Image Processing Course Project: Image Filtering with Wiener Filter and Median Filter

Le-Anh Tran

Dept. of Electronics Engineering

Myongji University

leanhtran@mju.ac.kr

Introduction

This project report includes:

1. Gaussian Noise Generation
2. Wiener Filter Implementation
3. Median Filter Implementation

The source code is available at:

https://github.com/tranleanh/Wiener-Median-Comparison

DOI: 10.13140/RG.2.2.15700.65921

Yongin, Korea

April 2019

1. Gaussian Noise Generation

Gaussian noise is statistical noise having a probability density function (PDF) equal to that of the Gaussian distribution. In other words, the values that the noise can take on are Gaussian-distributed. Figure 1 shows some Gaussian noisy images, left to right: $\sigma_1 = 10, \sigma_2 = 20, \sigma_3 = 30$; while figure 2 depicts the implementation to confirm $\sigma_1, \sigma_2, \sigma_3$ by calculating rms values.

a) Generate noisy image

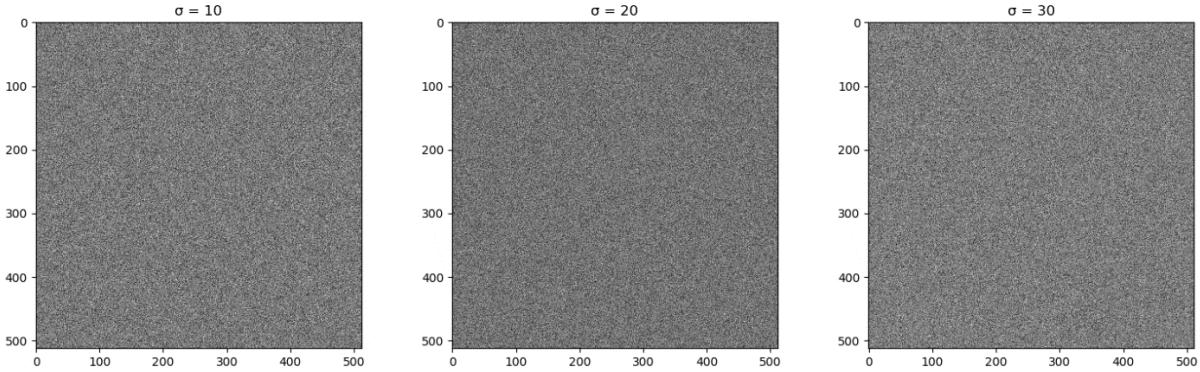

Figure 1. Noisy images generation. Left to right: $\sigma_1 = 10, \sigma_2 = 20, \sigma_3 = 30$.

b) Confirm $\sigma_1, \sigma_2, \sigma_3$

```
# Load image
file_name = os.path.join('black.jpg')
img = rgb2gray(plt.imread(file_name))
```

```
# Add Gaussian noise
noisy_img_1 = add_gaussian_noise(img, sigma = 10)
noisy_img_2 = add_gaussian_noise(img, sigma = 20)
noisy_img_3 = add_gaussian_noise(img, sigma = 30)
```

```
def rms(img):
    return np.sqrt(np.mean(img**2))
```

```
# Calculate rms
sigma_1 = rms(noisy_img_1)
sigma_2 = rms(noisy_img_2)
sigma_3 = rms(noisy_img_3)

print('\u03c3_1 = ' + str(sigma_1))
print('\u03c3_2 = ' + str(sigma_2))
print('\u03c3_3 = ' + str(sigma_3))
```

```
σ_1 = 9.968774685451546
σ_2 = 19.990788583150575
σ_3 = 29.957471978350974
```

Figure 2. Confirm $\sigma_1, \sigma_2, \sigma_3$ by calculating rms values.

c) Add noise to image

Figure 3 shows a 256-level gray image, and figure 4 illustrates some images added Gaussian noise generated previously.

Figure 3. 256-level gray image.

