡터를 외적하여 x,y 2차원 mask를 생성
    result = np.outer(gaussian1D(sigma),gaussian1D(sigma))
    total = np.sum(result)
    result/=total

    return result
```

```
def convolution(array, f):
    a = (len(f)-1)//2
    # boundary를 0으로 채운다
    img_pad = np.pad(array, ((a,a),(a,a)), 'constant')

    #convolution을 위해 2번 뒤집는다
    mask = np.rot90(f)
    mask = np.rot90(mask)

    result_img = np.zeros((len(array), len(array[0])))
    result_img = result_img.astype('float32')

    for i in range(len(result_img)):
        for j in range(len(result_img[0])):
            result_img[i][j] = np.sum(img_pad[i:i+len(f),j:j+len(f)]*mask)

    return result_img
```

```
def real_convolution(array, sigma):
    return convolution(array, gaussian2D(sigma))
```

```
[[0.09555231 0.10678153 0.10678153]
 [0.10678153 0.11933039 0.11933039]
 [0.10678153 0.11933039 0.11933039]]
```

[3*3 mask]

```
[[0.03402125 0.03685478 0.03835885 0.03835885 0.03685478]
 [0.03685478 0.0399243 0.04155364 0.04155364 0.0399243 ]
 [0.03835885 0.04155364 0.04324948 0.04324948 0.04155364]
 [0.03835885 0.04155364 0.04324948 0.04324948 0.04155364]
 [0.03685478 0.0399243 0.04155364 0.04155364 0.0399243 ]]
```

[5*5 mask]

```
[[0.01730548 0.01839811 0.01916458 0.01955972 0.01955972 0.01916458
 0.01839811]
 [0.01839811 0.01955972 0.02037459 0.02079467 0.02079467 0.02037459
 0.01955972]
 [0.01916458 0.02037459 0.02122341 0.02166099 0.02166099 0.02122341
 0.02037459]
 [0.01955972 0.02079467 0.02166099 0.02210759 0.02210759 0.02166099
 0.02079467]
 [0.01955972 0.02079467 0.02166099 0.02210759 0.02210759 0.02166099
 0.02079467]
 [0.01916458 0.02037459 0.02122341 0.02166099 0.02166099 0.02122341
 0.02037459]
 [0.01839811 0.01955972 0.02037459 0.02079467 0.02079467 0.02037459
 0.01955972]]
```

[7*7 mask]

median filtering – 소스코드

```
import cv2
import numpy as np
```

```
img = cv2.imread('Fig0507(a)(ckt-board-orig).jpg')
height, width, channel = img.shape

# 결과 배열 생성
out1 = np.zeros((height + 2, width + 2, channel), dtype=np.float)
out1[1: 1 + height, 1: 1 + width] = img.copy().astype(np.float)
templ = out1.copy()

mask = 3

for i in range(height):
    for j in range(width):
        for k in range(channel): # mask size만큼 중간값 계산
            out1[1 + i, 1 + j, k] = np.median(templ[i:i + mask, j:j + mask, k])

out1 = out1[1:1 + height, 1:1 + width].astype(np.uint8)
cv2.imshow('result', out1)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

