Project #2

-Audio & Image Signal Processing 2-

2016135011 신민영

1-1. Plot the $x[n]$ and |X(w)|. Generate $v[n], \boldsymbol{where\ v\ is\ whith\ gaussian\ noise\ N(0, 0.02)}$

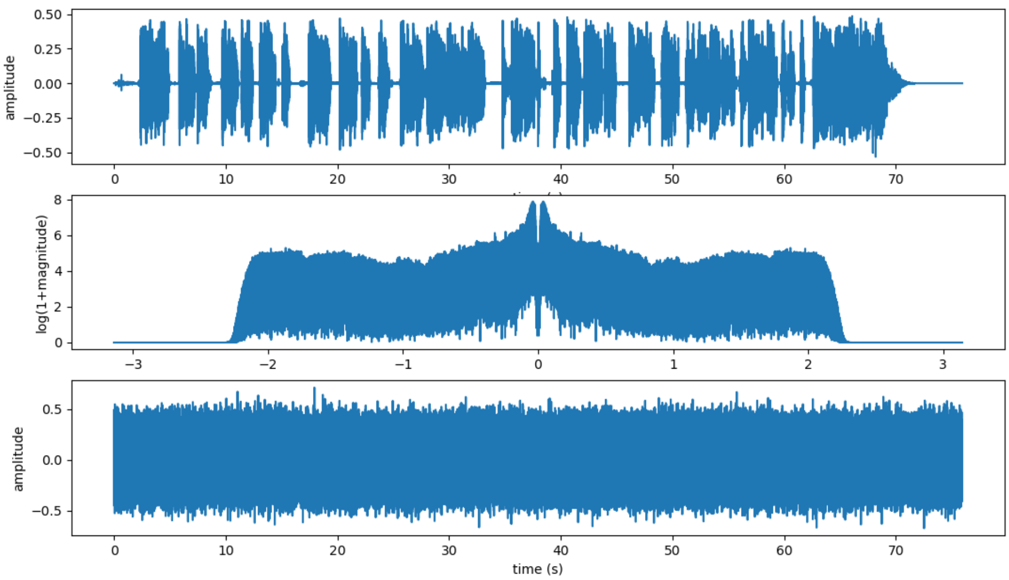

Figure 1. x[n](위), |X(w)|(중간, 로그 스케일), 그리고 $v[n]$(아래)

Python code

```
#extract x[n]
data, samplerate = librosa.load('x[n].wav')   #16000 samplerate
times = np.arange(len(data))/float(samplerate)

#Listen original sound
# sd.play(data, samplerate)
# sd.wait()

#DFT of original sound
X = np.fft.fftshift(np.fft.fft(data))
w = np.linspace(-np.pi, +np.pi, len(X))

#Gaussian Noise
v = np.random.normal(0, np.sqrt(0.02), size=data.shape)
V = np.fft.fftshift(np.fft.fft(v))
```

Python의 librosa 라이브러리를 활용하여 wav 음원을 읽어올 수 있다. samplerate는 **16000Hz**인 것 역시 확인할 수 있다. x[n]의 경우 말과 말 사이의 빈 시간들을 확인할 수 있고, 주파수 대역에

서 음원은 **저주파** 대역에 신호가 몰려 있는 것을 볼 수 있다. 이는 남성의 일반적인 음역대인 50Hz~500Hz라는 사실과 부합한다. 가우시안 노이즈의 경우 $\sigma^2 = 0.02$ 이기 때문에 **numpy.random.normal** 함수를 사용해서 노이즈를 생성할 수 있었다. 그 결과 신호의 진폭이 최대 0.6정도까지의 신호가 가우시안 분포를 토대로 발생한 것을 알 수 있다. 직접 음원을 들어보면 지지직 하는 소리로 노이즈가 발생한 것을 들을 수 있다.

1-2. Plot real part of $H_d(w)$.

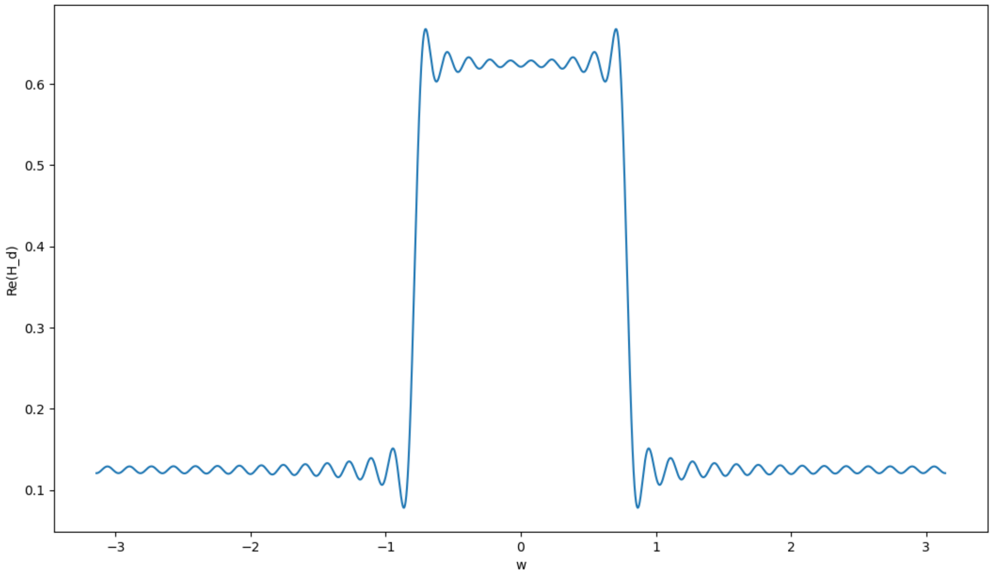

figure 2. Real part of $H_d(w)$

Python code

```
#truncated filter
N = 39
n = np.arange(0, N)
h_d = []
for i in n:
    if i==0:
        h_d.append(1/4)
    else:
        h_d.append(1 / 4 * np.sin(np.pi / 4 * i) / np.pi * 4 / i)
h_d = np.pad(h_d, (0, len(data)-N), 'constant', constant_values=0)

#DFT of h_d
H_d = np.fft.fftshift(np.fft.fft(h_d))
```