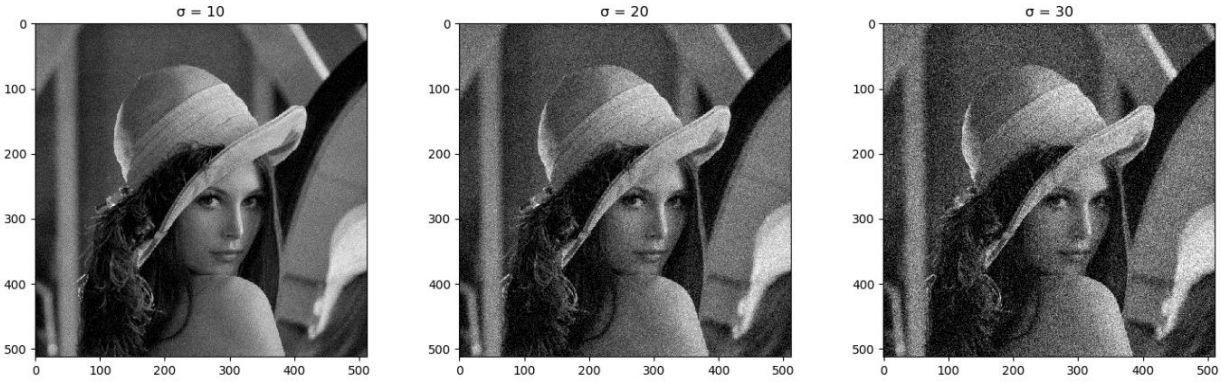

Figure 4. Images added Gaussian noise.

2. Wiener Filter Implementation

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

In which,

$\hat{F}(u,\ v)$ = the estimate

$G(u,\ v)$ = degraded image

$H(u,\ v)$ = degradation function

$H^*(u,\ v)$ = complex conjugate of $H(u,\ v)$

K = constant

Eq. (1) expresses Wiener Filter and it is applied in the frequency domain. Figure 5 shows an implementation of Wiener Filter using Python.

a, b) Construct Wiener Filter and apply it to noisy image

```
def gaussian_kernel(kernel_size = 3):
    h = gaussian(kernel_size, kernel_size / 3).reshape(kernel_size, 1)
    h = np.dot(h, h.transpose())
    h /= np.sum(h)
    return h

def wiener_filter(img, kernel, K):
    kernel /= np.sum(kernel)
    dummy = np.copy(img)
    dummy = fft2(dummy)
    kernel = fft2(kernel, s = img.shape)
    kernel = np.conj(kernel) / (np.abs(kernel) ** 2 + K)
    dummy = dummy * kernel
    dummy = np.abs(ifft2(dummy))
    return dummy
```

```
# Load image and convert it to gray scale
file_name = os.path.join('lena1000p.jpg')
img = rgb2gray(plt.imread(file_name))

# Blur the image
blurred_img = blur(img, kernel_size = 7)

# Add Gaussian noise
noisy_img_1 = add_gaussian_noise(blurred_img, sigma = 10)
noisy_img_2 = add_gaussian_noise(blurred_img, sigma = 20)
noisy_img_3 = add_gaussian_noise(blurred_img, sigma = 30)

# Apply Wiener Filter
kernel = gaussian_kernel(5)
filtered_img_1 = wiener_filter(noisy_img_1, kernel, K = 10)
filtered_img_2 = wiener_filter(noisy_img_2, kernel, K = 10)
filtered_img_3 = wiener_filter(noisy_img_3, kernel, K = 10)
```

Figure 5. Wiener Filter implementation using Python.

Results of applying Wiener Filter to restore noisy images are illustrated in figure 6, left-to-right then top-to-bottom, the first three images are blurred and added Gaussian noise with the sigma value of 10, 20, and 30, respectively, and the rest images are the results of applying Wiener Filter correspondingly to the first three degraded images, meanwhile, figure 7 depicts an implementation to calculate rms error of Wiener Filter.