average filtering – 소스 코드

```
import cv2
import numpy as np
from PIL import Image
```

```
def average_filter(img, size):

    if len(img.shape) == 3: # 컬러 이미지
        H, W, C = img.shape
    else: # 흑백 이미지
        img = np.expand_dims(img, axis=-1)
        H, W, C = img.shape

    pad = size // 2
    out = np.zeros((H + pad * 2, W + pad * 2, C), dtype=np.float)
    out[pad: pad + H, pad: pad + W] = img.copy().astype(np.float)

    # mask 생성
    mask = np.ones((size, size))/size**2
    print(mask)

    tmp = out.copy()

    for i in range(H):
        for j in range(W):
            for k in range(C):
                out[pad + i, pad + j, k] = np.sum(mask * tmp[i: i + size, j: j + size, k])

    # 0-255사이의 값으로 변환
    out = np.clip(out, 0, 255)
    out = out[pad: pad + H, pad: pad + W].astype(np.uint8)

    return out
```

```
img = cv2.imread('Fig0507(a)(ckt-board-orig).jpg')
out = average_filter(img, 5)

cv2.imshow("result", out)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Smoothing 필터 및 noise filtering 이미지

원본 이미지

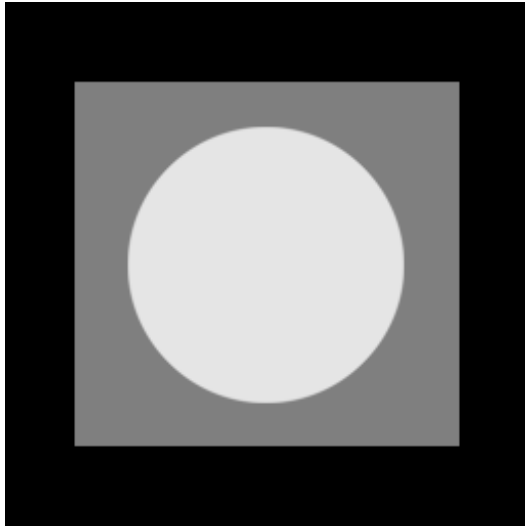


Fig0503 (original_pattern).jpg

gaussian 노이즈 이미지

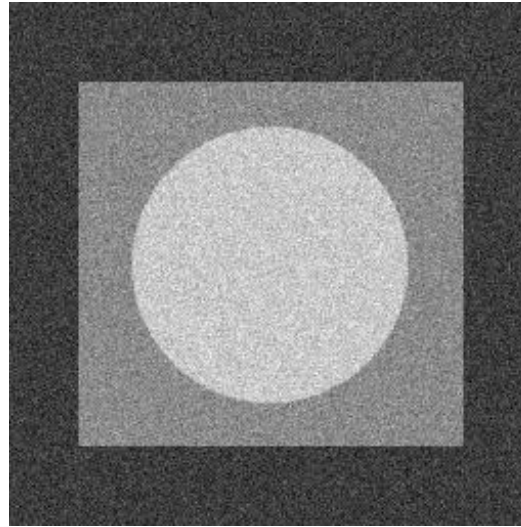


Fig0504(a)(gaussian-noise).jpg

salt and pepper 노이즈 이미지

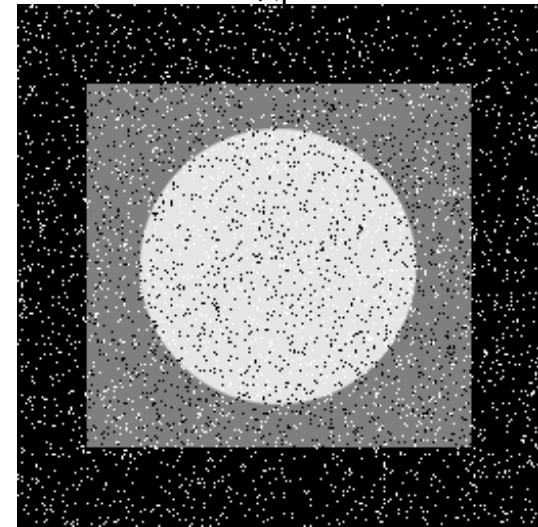



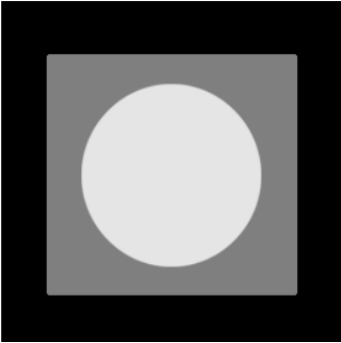

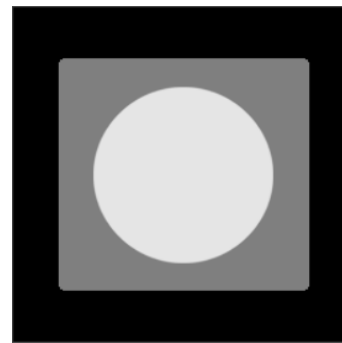



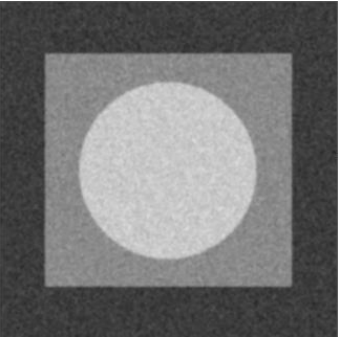
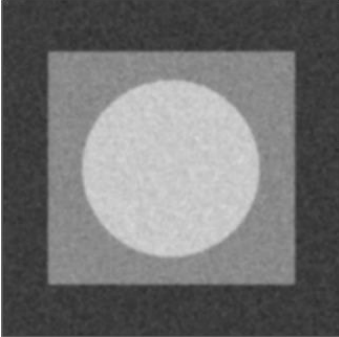
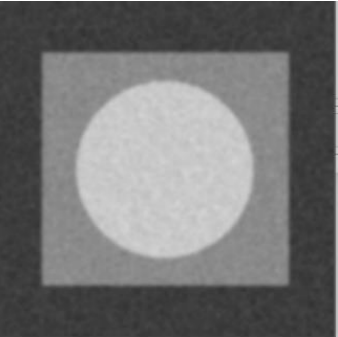
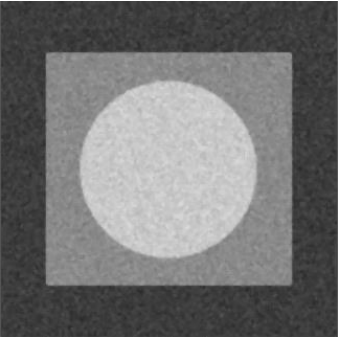
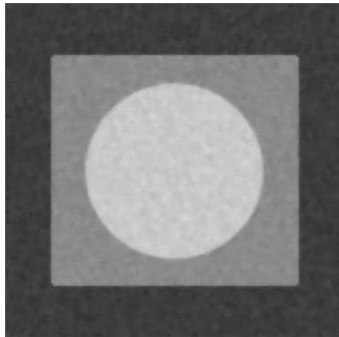
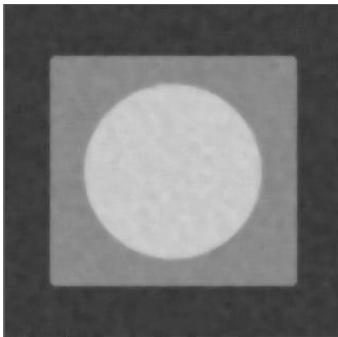
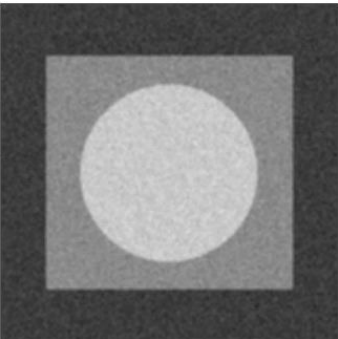
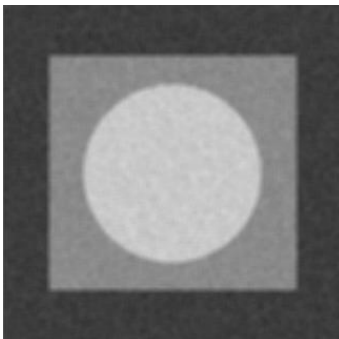
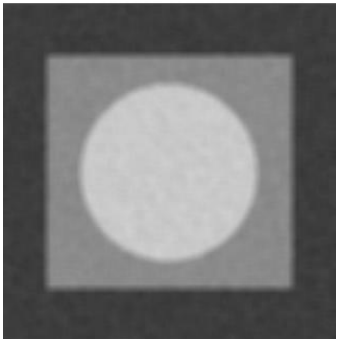


Fig0504(i)(salt-pepper-noise).jpg

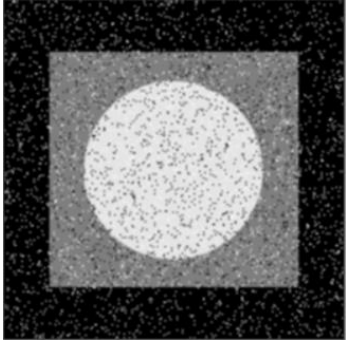
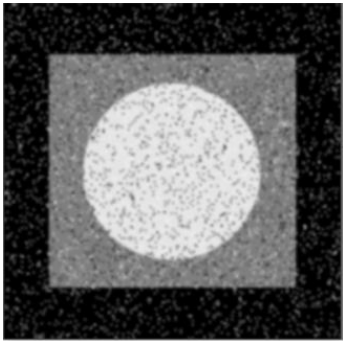
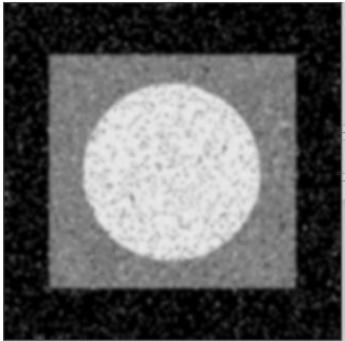
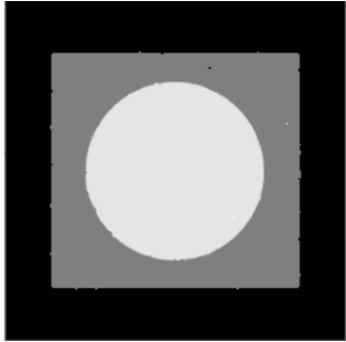
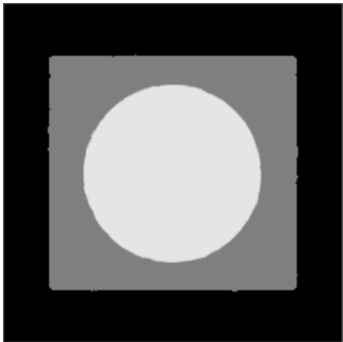
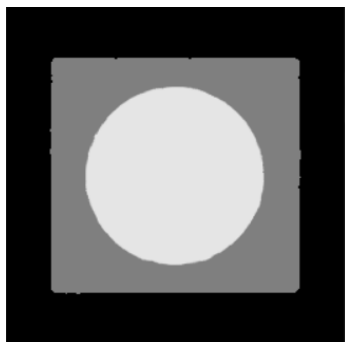
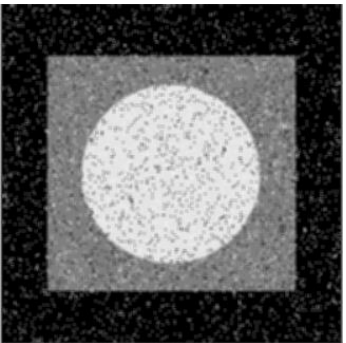
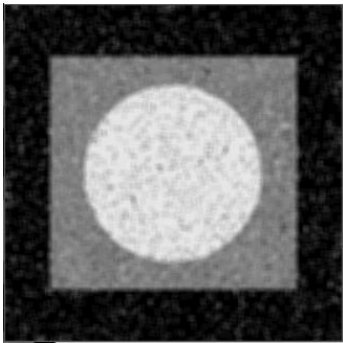
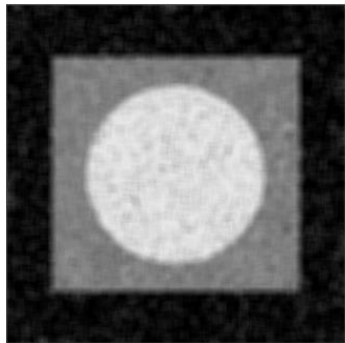
“원본 이미지”에 Smoothing 필터 적용

종류 \ mask size	3 * 3	5 * 5	7 * 7	분석
[Gaussian filtering]				<ul style="list-style-type: none"> - average filter보다 블러링이 덜하다. - mask size가 커질수록 edge가 명확하지 않다.