Sinc 의 오른쪽 절반을 값으로 가지는 $h_d[n]$을 파이썬 리스트에 저장하였다. 이 때 n = 0 일 때는 분모가 0이 되기 때문에, 극한에서의 로피탈 정리 특성을 이용하면 $h_d[0] = \frac{w_c}{\pi}$ 가 되는 것을 이용하였다. 또한 $h_d[n]$ 는 truncated function이기 때문에, n >= N 일 때는 0으로 zero-padding을 해주었다. ***figure 2***를 통해 truncated impulse response의 문제점을 확인할 수 있다. $cutoff\ freq.$인 $\frac{\pi}{4}$

부근에 큰 진동이 발생하고 passband에서도 리플이 생기는 **Gibbs phenomenon**을 볼 수 있다. 이는 무한 범위의 desired freq. response를 유한하게 잘랐기 때문에 발생한 현상이며, N 값이 커질 수록 이런 현상은 줄어들 것이다. 하지만 실제로 N 값을 무한대로 키우는 것은 불가능하기 때문에 windowing을 통해서 이를 줄여야 함을 알 수 있다.

1-3. Plot the |H(w)| and discuss about the result.

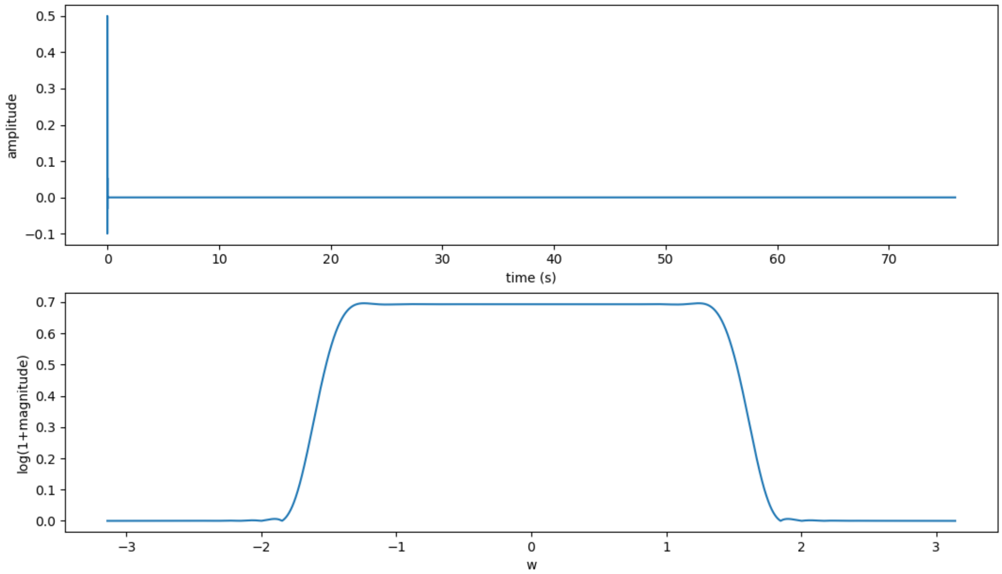

figure 3. h[n](위), H(w)(아래, 로그 스케일)

Python code

```
#Hanning windowing, wc = pi/2
h = []
for i in n:
    if i == (N-1)/2:
        h.append(1/2 * 0.5*(1- np.cos(2*np.pi/(N-1) * i)))
    else :
        h.append(1/2 * np.sin(np.pi / 2 * (i - (N-1)/2))/np.pi*2/(i - (N-
1)/2) * 0.5*(1- np.cos(2*np.pi/(N-1) * i)))

h = np.pad(h, (0, len(data)-N), 'constant', constant_values=0)

#DFT of h
H = np.fft.fftshift(np.fft.fft(h))
```

figure 3에서 h[n]은 time domain이기 때문에 n= 0, 1, ..., N-1 까지만 0이 아닌 값을 가지는 것을 볼 수 있다. H(w)의 로그스케일 그림을 보면 LPF의 형태를 보이는 것을 확인 할 수 있다. Cutoff frequency인 4000Hz를 맞춰주기 위하여 $w_c = \frac{\pi}{2}$ 값을 넣어주었다. 이것은 왜냐하면 samplerate 가 16000이고, $f = \frac{F}{F_s} = \frac{4000}{16000} = \frac{1}{4}, w_c = 2\pi f = \frac{\pi}{2}$ 으로 계산되기 때문이다. Hanning windowing을 하는

과정에서 **Casual** 한 시스템을 만들기 위하여 $\frac{N-1}{2}$만큼 +n 방향으로 shift를 해주어야 한다. 따라서 $h[n] = w[n] * h_d[n - \frac{N-1}{2}]$ 으로 impulse response를 만들 수 있다. 1-2에서 했던 것과 같은 방법으로 나머지 부분은 zero-padding을 하고 fft 과정을 거치면 ***figure 3***의 아래와 같이 H(w) 결과를 확인할 수 있다. $\frac{\pi}{2}$ 부근에서 cutoff freq.를 확인할 수 있으면 1-2와 비교하면 확연하게 **Gibbs phenomenon이 사라진 것을 볼 수 있다**. 그리고 main lobe와 side lobe 역시 관찰할 수 있다.

1-4. Plot the $v_f[n]$ and $|V_f(w)|$.

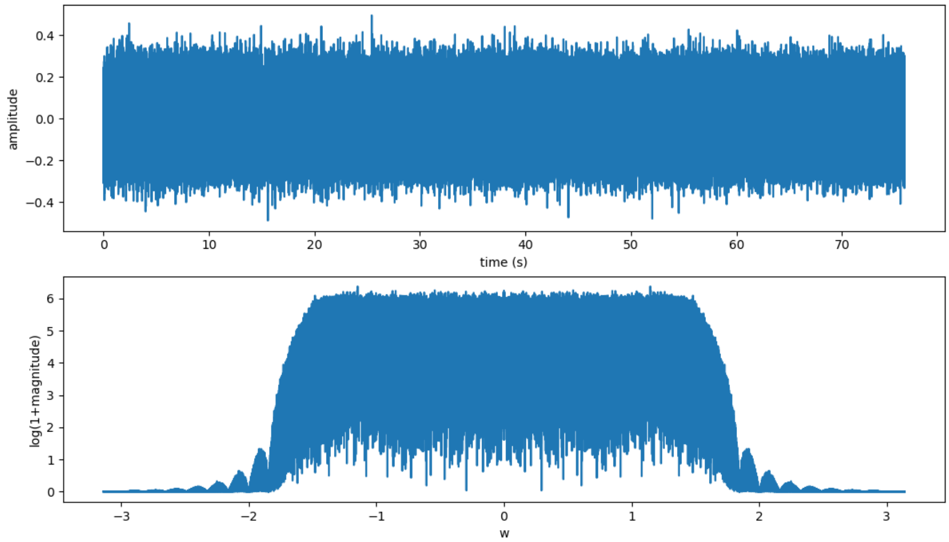

figure 4. $v_f[n]$ (*위*), $V_f(w)$ (*아래, 로그스케일*)

Python code

```
#Filtering gaussian noise
V_f = H * V

#IFFT to get filtered noise
v_f= np.fft.ifft(np.fft.ifftshift(V_f)).real

#x_d[n] and X_d(w)
x_d = data + v_f
X_d = np.fft.fftshift(np.fft.fft(x_d))

#generate x_d[n]
sd.play(x_d, samplerate)
sd.wait()
```

H(w)를 $V_f(w)$ 과 곱하면 필터링 된 신호의 주파수 영역 성분을 얻을 수 있다. 이를 역푸리에 변환하면 필터링 된 노이즈를 확인할 수 있다. 예상했던 대로, $\frac{\pi}{2}$ 이상의 값을 필터링한 것을 $V_f(w)$

를 보면 확인할 수 있다. 또한 시간 도메인에서 분석한다면 ***figure 1***에서 최대 진폭인 0.5~0.6이었던 것에 반해 필터링을 한 결과 최대 진폭인 0.3~0.4로 줄어든 것을 확인할 수 있었다. 실제 소리를 들어보면, 노이즈가 작게 들림으로써 필터링이 됐음을 알 수 있다.

1-5. Design your own filter $H_2(w)$. Plot $|H_2(w)|\ and < H_2(w)$.

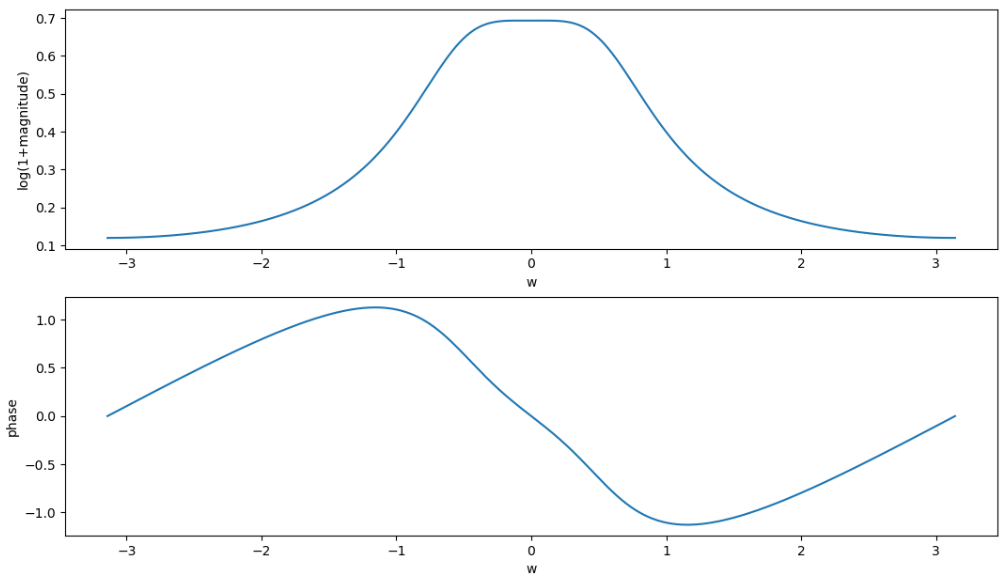

figure 5. My own filter magnitude(위, 로그스케일), phase(아래)

```
E:\maysh\anaconda3\python.exe E:/maysh/PycharmProjects/Lecture/DSP2/audio.py
126.64169030566984
97.95759681038405

Process finished with exit code 0
```

figure 6. 1-4번의 PSNR 스코어(위), My own filter의 PSNR 스코어(아래)

Python code

```
#My own filter design
b = 0.3069
r = 0.6
theta = 0.5
H2 = b / ((1- r * np.exp(1j*(theta - w)))*(1-r*np.exp(-1j*(theta+w))))

# Filtering signal
X_d2 = X_d * H2
x_d2 = np.fft.ifft(np.fft.ifftshift(X_d2)).real

#evalute
print(np.sqrt(np.sum(np.square(v_f)))) # 1-4 , hanning windowing
print(np.sqrt(np.sum(np.square(data - x_d2)))) # 1-5, my own filter
```

Pole-zero placement를 이용하여 **2차 order LPF**를 설계하였다. 남성의 목소리가 800Hz를 넘기지 않을 것으로 예상하였기 때문에, $\boldsymbol{-3dB\ at\ w = \frac{\pi}{10}}$, $\boldsymbol{-20dB\ at\ w = \pi}$ 가 되게끔 하였다. 이 때 $w = \frac{\pi}{10} \rightarrow F = 800Hz, w = \pi \rightarrow 8000Hz$ 로 samplerate를 고려하여 생각해야 한다. 즉, 정리하면

$$\boldsymbol{H(w) = \frac{b}{(1 - pe^{-jw})(1 - p^{*}e^{-jw})}, \qquad p = re^{j\theta}}$$

$$\boldsymbol{H(w)|_{w=\frac{\pi}{10}} = \frac{1}{\sqrt{2}},\ \ H(w)|_{w=\pi} = \frac{1}{10}, H(0) = 1}$$

로 pole-zero placement 방식을 이용하여 필터를 설계하면, $\boldsymbol{b = 0.3069, r = 0.6, theta = 0.5\ rad}$ 로 계수를 근사하여 연립방정식의 해를 구할 수 있다. 이는 교재에 Figure 5.4.2 중에 아래와 같은 케이스에 해당한다.