[median filtering]				<ul style="list-style-type: none"> - mask size가 커져도 edge가 명확하다. - 원본 색감을 유지한다. - gaussian, average보다 smoothing이 덜하다.
[average filtering]				<ul style="list-style-type: none"> - 블러링이 제일 심하다. - mask size가 커질수록 edge가 명확하지 않다.

“Gaussian noise”에 Smoothing 필터 적용

종류 \ mask size	3 * 3	5 * 5	7 * 7	분석
[Gaussian filtering]				<ul style="list-style-type: none"> - 자신과 가까운 부분에 가중치를 두기 때문에 Gaussian noise를 가장 잘 Smoothing 한다. - mask size가 커질수록 edge가 명확하지 않다. - average filter 보다는 edge가 명확하다.
[median filtering]				<ul style="list-style-type: none"> - mask size가 커져도 edge가 구분된다. - 경계 부분은 노이즈를 잘 제거하는 것 같으나, 중심 부분은 덜 제거된다. - 하지만 여전히 noise가 완벽하게 제거되지 않는다.
[average filtering]				<ul style="list-style-type: none"> - 블러링이 제일 심하다. - mask size가 커질수록 edge가 명확하지 않다.

“Salt-and-pepper noise”에 Smoothing 필터 적용

종류 \ mask size	3 * 3	5 * 5	7 * 7	분석
[Gaussian filtering]				<ul style="list-style-type: none"> - 오히려 average filter보다 노이즈 제거를 못 한다. - 하지만 average보다 edge 구분은 잘 된다. - mask를 키워도 노이즈 제거에 효과적이지 않다.
[median filtering]				<ul style="list-style-type: none"> - mask size가 커져도 edge가 구분된다. - salt and pepper 노이즈가 완벽히 제거된다. - 3*3 에서 제거되지 않은 노이즈가 7*7 에서 제거되었다. - 그러나 원본 이미지보다 edge가 울긋불긋하다.
[average filtering]				<ul style="list-style-type: none"> - 블러링이 제일 심하다. - gaussian 보다 노이즈 제거를 잘한다. - 그러나 mask를 키울수록 edge가 명확하지 않다.

Smoothing 필터 및 noise filtering 이미지2

원본 이미지

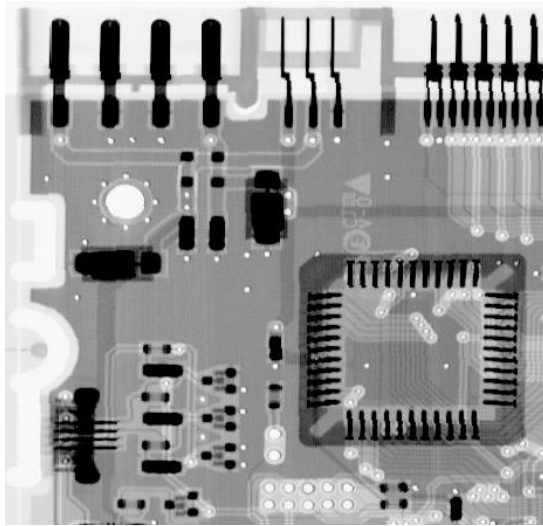


Fig0507(a)(ckt-board-orig).jpg

gaussian 노이즈 이미지

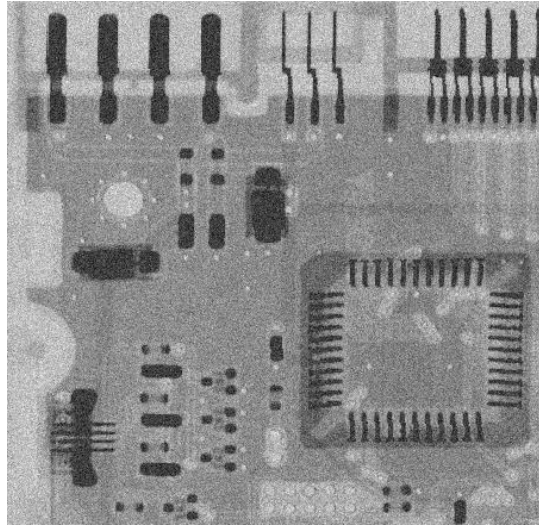


Fig0513(a)(ckt_gaussian_var_1000_mean_0).jpg

salt and pepper 노이즈 이미지

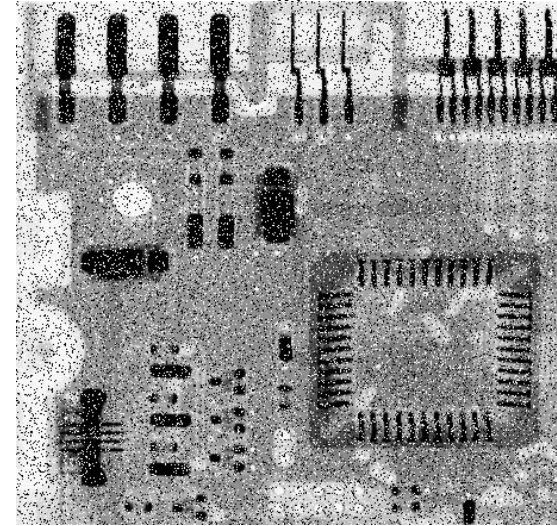
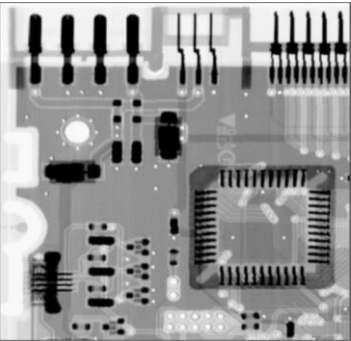
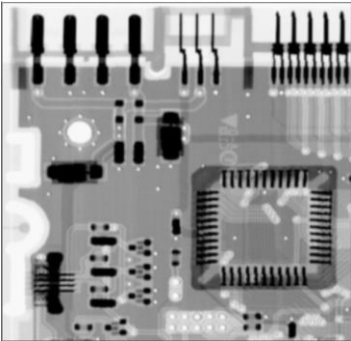
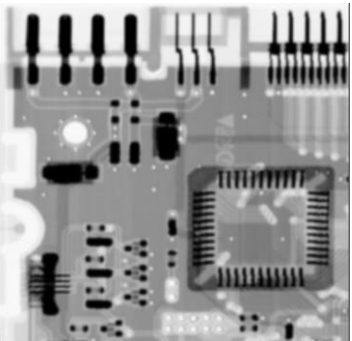
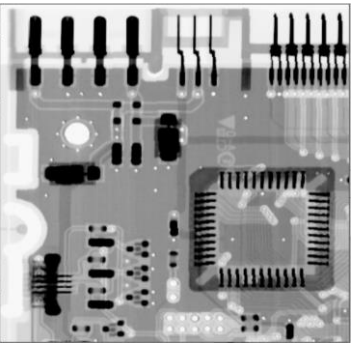
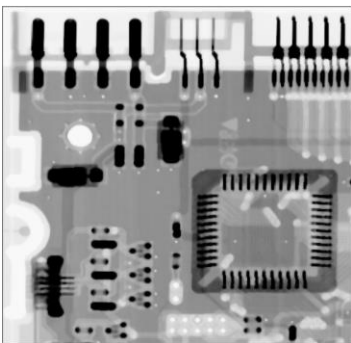
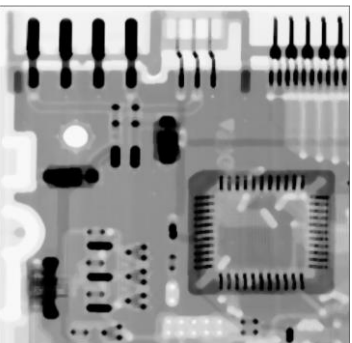
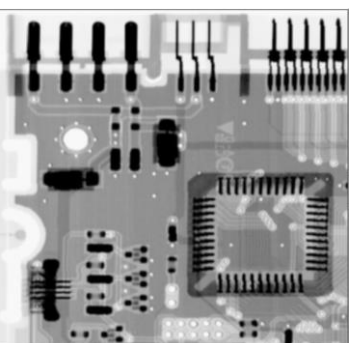
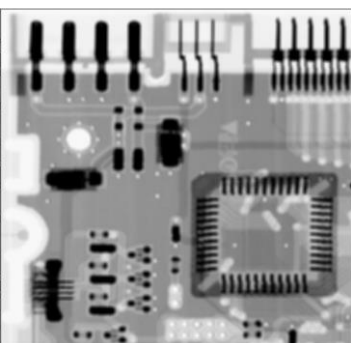

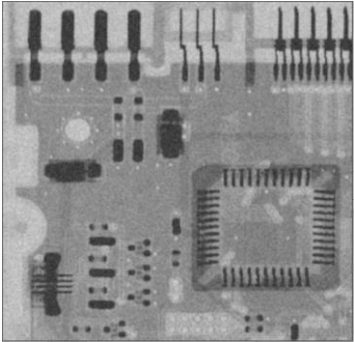
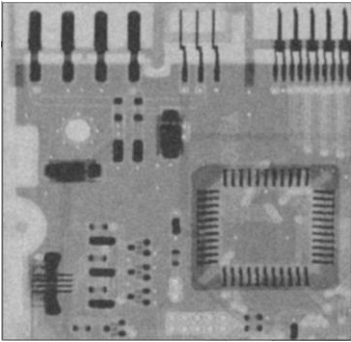
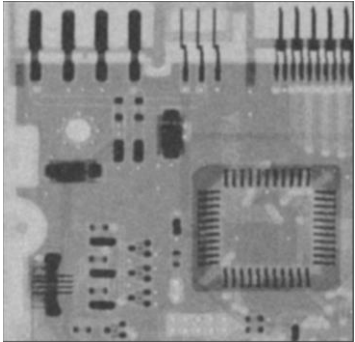
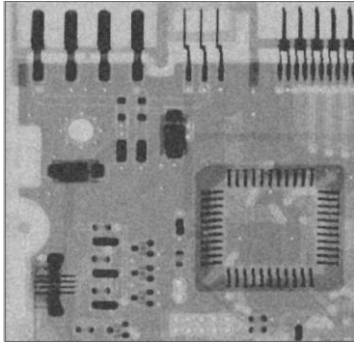
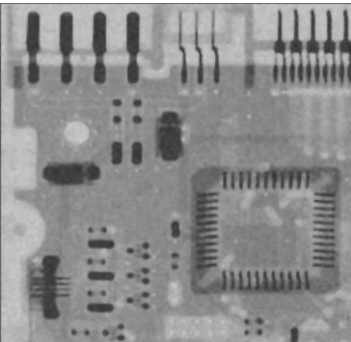
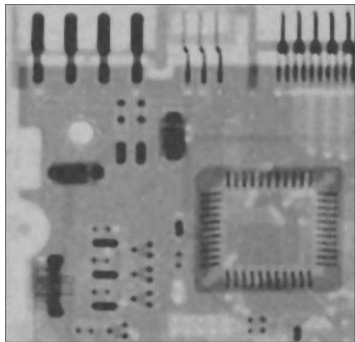
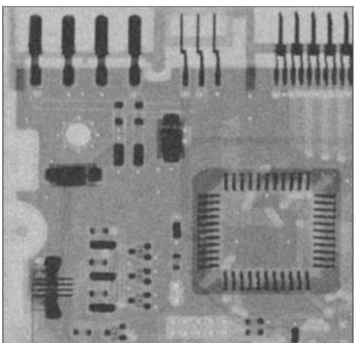
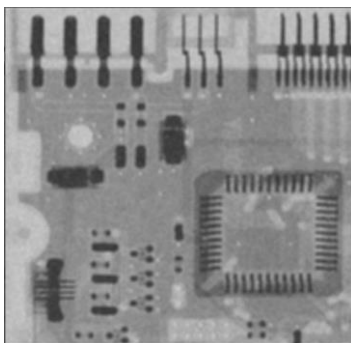
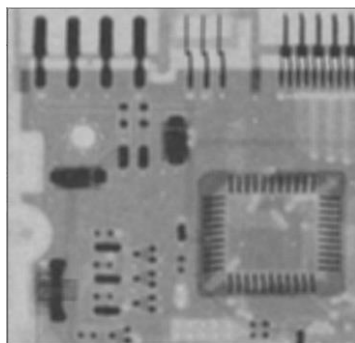


Fig0510(a)(ckt-board-saltpep-prob.pt05).jpg

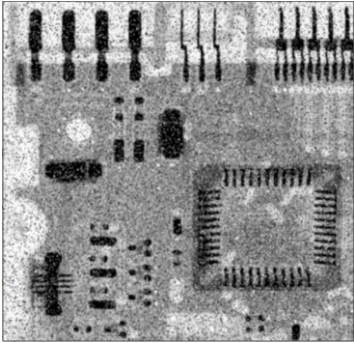