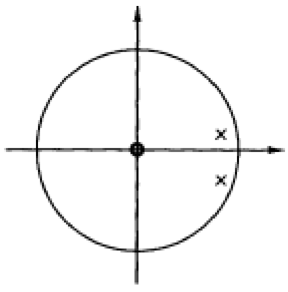

figure 7. pole-zero diagram(Figure 5.4.2 in Textbook 350 page)

이것으로 PSNR을 구해보면 97점으로 Hanning Window를 활용하여 측정한 스코어인 126점보다 낮아진 것을 확인할 수 있었다. 다만, 점수는 scipy 라이브러리, wav 라이브러리, librosa 라이브러리 등 wav 파일을 읽을 때 사용한 라이브러리의 종류에 따라 값의 크기가 달라진다. 따라서 Hanning window 보다 직접 설계한 필터의 **점수가 낮아졌다**는 데에서 의의가 있다.

마지막으로 실제 음원을 들어보면 확실히 노이즈가 1-4에서 했던 것보다 줄어들었음을 알 수 있다. 따라서 0~800Hz 이외의 영역의 가우시안 노이즈가 대부분 필터링 되고, 0~800Hz는 남성의 목소리와 남은 가우시안 노이즈가 함께 출력되고 있음을 알 수 있다.

2-1. Do fourier transform and plot $|I(w_u, w_v)|$

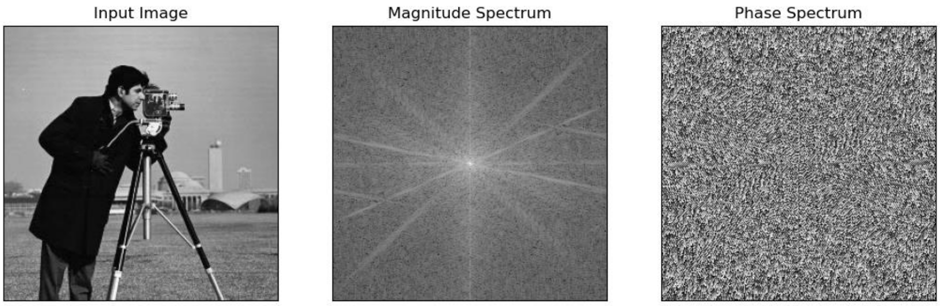

figure 8. cameramen Fourier transform

Python code

```
#Image load
img = cv2.imread('cameraman.jpg',0)

#FFT original image
f = np.fft.fft2(img)
fshift = np.fft.fftshift(f)
```

간단하게 cv2 라이브러리를 이용하여 이미지를 읽고, 2차원 fourier transform 및 fftshift를 하였다.

2-2 Show the ideal low-pass filter.

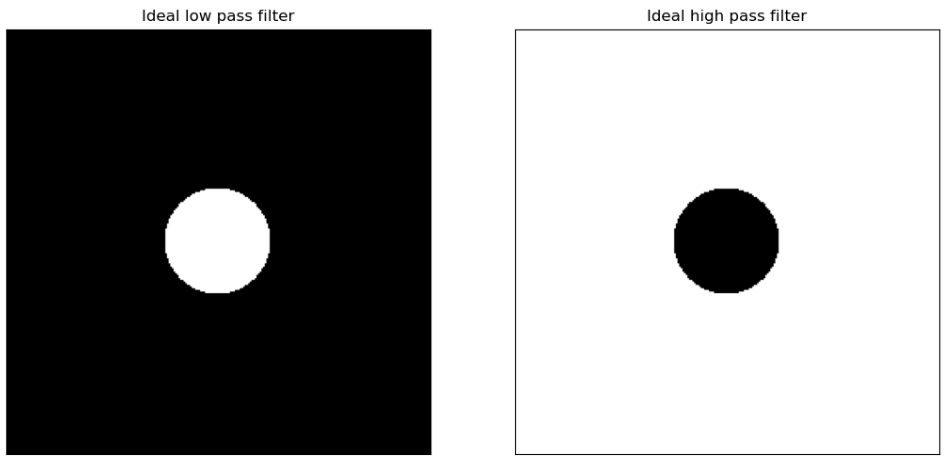

figure 9. Ideal LPF(왼쪽), Ideal HPF(오른쪽)

Python code

```
#H_lpf(wx, wy)
H_lpf = []
H_hpf = []
origin_row = (fshift.shape[0]-1) // 2
origin_col = (fshift.shape[1]-1) // 2
w_row = np.linspace(-np.pi, +np.pi, fshift.shape[0])
w_col = np.linspace(-np.pi, +np.pi, fshift.shape[1])

#make Ideal LPF and HPF
for i in range(0, fshift.shape[0]):
    temp_lpf = []
    temp_hpf = []
    for j in range(0, fshift.shape[1]):
        if np.sqrt(np.square(w_row[i] - w_row[origin_row]) +
np.square(w_col[j] - w_col[origin_col])) <= (np.pi/4) :
            temp_lpf.append(1)
            temp_hpf.append(0)
        else:
            temp_lpf.append(0)
            temp_hpf.append(1)
    H_lpf.append(temp_lpf)
    H_hpf.append(temp_hpf)
```

LPF를 만들고, HPF는 1-LPF이기 때문에 동시에 만드는 작업을 진행했다. 유클리디안 거리가 D_0 보다 작은 거리에 있는 부분은 1로 만들고, 먼 부분은 0으로 만들기 때문에 LPF는 원 모양으로 출력될 것이고, HPF는 이와 반전으로 표시됨을 알 수 있다.

2-3 Plot the output image signal.

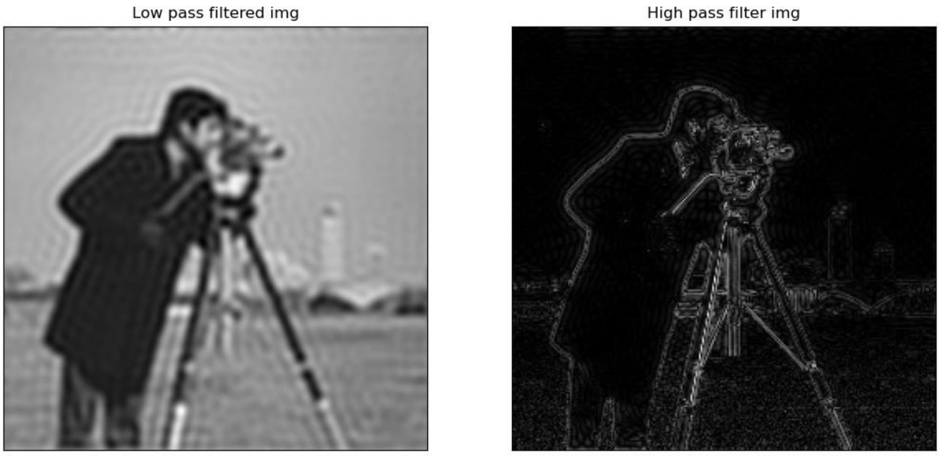

figure 10. LPF로 필터링한 그림(왼쪽), HPF로 필터링한 그림(오른쪽)

Python code

```
#Filtering by Ideal LPF and HPF
Y_lpf = fshift * H_lpf
Y_hpf = fshift * H_hpf
y_lpf = np.fft.ifft2(np.fft.ifftshift(Y_lpf)).real
y_hpf = np.fft.ifft2(np.fft.ifftshift(Y_hpf)).real
y_hpf = np.abs(y_hpf) / np.max(y_hpf)
```

2-2 에서 구한 필터의 Impulse response를 주파수 도메인에서 원본 이미지의 푸리에 변환과 곱하면 필터링을 할 수 있다. ***figure 10*** 에서 그 결과를 확인할 수 있다. LPF는 사진이 blur 처리된 것처럼 보이며, HPF는 이미지의 윤곽선을 보여주고 있다. 이를 통해서 Low frequency에서는 이미지의 전반적인 추상적 정보를 가지고 있으며(윤곽선 안의 내용들), High frequency는 이미지의 edge 정보를 가지고 있음을 알 수있다.

2-4. Generate gaussian filter and Laplacian filter.

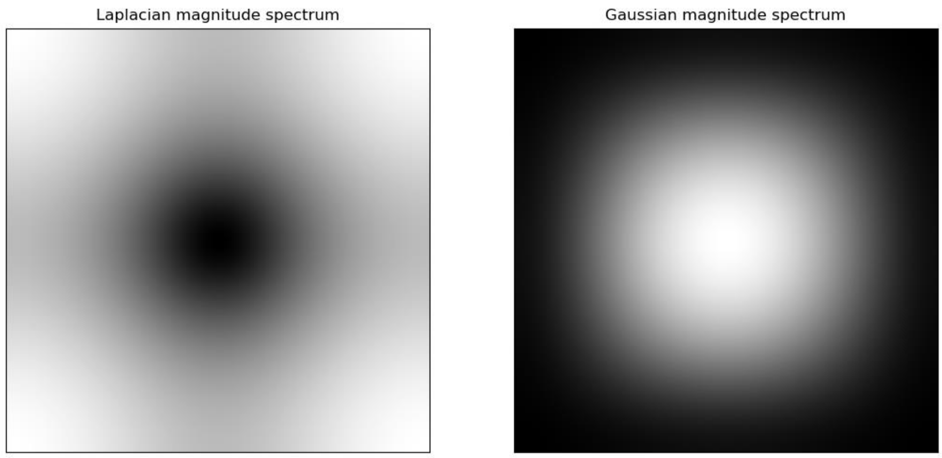

figure 11. 주파수 영역에서 그린 Laplacian filter(왼쪽), Gaussian Filter(오른쪽)

Python

```
#3x3 laplacian filter
laplacian = np.array([[0, -1, 0], [-1, 4, -1], [0, -1, 0]])

#Gausian filter with std.dev = 0.8, kernel size =3
sigma = 0.8
kernel_size = 3
gaussian = []
for i in range(0, kernel_size):
    temp = []
    for j in range(0, kernel_size):
        x = np.abs(i - (kernel_size // 2))
        y = np.abs(j - (kernel_size // 2))
        temp.append(1 / 2 / np.pi / sigma / sigma * np.exp(-(np.square(x) +
np.square(y)) / 2 / sigma / sigma))
    gaussian.append(temp)

#filtering by laplacian and gaussian filter
Y_laplacian = np.fft.fftshift(np.fft.fft2(expand(laplacian, len(img), 3)))
Y_gaussian = np.fft.fftshift(np.fft.fft2(expand(gaussian, len(img), 3)))
```

라플라시안 필터는 3x3의 필터를 이용하였다. 가우시안 필터의 경우는 $\boldsymbol{\sigma} = \mathbf{0.8}$인 3x3 필터를 만들었다. $\boldsymbol{\sigma} = \mathbf{0.8}$ 로 설정한 것은 필터 계수의 총합이 1이 되도록 하여 필터링 된 이미지의 신호가 과도하게 커지거나 작아지는 것을 막기 위함이다. ***figure 11***을 보면 Laplacian 필터는 저주파 대역을 통과시키지 않고 고주파 대역을 통과시키는 2-3의 HPF처럼 동작하고 있다. Gaussian filter는 저주파만을 통과시키는 2-3의 LPF처럼 동작하고 있다. 따라서 **Gaussian filter는 LPF, Laplacian filter는 HPF로 활용할 수 있다.**

2-5. Filter the noised image using LPF.