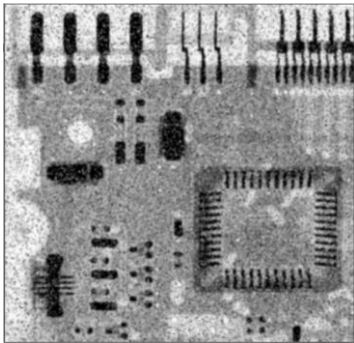
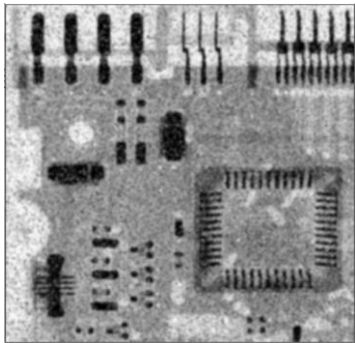
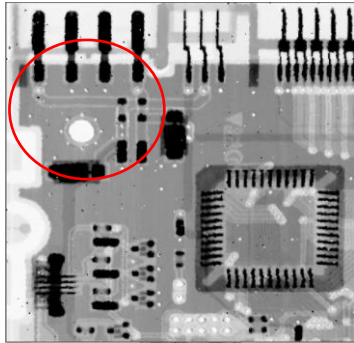
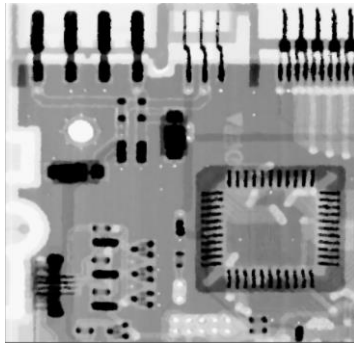
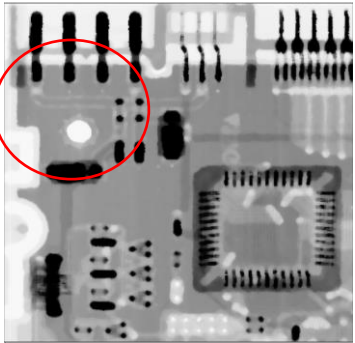
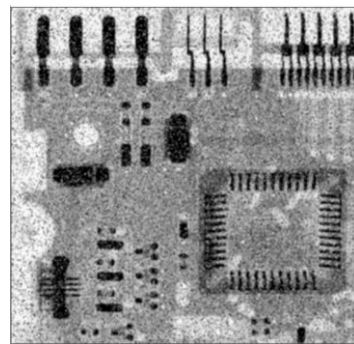
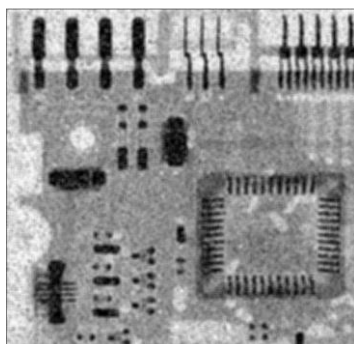
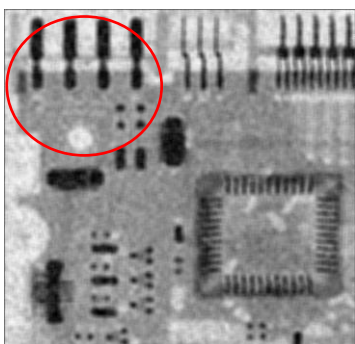
“원본 이미지”에 Smoothing 필터 적용

종류 \ mask size	3 * 3	5 * 5	7 * 7	분석
[Gaussian filtering]				<ul style="list-style-type: none"> - average filter보다 블러링이 덜하다. - mask size가 커질수록 edge가 불명확하며, 이미지가 연해진다. - 색감 변화가 많은 부분에 블러링이 더 많다.
[median filtering]				<ul style="list-style-type: none"> - mask size가 커져도 edge가 명확하다. - 원본 이미지의 명도를 유지한다. - 원본 이미지보다 약간의 smoothing이 보인다.
[average filtering]				<ul style="list-style-type: none"> - 블러링이 제일 심하다. - mask size가 커질수록 edge가 불명확하다. - 색감 변화의 밀집 여부와 관계없이 전체적으로 블러링이 일어난다.

“Gaussian noise”에 Smoothing 필터 적용

종류 \ mask size	3 * 3	5 * 5	7 * 7	분석
[Gaussian filtering]				<ul style="list-style-type: none"> - 가장 노이즈가 잘 제거된다. - mask size가 커질수록 edge가 불명확하며, 뿌옇게 보인다.
[median filtering]				<ul style="list-style-type: none"> - 7*7 mask에서 원본 이미지보다 smoothing이 많이 일어났다. - 명도는 유지하나, 여전히 noise가 완벽하게 제거되지 않는다.
[average filtering]				<ul style="list-style-type: none"> - 블러링이 제일 심하다. - mask size가 커질수록 edge가 불명확하다. - median보다 noise를 잘 제거한다.

“Salt-and-pepper noise”에 Smoothing 필터 적용

종류 \ mask size	3 * 3	5 * 5	7 * 7	분석
[Gaussian filtering]				<ul style="list-style-type: none"> - average보다 edge는 명확하나 노이즈는 잘 제거하지 못한다. - mask를 키워도 노이즈 제거에 효과적이지 않다.
[median filtering]				<ul style="list-style-type: none"> - mask size가 커져도 edge가 구분된다. - salt and pepper 노이즈가 완벽히 제거된다. - 하지만 7*7에서 노이즈가 아닌 하얀 점들까지 제거한다.
[average filtering]				<ul style="list-style-type: none"> - 블러링이 제일 심하다. - gaussian 보다 노이즈 제거를 잘한다. - 그러나 mask를 키울수록 edge가 명확하지 않다. - 7*7 에서 median 필터가 지운 하얀 점의 흔적이 남아 있다.

Smoothing 필터 및 noise filtering 성능 분석 및 결론

성능 분석

Gaussian filtering – 가우시안 필터 계산 및 convolution
median filtering – sorting
average filtering – convolution



실행 시간

average < gaussian < median

노이즈 제거

원본 이미지 – median filtering
Gaussian noise – Gaussian filtering
Salt and pepper noise – median filtering



노이즈 제거 효과

average < gaussian < median

“ 실행시간만을 고려한다면 average filter를 사용”

“노이즈 제거 성능만을 고려한다면 median filter를 사용”

“ 빠른 속도와 노이즈 제거 성능을 모두 고려한다면 Gaussian filter를 사용”

Sharpening: Highboost filtering 테스트

HighBoost Filtering- 소스코드