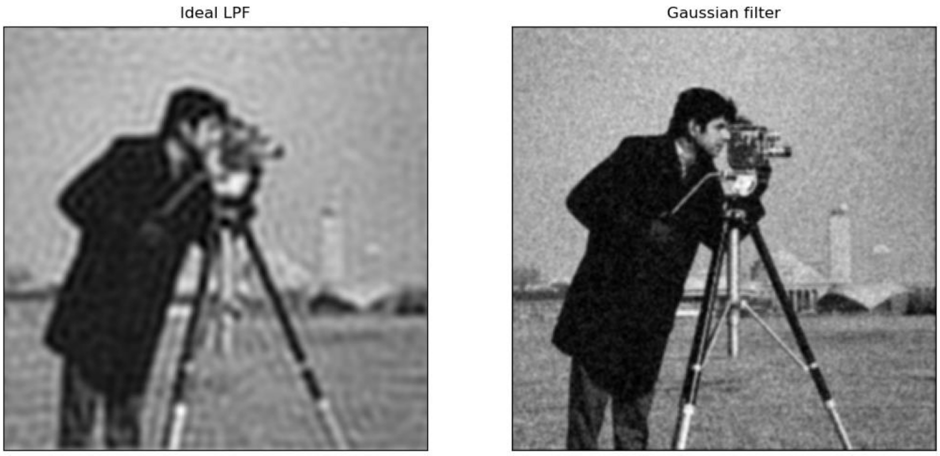

figure 12. noised image를 Ideal LPF로 필터링(왼쪽), Gaussian filter로 필터링(오른쪽)

```
E:\maysh\anaconda3\python.exe E:/maysh/PycharmProjects/Lecture/DSP2/image.py
22.481693292224058
22.968991540557045

Process finished with exit code 0
```

figure 13. Ideal LPF의 PSNR(위), Gaussian filter의 필터링(아래)

Python code. conv2 function

```python
def conv2(original, kernel, kernel_size):
    num_pad = kernel_size // 2;
    pd = ((num_pad, num_pad), (num_pad, num_pad))
    pad_img = np.pad(original, pd, 'constant', constant_values=(0))
    conv_img = []
    length = len(original)
    for i in range(0, length):
        temp = []
        for j in range(0, length):
            sum = 0
            for m in range(0, kernel_size):
                for n in range(0, kernel_size):
                    sum+=pad_img[i+m][j+n] * kernel[m][n]
            temp.append(sum)
        conv_img.append(temp)
    return conv_img
```

Python code. filtering

```
#filtering noised image by gaussian filter
noised_img = cv2.imread('noised_img.jpg',0)
noised_fshift = np.fft.fftshift(np.fft.fft2(noised_img))
Y_lpf_noised = noised_fshift * H_lpf
y_lpf_noised = np.fft.ifft2(np.fft.ifftshift(Y_lpf_noised)).real
y_gaussian_noised = conv2(noised_img, gaussian, 3)

#print PSNR score
psnr(img, y_lpf_noised)
psnr(img, y_gaussian_noised)
```

conv2 라는 함수를 직접 구현하였다. 먼저, 커널 사이즈에 따라서 zero-padding을 해준다. 그 뒤, 필터 윈도우를 따라가면서 convolution 합성곱을 진행하였다. filtering 코드는 noised image를 cv2로 읽어들인 뒤, ideal LPF와 Gaussian filter로 필터링을 한 것이다. 2-4에서도 언급했듯이, LPF는 Gaussian filter이며, 윈도우의 계수들 합이 1이 되도록 $\sigma = 0.8$, kernel_size = 3 으로 Gaussian filter를 만들어 사용하였다. 그 결과 ***figure 12***에서 볼 수 있듯이, 육안으로도 gaussian filter가 노이즈를 더 효과적으로 제거했으며, PSNR 점수 역시 Ideal LPF는 22.48, Gaussian filter는 22.96으로 Gaussian filter의 성능이 더 효과적임을 알 수 있다.

2-6. Filter the original image using HPF.

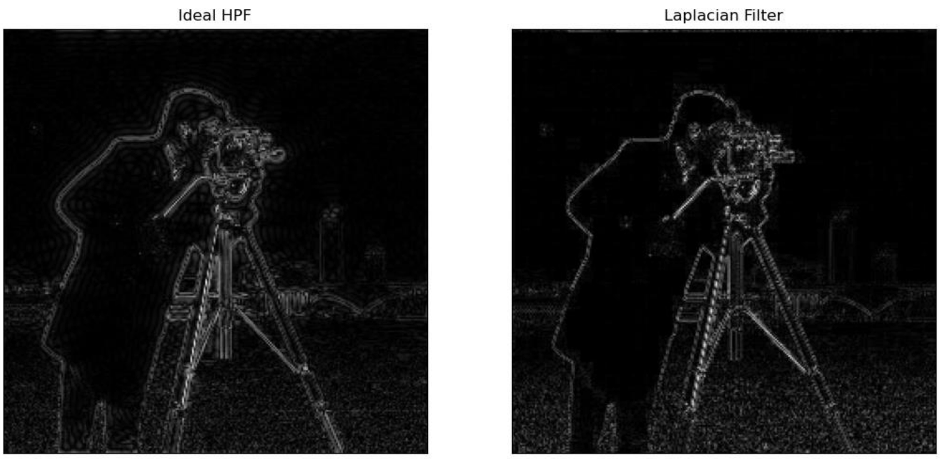

figure 14. Ideal HPF(왼쪽), Laplacian filter(오른쪽)

Python code. make filter.

```python
def expand(kernel, img_size, kernel_size):
    filter_expand = []
    for row in range(0, img_size):
        temp = []
        for col in range(0, img_size):
            if row < kernel_size and col < kernel_size:
                temp.append(kernel[row][col])
            else:
                temp.append(0)
        filter_expand.append(temp)
    return filter_expand
```

Python code. filtering

```python
Y_laplacian = np.fft.fftshift(np.fft.fft2(expand(laplacian, len(img), 3)))
* fshift
y_laplacian = np.fft.ifft2(np.fft.ifftshift(Y_laplacian)).real
y_laplacian = np.abs(y_laplacian) / np.max(y_laplacian)
```

Ideal HPF와 비슷한 Laplacian filter를 사용하여 필터링하고 결과를 비교해보았다. 주파수 영역에서의 원본 이미지와 필터를 곱하면 된다. 3x3 의 커널을 fft하여 256x256의 주파수 영역으로 변환하기 위해서는 zero padding을 해주어야 한다. 함수 expand는 커널을 확장하는 기능을 한다. 라플라시안 커널을 입력받아서, 원하는 사이즈가 되도록 zero-padding을 해준다. 그 뒤, 아래에 있는 filtering을 진행하면 된다. expand 함수가 반환하는 값을 2차원 fft해주고, 원본과 multiplication하면 ***figure 14***와 같은 그림을 얻을 수 있다. 결과를 비교하면 **Ideal HPF보다 Laplacian filter의 edge detection이 더욱 성공적**으로 이루어진 것을 확인할 수 있다. 이것은 Ideal HPF의 D_0 를 조절하여 cutoff frequency를 적절히 변경하면 Ideal HPF도 더욱 선명하게 edge detection할 수 있을 것이다.

Appendix a. audio.py(1번) 전체 코드

```
import numpy as np
import librosa, librosa.display

import matplotlib.pyplot as plt
import sounddevice as sd
import soundfile as sf

#extract x[n]
data, samplerate = librosa.load('x[n].wav')   #16000 samplerate
times = np.arange(len(data))/float(samplerate)

#Listen original sound
# sd.play(data, samplerate)
# sd.wait()

#DFT of original sound
X = np.fft.fftshift(np.fft.fft(data))
w = np.linspace(-np.pi, +np.pi, len(X))

#Gaussian Noise
v = np.random.normal(0, np.sqrt(0.02), size=data.shape)
V = np.fft.fftshift(np.fft.fft(v))

#Listen gaussian noised sound
# sd.play(data + v, samplerate)
# sd.wait()

#save sounds
sf.write('gaussian_noise_only.wav', v, samplerate, 'PCM_16')
sf.write('gaussian_noise_with_original_sound.wav', data+v, samplerate,
'PCM_16')

#plot x[n]
plt.figure('x[n], |X(w)| and v[n]')
plt.subplot(311)
plt.plot(times, data)
plt.xlabel('time (s)')
plt.ylabel('amplitude')

#plot |X(w)|
plt.subplot(312)
plt.plot(w, np.log(1+np.abs(X)))
plt.xlabel('w')
plt.ylabel('log(1+magnitude)')

#plot v[n]
plt.subplot(313)
plt.plot(times, v)
plt.xlabel('time (s)')
plt.ylabel('amplitude')
plt.show()

#truncated filter
N = 39
n = np.arange(0, N)
h_d = []
for i in n:
    if i==0:
```

```
        h_d.append(1/4)
    else:
        h_d.append(1 / 4 * np.sin(np.pi / 4 * i) / np.pi * 4 / i)
h_d = np.pad(h_d, (0, len(data)-N), 'constant', constant_values=0)

#DFT of h_d
H_d = np.fft.fftshift(np.fft.fft(h_d))

#plot Re(H_d(w))
plt.figure('Real part of H_d(w)')
plt.plot(w, np.real(H_d))
plt.xlabel('w')
plt.ylabel('Re(H_d)')
plt.show()

#Hanning windowing, wc = pi/2
h = []
for i in n:
    if i == (N-1)/2:
        h.append(1/2 * 0.5*(1- np.cos(2*np.pi/(N-1) * i)))
    else :
        h.append(1/2 * np.sin(np.pi / 2 * (i - (N-1)/2))/np.pi*2/(i - (N-
1)/2) * 0.5*(1- np.cos(2*np.pi/(N-1) * i)))

h = np.pad(h, (0, len(data)-N), 'constant', constant_values=0)

#DFT of h
H = np.fft.fftshift(np.fft.fft(h))

#plot h[n]
plt.figure('h[n] and |H(w)|')
plt.subplot(211)
plt.plot(times, h)
plt.xlabel('time (s)')
plt.ylabel('amplitude')

#plot |H(w)|
plt.subplot(212)
plt.plot(w, np.log(1+np.abs(H)))
plt.xlabel('w')
plt.ylabel('log(1+magnitude)')
plt.show()

#Filtering gaussian noise
V_f = H * V

#IFFT to get filtered noise
v_f= np.fft.ifft(np.fft.ifftshift(V_f)).real

#x_d[n] and X_d(w)
x_d = data + v_f
X_d = np.fft.fftshift(np.fft.fft(x_d))

#generate x_d[n]
# sd.play(x_d, samplerate)
# sd.wait()
sf.write('sound filtered by Hanning Window.wav', x_d, samplerate, 'PCM_16')

#plot v_f[n]
plt.figure('v_f[n] and |V_f(w)|')
```

```python
plt.subplot(211)
plt.plot(times, v_f)
plt.xlabel('time (s)')
plt.ylabel('amplitude')

#plot |V_f(w)|
plt.subplot(212)
plt.plot(w, np.log(1+np.abs(V_f)))
plt.xlabel('w')
plt.ylabel('log(1+magnitude)')
plt.show()

#My own filter design
b = 0.3069
r = 0.6
theta = 0.5
H2 = b / ((1- r * np.exp(1j*(theta - w)))*(1-r*np.exp(-1j*(theta+w))))

# Filtering signal
X_d2 = X_d * H2
x_d2 = np.fft.ifft(np.fft.ifftshift(X_d2)).real

#evalute
print(np.sqrt(np.sum(np.square(v_f)))) # 1-4 , hanning windowing
print(np.sqrt(np.sum(np.square(data - x_d2)))) # 1-5, my own filter

# sd.play(x_d2, samplerate)
# sd.wait()
sf.write('my_filter.wav', x_d2, samplerate, 'PCM_16')

#plot |H2(w)|
plt.figure('|H2(w)| and <H2(w)')
plt.subplot(211)
plt.plot(w, np.abs(H2))
plt.xlabel('w')
plt.ylabel('log(1+magnitude)')


#plot <H2(w)
plt.subplot(212)
plt.plot(w, np.angle(H2))
plt.xlabel('w')
plt.ylabel('phase')
plt.show()
```

Appendix b. image.py(2번) 전체 코드

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
import math

#PSNR metric
def psnr(original, contrast):
    mse = np.mean((original - contrast) ** 2)
    if mse == 0:
        return 100
    PIXEL_MAX = 255.0
    PSNR = 20 * math.log10(PIXEL_MAX / math.sqrt(mse))
    print(PSNR)
    return PSNR