```
import cv2
import numpy as np

def convolution(img, size, mask):

    if len(img.shape) == 3: # 컬러 이미지
        H, W, C = img.shape
    else: # 흑백 이미지
        img = np.expand_dims(img, axis=-1)
        H, W, C = img.shape

    pad = size // 2
    out = np.zeros((H + pad * 2, W + pad * 2, C), dtype=np.float)
    out[pad: pad + H, pad: pad + W] = img.copy().astype(np.float)

    print(mask)

    tmp = out.copy()

    for i in range(H):
        for j in range(W):
            for k in range(C):
                out[pad + i, pad + j, k] = np.sum(mask * tmp[i: i + size, j: j + size, k])

    # 0~255사이의 값으로 변환
    out = np.clip(out, 0, 255)
    out = out[pad: pad + H, pad: pad + W].astype(np.uint8)

    return out
```

```
img = cv2.imread('Fig0327(a)(tungsten_original).jpg')

A = 1.2

mask1 = np.array([[0, -1, 0],
                  [-1, A+4, -1],
                  [0, -1, 0]])
mask2 = np.array([[-1, -1, -1],
                  [-1, A+8, -1],
                  [-1, -1, -1]])

result = convolution(img, 3, mask1)
result2 = convolution(img, 3, mask2)

A = 1.5

mask3 = np.array([[0, -1, 0],
                  [-1, A+4, -1],
                  [0, -1, 0]])
mask4 = np.array([[-1, -1, -1],
                  [-1, A+8, -1],
                  [-1, -1, -1]])

result3 = convolution(img, 3, mask3)
result4 = convolution(img, 3, mask4)

cv2.imshow('mask1_1.2', result)
cv2.imshow('mask2_1.2', result2)
cv2.imshow('mask1_1.5', result3)
cv2.imshow('mask2_1.5', result4)
cv2.waitKey(0)
cv2.destroyAllWindows()
```


High-Boost Filtering 원본 이미지 및 Mask

	A = 1.2	A = 1.5
Mask 1	$\begin{bmatrix} 0. & -1. & 0. \\ -1. & 5.2 & -1. \\ 0. & -1. & 0. \end{bmatrix}$	$\begin{bmatrix} 0. & -1. & 0. \\ -1. & 5.5 & -1. \\ 0. & -1. & 0. \end{bmatrix}$
Mask 2	$\begin{bmatrix} -1. & -1. & -1. \\ -1. & 9.2 & -1. \\ -1. & -1. & -1. \end{bmatrix}$	$\begin{bmatrix} -1. & -1. & -1. \\ -1. & 9.5 & -1. \\ -1. & -1. & -1. \end{bmatrix}$

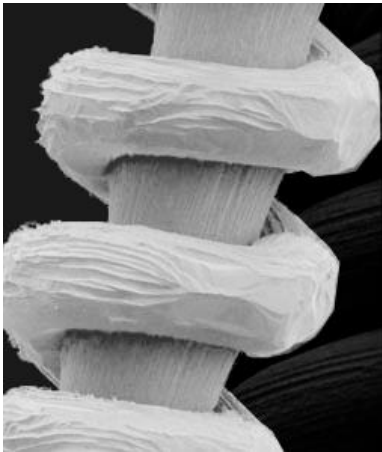
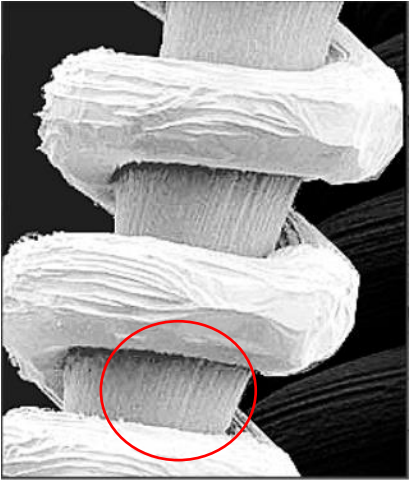
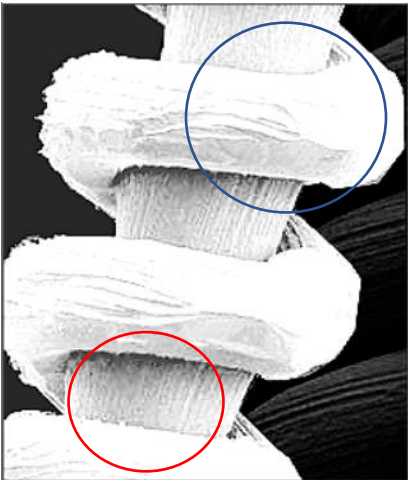
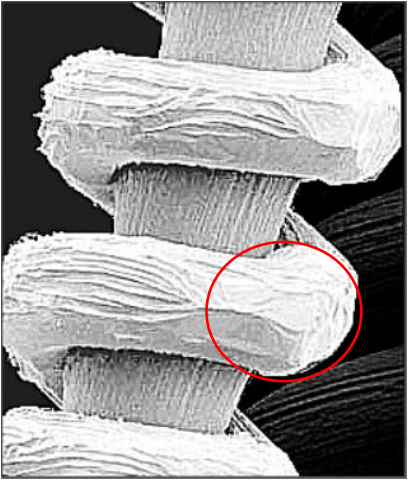
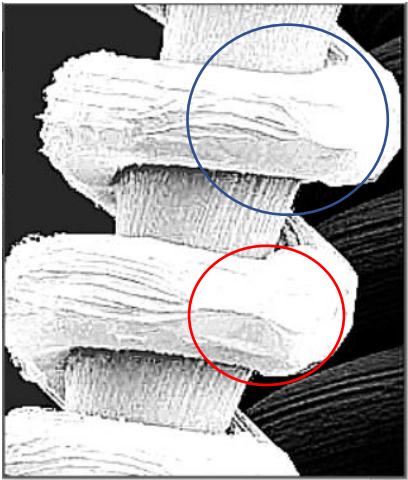


Fig0327(a)(tungsten_original).jpg



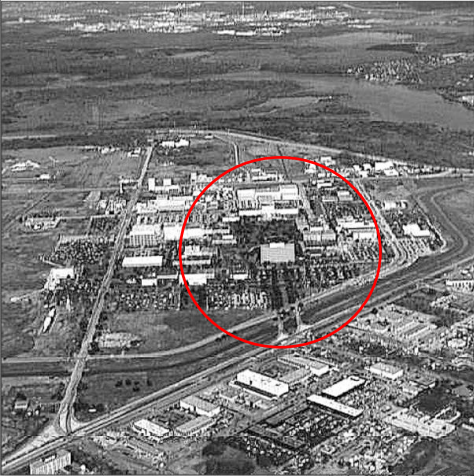

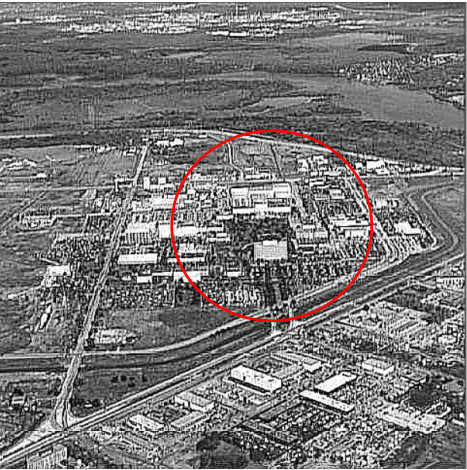

Fig0525(a)(aerial_view_no_turb).jpg

“이미지1” 에 High-Boost Filtering 적용

종류 \ A 값	1.2	1.5	분석
Mask 1			<div> <ul style="list-style-type: none"> - Mask 1은 자신과 가까운 부분만 비교하므로 Mask 2보다 edge가 흐릿하다. - Mask 2는 Mask 1보다 더 넓은 범위와 비교하므로 edge가 선명하다. </div>
Mask 2			

- Mask 1은 자신과 가까운 부분만 비교하므로 Mask 2보다 edge가 흐릿하다.
 - Mask 2는 Mask 1보다 더 넓은 범위와 비교하므로 edge가 선명하다.
- A 값이 커질수록 원본 이미지가 밝아져 A 1.2에서는 선명히 보이지 않았던 edge를 검출할 수 있다.
 - 하지만, A 값이 커질수록 A 1.2에서는 선명히 보였던 부분이 A 1.5에서는 보이지 않는다.
 - A 값이 1.2일 때 edge가 더 잘 보인다.

“이미지2” 에 High-Boost Filtering 적용

종류 \ A 값	1.2	1.5	분석
Mask 1			<ul style="list-style-type: none">- 원본 이미지가 어두우므로, 밝기가 높은 A 1.5일 때 더 edge가 잘 보인다.- 원본 이미지에서 어둡게 보였던 부분의 edge는 Mask 2에서 더 선명히 보인다.- Mask 1보다 Mask 2가 더 밝고 edge가 선명하다.
Mask 2			

Color space에서 smoothing

Gaussian filtering in color – 소스코드