#convolution in time domain
def conv2(original, kernel, kernel_size):
    num_pad = kernel_size // 2;
    pd = ((num_pad, num_pad), (num_pad, num_pad))
    pad_img = np.pad(original, pd, 'constant', constant_values=(0))
    conv_img = []
    length = len(original)
    for i in range(0, length):
        temp = []
        for j in range(0, length):
            sum = 0
            for m in range(0, kernel_size):
                for n in range(0, kernel_size):
                    sum+=pad_img[i+m][j+n] * kernel[m][n]
            temp.append(sum)
        conv_img.append(temp)
    return conv_img

#expand filter's kernel by zero-padding in time domain
def expand(kernel, img_size, kernel_size):
    filter_expand = []
    for row in range(0, img_size):
        temp = []
        for col in range(0, img_size):
            if row < kernel_size and col < kernel_size:
                temp.append(kernel[row][col])
            else:
                temp.append(0)
        filter_expand.append(temp)
    return filter_expand

#Image load
img = cv2.imread('cameraman.jpg',0)

#FFT original image
f = np.fft.fft2(img)
fshift = np.fft.fftshift(f)

#Plot image, magnitude spectrum and phase spectrum before filtering
magnitude_spectrum = 20*np.log(1+np.abs(fshift))
phase_spectrum = np.angle(fshift)
plt.subplot(131),plt.imshow(img, cmap = 'gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
```

```python
plt.subplot(132),plt.imshow(magnitude_spectrum, cmap = 'gray')
plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])
plt.subplot(133),plt.imshow(phase_spectrum, cmap = 'gray')
plt.title('Phase Spectrum'), plt.xticks([]), plt.yticks([])
plt.show()

#H_lpf(wx, wy)
H_lpf = []
H_hpf = []
origin_row = (fshift.shape[0]-1) // 2
origin_col = (fshift.shape[1]-1) // 2
w_row = np.linspace(-np.pi, +np.pi, fshift.shape[0])
w_col = np.linspace(-np.pi, +np.pi, fshift.shape[1])

#make Ideal LPF and HPF
for i in range(0, fshift.shape[0]):
    temp_lpf = []
    temp_hpf = []
    for j in range(0, fshift.shape[1]):
        if np.sqrt(np.square(w_row[i] - w_row[origin_row]) +
np.square(w_col[j] - w_col[origin_col])) <= (np.pi/4) :
            temp_lpf.append(1)
            temp_hpf.append(0)
        else:
            temp_lpf.append(0)
            temp_hpf.append(1)
    H_lpf.append(temp_lpf)
    H_hpf.append(temp_hpf)

#Plot Ideal LPF and HPF
plt.subplot(121),plt.imshow(H_lpf, cmap = 'gray')
plt.title('Ideal low pass filter'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(H_hpf, cmap = 'gray')
plt.title('Ideal high pass filter'), plt.xticks([]), plt.yticks([])
plt.show()

#Filtering by Ideal LPF and HPF
Y_lpf = fshift * H_lpf
Y_hpf = fshift * H_hpf
y_lpf = np.fft.ifft2(np.fft.ifftshift(Y_lpf)).real
y_hpf = np.fft.ifft2(np.fft.ifftshift(Y_hpf)).real
y_hpf = np.abs(y_hpf) / np.max(y_hpf)

plt.subplot(121),plt.imshow(y_lpf, cmap = 'gray')
plt.title('Low pass filtered img'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(y_hpf, cmap = 'gray')
plt.title('High pass filter img'), plt.xticks([]), plt.yticks([])
plt.show()

#3x3 laplacian filter
laplacian = np.array([[0, -1, 0], [-1, 4, -1], [0, -1, 0]])

#Gausian filter with std.dev = 0.8, kernel size =3
sigma = 0.8
kernel_size = 3
gaussian = []
for i in range(0, kernel_size):
    temp = []
    for j in range(0, kernel_size):
        x = np.abs(i - (kernel_size // 2))
```

```
        y = np.abs(j - (kernel_size // 2))
        temp.append(1 / 2 / np.pi / sigma / sigma * np.exp(-(np.square(x) +
np.square(y)) / 2 / sigma / sigma))
    gaussian.append(temp)

#filtering by laplacian and gaussian filter
Y_laplacian = np.fft.fftshift(np.fft.fft2(expand(laplacian, len(img), 3)))
Y_gaussian = np.fft.fftshift(np.fft.fft2(expand(gaussian, len(img), 3)))

#plot images filtered by laplacian and gaussain filter
plt.subplot(121), plt.imshow(20*np.log(1+np.abs(Y_laplacian)), cmap='gray')
plt.title('Laplacian magnitude spectrum'), plt.xticks([]), plt.yticks([])
plt.subplot(122), plt.imshow(20*np.log(1+np.abs(Y_gaussian)), cmap='gray')
plt.title('Gaussian magnitude spectrum'), plt.xticks([]), plt.yticks([])
plt.show()

#filtering noised image by gaussian filter
noised_img = cv2.imread('noised_img.jpg',0)
noised_fshift = np.fft.fftshift(np.fft.fft2(noised_img))
Y_lpf_noised = noised_fshift * H_lpf
y_lpf_noised = np.fft.ifft2(np.fft.ifftshift(Y_lpf_noised)).real
y_gaussian_noised = conv2(noised_img, gaussian, 3)

#print PSNR score
psnr(img, y_lpf_noised)
psnr(img, y_gaussian_noised)

#compare Ideal LPF and Gaussian filter
plt.subplot(121), plt.imshow(y_lpf_noised, cmap='gray')
plt.title('Ideal LPF'), plt.xticks([]), plt.yticks([])
plt.subplot(122), plt.imshow(y_gaussian_noised, cmap='gray')
plt.title('Gaussian filter'), plt.xticks([]), plt.yticks([])
plt.show()

#filtering by laplacian filter
Y_laplacian = np.fft.fftshift(np.fft.fft2(expand(laplacian, len(img), 3)))
* fshift
y_laplacian = np.fft.ifft2(np.fft.ifftshift(Y_laplacian)).real
y_laplacian = np.abs(y_laplacian) / np.max(y_laplacian)

#compare Ideal HPF and Laplacian filter
plt.subplot(121),plt.imshow(y_hpf, cmap = 'gray')
plt.title('Ideal HPF'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(y_laplacian, cmap ='gray')
plt.title('Laplacian Filter'), plt.xticks([]), plt.yticks([])
plt.show()
```