```
import cv2
import numpy as np
from PIL import Image
import math
```

```
def gaussian1D(sigma):
    a = sigma

    if a % 2 == 0: #mask size가 짝수일 경우 1을 키운다
        a+=1

    a2 = a/2
    filter_size = np.array(range(-int(a/2),int(a/2)+1))
    result = np.zeros(len(filter_size))

    for i in range(len(filter_size)):
        x = i-a2
        result[i] = float(np.exp(-(x**2)/(2*sigma**2)) / (2*3.14*sigma**2)) #가우시안 공식

    total = np.sum(result)
    result/=total

    return result
```

```
def gaussian2D(sigma):
    # 두 벡터를 외적하여 x,y 2차원 mask를 생성
    result = np.outer(gaussian1D(sigma),gaussian1D(sigma))
    total = np.sum(result)
    result/=total

    return result
```

```
def convolution_color(array, f):
    a = (len(f)-1)//2
    # boundary를 0으로 채운다
    img_pad = np.pad(array, ((a,a),(a,a),(0,0)), 'constant', constant_values=0)

    #convolution을 위해 2번 뒤집는다
    mask = np.rot90(f)
    mask = np.rot90(mask)

    #color img 이므로 3차원으로 배열 설정
    result_img = np.zeros((len(array),len(array[0]),3))
    result_img = result_img.astype('float32')

    for i in range(len(result_img)):
        for j in range(len(result_img[0])):
            result_img[i][j][0] = np.sum(img_pad[i:i+len(f), j:j+len(f),0]*mask)
            result_img[i][j][1] = np.sum(img_pad[i:i+len(f), j:j+len(f),1]*mask)
            result_img[i][j][2] = np.sum(img_pad[i:i+len(f), j:j+len(f),2]*mask)

    return result_img
```

```
def real_convolution_color(array, sigma):
    return convolution_color(array, gaussian2D(sigma))
```

```
img = Image.open('Salt&pepper noise.png')

img_arr = np.asarray(img)
img_arr = img_arr.astype('float32')

img_result = real_convolution_color(img_arr, 20)
img_result = img_result.astype('uint8')
img_result = Image.fromarray(img_result)

img_result.show()
```

```
[[0.09555231 0.10678153 0.10678153]
 [0.10678153 0.11933039 0.11933039]
 [0.10678153 0.11933039 0.11933039]]
```

[3*3 mask]

```
[[0.03402125 0.03685478 0.03835885 0.03835885 0.03685478]
 [0.03685478 0.0399243 0.04155364 0.04155364 0.0399243 ]
 [0.03835885 0.04155364 0.04324948 0.04324948 0.04155364]
 [0.03835885 0.04155364 0.04324948 0.04324948 0.04155364]
 [0.03685478 0.0399243 0.04155364 0.04155364 0.0399243 ]]
```

[5*5 mask]

```
[[0.01730548 0.01839811 0.01916458 0.01955972 0.01955972 0.01916458
 0.01839811]
 [0.01839811 0.01955972 0.02037459 0.02079467 0.02079467 0.02037459
 0.01955972]
 [0.01916458 0.02037459 0.02122341 0.02166099 0.02166099 0.02122341
 0.02037459]
 [0.01955972 0.02079467 0.02166099 0.02210759 0.02210759 0.02166099
 0.02079467]
 [0.01955972 0.02079467 0.02166099 0.02210759 0.02210759 0.02166099
 0.02079467]
 [0.01916458 0.02037459 0.02122341 0.02166099 0.02166099 0.02122341
 0.02037459]
 [0.01839811 0.01955972 0.02037459 0.02079467 0.02079467 0.02037459
 0.01955972]]
```

[7*7 mask]

median filtering in color – 소스코드

```
import numpy as np
import cv2
from matplotlib import pyplot as plt
from PIL import Image, ImageFilter
```

```
img = cv2.imread('Salt&pepper noise.png')
height, width, channel = img.shape
```

```
out1 = np.zeros((height + 2, width + 2, channel), dtype=np.float)
out1[1: 1 + height, 1: 1 + width] = img.copy().astype(np.float)
temp1 = out1.copy()
```

```
out2 = np.zeros((height + 4, width + 4, channel), dtype=np.float)
out2[2: 2 + height, 2: 2 + width] = img.copy().astype(np.float)
temp2 = out2.copy()
```

```
for i in range(height):
    for j in range(width):
        for k in range(channel):
            out1[1 + i, 1 + j, k] = np.median(temp1[i:i + 5, j:j + 5, k])
            hybrid_temp1 = np.median((temp2[i, j, k], temp2[i + 1, j + 1, k],
            temp2[i + 2, j + 2, k], temp2[i + 3, j + 3, k], temp2[i + 4, j + 4, k]))
            hybrid_temp2 = np.median((temp2[i + 4, j, k], temp2[i + 3, j + 1,
            k], temp2[i + 2, j + 2, k], temp2[i + 1, j + 3, k], temp2[i, j + 4, k]))
            hybrid_temp3 = np.median((temp2[i + 5, j:j + 5, k]))
            out2[2 + i, 2 + j, k] = np.median((hybrid_temp1, hybrid_temp2,
            hybrid_temp3))
```

```
out1 = out1[1:1 + height, 1:1 + width].astype(np.uint8)
out2 = out2[2:2 + height, 2:2 + width].astype(np.uint8)
```

```
cv2.imshow('result', out1)
cv2.imshow('result2', out2)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

average filtering in color – 소스 코드

```
import cv2
import numpy as np
from PIL import Image
```

```
def average_filter(img, size):
```

```
    if len(img.shape) == 3: # 컬러 이미지
        H, W, C = img.shape
    else: # 흑백 이미지
        img = np.expand_dims(img, axis=-1)
        H, W, C = img.shape
```

```
    pad = size // 2
    out = np.zeros((H + pad * 2, W + pad * 2, C), dtype=np.float)
    out[pad: pad + H, pad: pad + W] = img.copy().astype(np.float)
```

```
    # mask 생성
    mask = np.ones((size, size))/size**2
    print(mask)
```

```
    tmp = out.copy()
```

```
    for i in range(H):
        for j in range(W):
            for k in range(C):
                out[pad + i, pad + j, k] = np.sum(mask * tmp[i: i + size, j: j + size, k])
```

```
    # 0~255사이의 값으로 변환
    out = np.clip(out, 0, 255)
    out = out[pad: pad + H, pad: pad + W].astype(np.uint8)
```

```
    return out
```

```
img = cv2.imread('Fig0507(a)(ckt-board-orig).jpg')
out = average_filter(img, 5)
```

```
cv2.imshow("result", out)
cv2.waitKey(0)
cv2.destroyAllWindows()
```


Color space에서 smoothing 이미지

Gaussian noise 이미지



Gaussian noise.png

Lena noise 이미지



Lena_noise.png

salt and pepper 노이즈 이미지



Salt&pepper noise.png


“Gaussian noise이미지”에 Color Smoothing 필터 적용

종류 \ mask size	3 * 3	5 * 5	7 * 7	분석
[Gaussian filtering]				<ul style="list-style-type: none"> - 노이즈도 잘 제거되고, 눈과 같은 섬세한 부분의 edge도 잘 살아있다. - 더 큰 mask를 사용하면 더 많은 노이즈를 제거할 것으로 예상된다.
[median filtering]				<ul style="list-style-type: none"> - 노이즈가 잘 제거되나, edge가 뭉툭해졌다.
[average filtering]				<ul style="list-style-type: none"> - 눈과 같은 부분이 블러링이 심해서 경계선이 명확하지 않다. - mask size가 커질수록 edge가 명확하지 않다.

“Lena noise 이미지”에 Smoothing 필터 적용

종류 \ mask size	3 * 3	5 * 5	7 * 7	분석
[Gaussian filtering]				<ul style="list-style-type: none"> - 7*7 mask에서도 노이즈가 선명히 보인다. - 블러링 현상은 적다. - 노이즈 제거의 관점에서 본다면 좋은 filtering은 아니다.
[median filtering]				<ul style="list-style-type: none"> - 노이즈 제거의 관점에서 본다면 잘 제거되므로 좋은 filtering이다. - 그러나 mask 크기에 따라 경계가 뭉툭해진다.
[average filtering]				<ul style="list-style-type: none"> - 블러링이 제일 심해서 경계선이 흐려진다. - mask size가 커질수록 edge가 명확하지 않다.

“Salt-and-pepper noise”에 Smoothing 필터 적용

종류 \ mask size	3 * 3	5 * 5	7 * 7	분석
[Gaussian filtering]				<ul style="list-style-type: none"> - 7*7 mask에서도 노이즈가 선명히 보이지만, 블러링 현상은 적다. - average filter랑 비슷하게 노이즈를 제거하지 못한다. - mask를 키워도 노이즈 제거에 효과적이지 않다.
[median filtering]				<ul style="list-style-type: none"> - salt and pepper 노이즈가 잘 제거된다. - 그러나 mask의 크기에 따라 edge 경계가 원본과 다르게 울퉁불퉁하게 보인다.
[average filtering]				<ul style="list-style-type: none"> - gaussian과 비슷하게 노이즈 제거를 못 한다. - 그러나 mask를 키울수록 edge가 명확하지 않다. - 전체적으로 비슷한 정도로 smoothing 된다.

Edge Detection

1차 미분 연산자 (Sobel, Prewitt) – 소스코드

```
import cv2
import numpy as np
from PIL import Image
```

```
def sobel_X():
    kernel = np.array([[ -1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
    return kernel
```

```
def sobel_Y():
    kernel = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]])
    return kernel
```

```
def prewitt_X():
    kernel = np.array([[ -1, 0, 1], [-1, 0, 1], [-1, 0, 1]])
    return kernel
```

```
def prewitt_Y():
    kernel = np.array([[1, 1, 1], [0, 0, 0], [-1, -1, -1]])
    return kernel
```

```
img = cv2.imread('Fig0327(a)(tungsten_original).jpg')

out_x = gradient(img, 3, sobel_X())
out_y = gradient(img, 3, sobel_Y())

merged = out_x+out_y

out_x2 = gradient(img, 3, prewitt_X())
out_y2 = gradient(img, 3, prewitt_Y())

merged2 = out_x2+out_y2

cv2.imshow("sobel", merged)
cv2.imshow("prewitt", merged2)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
def gradient(img, size, mask):
    if len(img.shape) == 3:
        H, W, C = img.shape
    else:
        img = np.expand_dims(img, axis=-1)
        H, W, C = img.shape

    pad = size // 2
    out = np.zeros((H + pad * 2, W + pad * 2, C), dtype=np.float)
    out[pad: pad + H, pad: pad + W] = img.copy().astype(np.float)

    print(mask)

    tmp = out.copy()

    for i in range(H):
        for j in range(W):
            for k in range(C):
                out[pad + i, pad + j, k] = np.sum(mask * tmp[i: i + size, j: j + size, k])

    out = np.clip(out, 0, 255)
    out = out[pad: pad + H, pad: pad + W].astype(np.uint8)

    return out
```


LoG - 소스코드

```
import cv2
import numpy as np
from PIL import Image
```

```
def log_3x3():
    kernel = np.array([[1, 1, 1], [1, -8, 1], [1, 1, 1]])
    return kernel
```

```
def log_5x5():
    kernel = np.array([[0,0,1,0,0], [0,1,2,1,0], [1,2,-16,2,1], [0,1,2,1,0], [0,0,1,0,0]])
    return kernel
```

```
def LoG_filter(img, size, mask):
    if len(img.shape) == 3:
        H, W, C = img.shape
    else:
        img = np.expand_dims(img, axis=-1)
        H, W, C = img.shape

    pad = size // 2
    out = np.zeros((H + pad * 2, W + pad * 2, C), dtype=np.float)
    out[pad: pad + H, pad: pad + W] = img.copy().astype(np.float)

    print(mask)

    tmp = out.copy()

    for i in range(H):
        for j in range(W):
            for k in range(C):
                out[pad + i, pad + j, k] = np.sum(mask * tmp[i: i + size, j: j + size, k])
    out = np.clip(out, 0, 255)
    out = out[pad: pad + H, pad: pad + W].astype(np.uint8)

    return out
```

```
img = cv2.imread('Fig0327(a)(tungsten_original).jpg')
```

```
out1 = LoG_filter(img, 3, log_3x3())
out2 = LoG_filter(img, 5, log_5x5())
```

```
cv2.imshow("log_3x3", out1)
cv2.imshow("log_5x5", out2)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Edge detection 원본 이미지

이미지 1

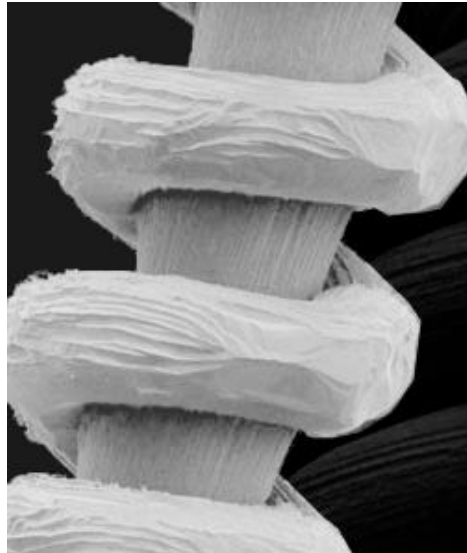


Fig0327(a)(tungsten_original).jpg

이미지 2

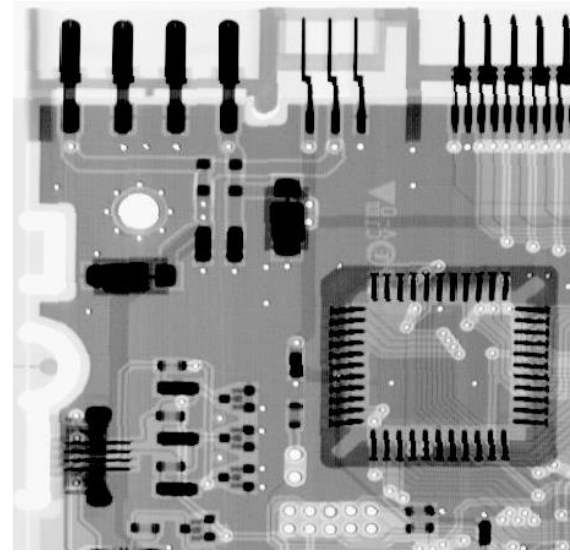




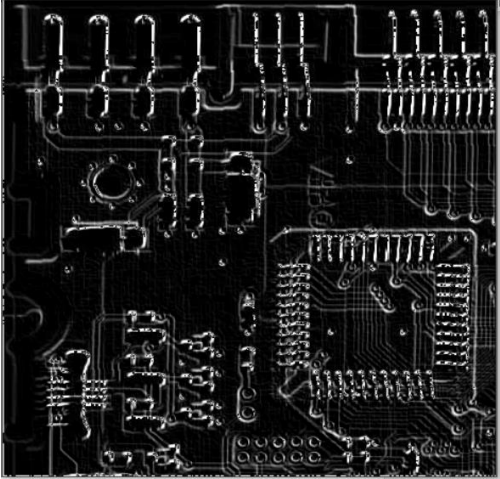
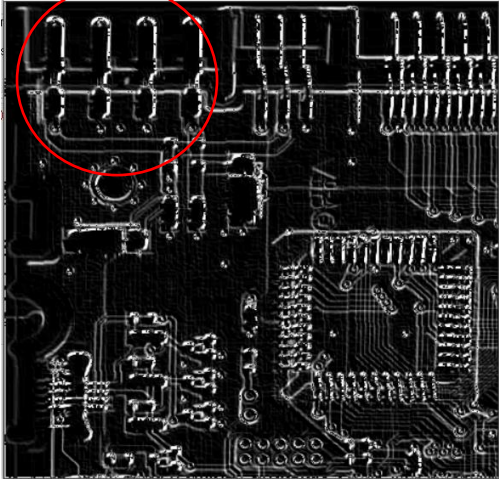
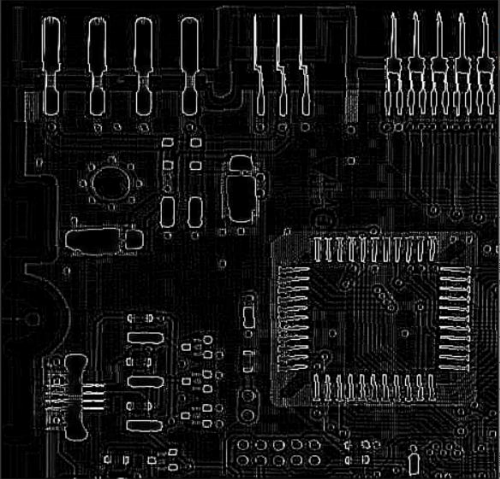
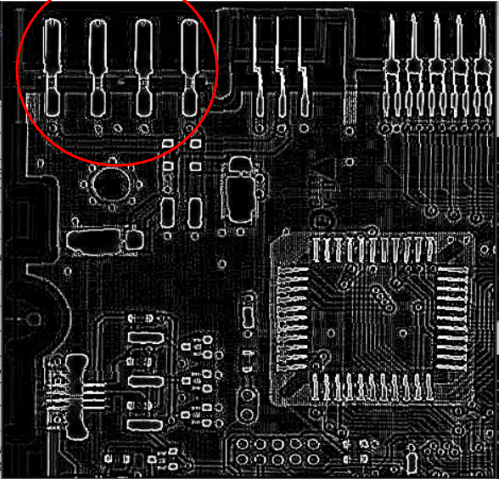


Fig0507(a)(ckt-board-orig).jpg

“이미지1” 에 Edge detection 적용

종류 \ A 값	prewitt	sobel	분석
Gradient			<div> <ul style="list-style-type: none"> - Gradient에서, prewitt보다 sobel의 detection 결과가 더 선명하다. - 두 종류 모두 대각선 edge를 잘 찾아낸다. </div>
LoG			
	3 * 3	5 * 5	

“이미지2” 에 Edge detection 적용

종류 \ A 값	prewitt	sobel	분석
Gradient			<ul style="list-style-type: none"> - Gradient에서 prewitt보다 sobel의 edge가 더 선명하다. - Gradient에서 black 색상인 경우 edge를 뽑아내지 못한다. - Gradient가 더 입체감이 있다.
LoG			<ul style="list-style-type: none"> - 가장 edge를 잘 뽑아낸 것은 LoG 5*5이다. - LoG의 5*5에서는 가느다란 전선도 edge로 찾아낸다. - LoG로 얻은 edge가 Canny와 비슷한 결과를 만들어낸다.
	3 * 3	5 * 5	

옵션) Canny edge operator

Canny edge operator 소스 코드 및 원본 이미지

```
import cv2
import numpy as np
```

```
img = cv2.imread('Fig0327(a)(tungsten_original).jpg')
```

```
out = cv2.Canny(img, 50, 200)
out2 = cv2.Canny(img, 100, 200)
out3 = cv2.Canny(img, 0, 200)
```

```
cv2.imshow('50 200', out)
cv2.imshow('100 200', out2)
cv2.imshow('0 200', out3)
```

```
cv2.waitKey(0)
cv2.destroyAllWindows()
```

이미지 1

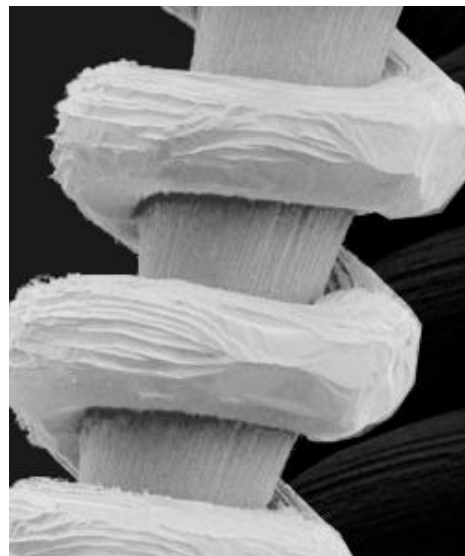


Fig0327(a)(tungsten_original).jpg

이미지 2

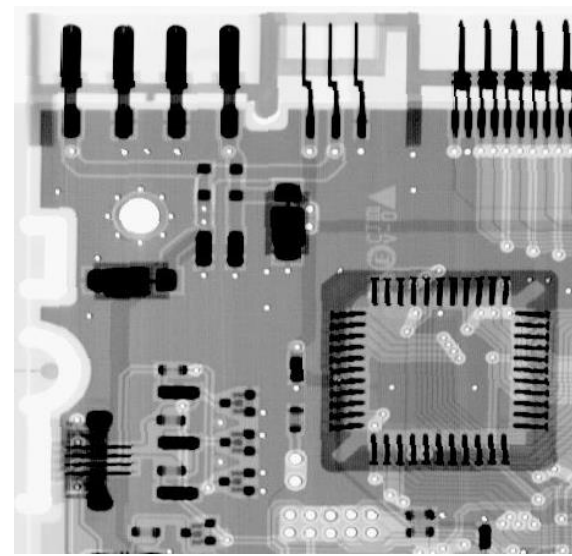


Fig0507(a)(ckt-board-orig).jpg

“이미지1” 에 Canny edge operator 적용



100 200 (min, max)



50 200



0 200



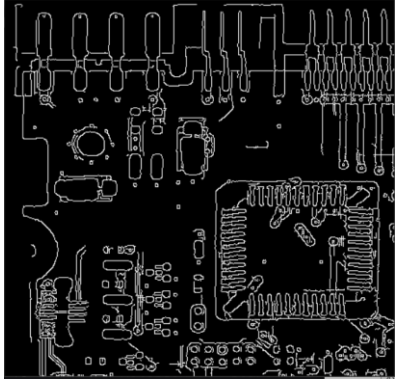
50 100



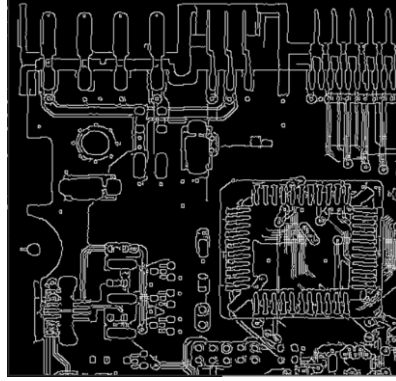
50 250

- min threshold 값이 낮아질 수록 더 많은 edge를 찾는다.
- max threshold 값이 높아질 수록 더 적은 edge를 찾는다.

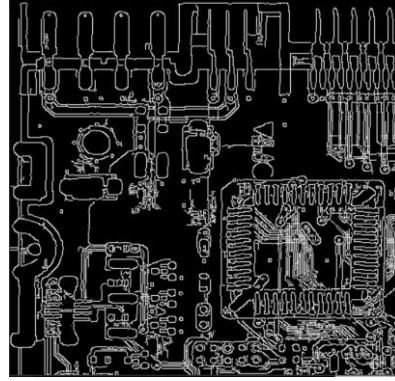
“이미지2” 에 Canny edge operator 적용



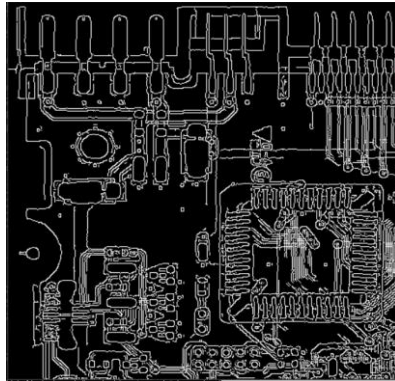
100 200 (min, max)



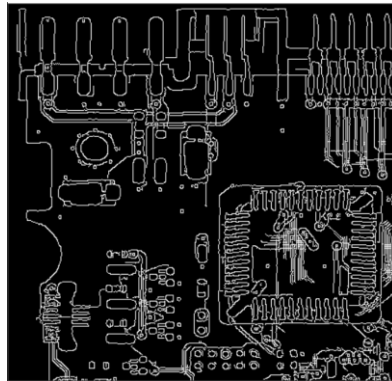
50 200



0 200



50 100



50 250

- min threshold 값이 낮아질 수록 더 많은 edge를 찾는다.
- max threshold 값이 높아질 수록 더 적은 edge를 찾는다.

감사합니다