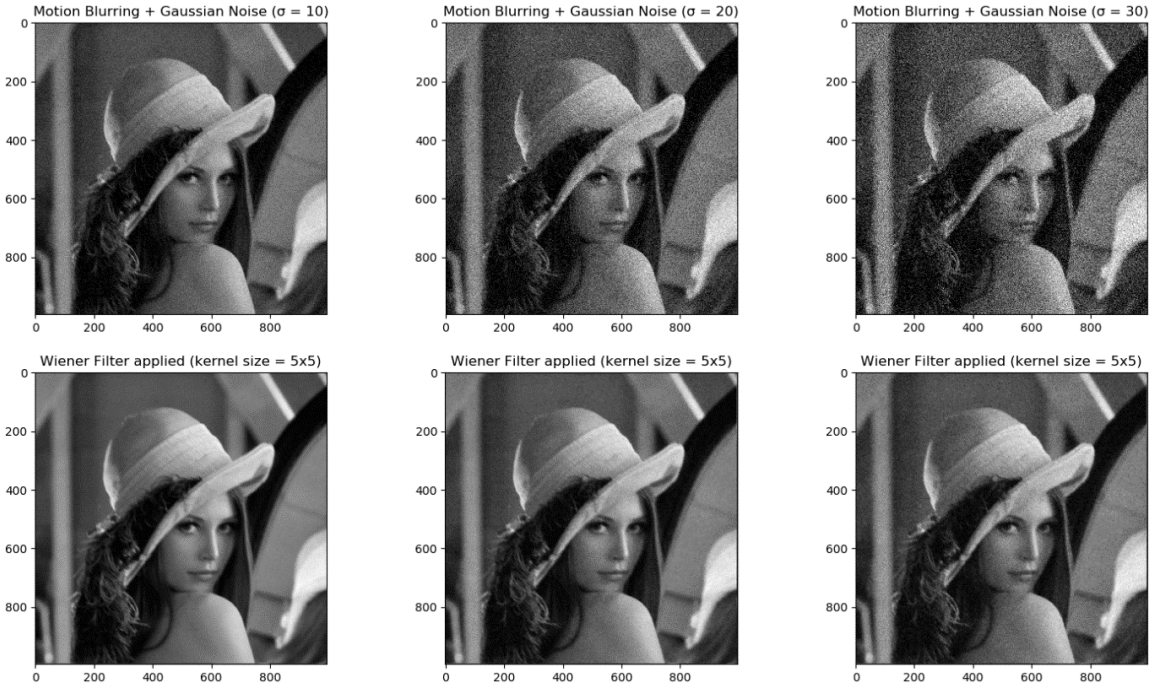

Figure 6. Results of applying Wiener Filter to restore noisy images.

c) Calculate rms error

```
# Load image and convert it to gray scale
file_name = os.path.join('lena1000p.jpg')
img = rgb2gray(plt.imread(file_name))

# Add Gaussian noise
noisy_img_1 = add_gaussian_noise(img, sigma = 10)
noisy_img_2 = add_gaussian_noise(img, sigma = 20)
noisy_img_3 = add_gaussian_noise(img, sigma = 30)

# Apply Wiener Filter
kernel = gaussian_kernel(5)
filtered_img_1 = wiener_filter(noisy_img_1, kernel, K = 10)
filtered_img_2 = wiener_filter(noisy_img_2, kernel, K = 10)
filtered_img_3 = wiener_filter(noisy_img_3, kernel, K = 10)
```

```
def wiener_rms(original, wiener):
    return np.sqrt(np.mean((wiener - original)**2))
```

```
# Calculate rms
rms_1 = wiener_rms(img, filtered_img_1)
rms_2 = wiener_rms(img, filtered_img_2)
rms_3 = wiener_rms(img, filtered_img_3)

print('rms_1 = ' + str(rms_1))
print('rms_2 = ' + str(rms_2))
print('rms_3 = ' + str(rms_3))
```

```
rms_1 = 90.0036927322022
rms_2 = 90.00267554306225
rms_3 = 90.00665602015498
```

Figure 7. Calculate rms error of Wiener Filter.

3. Median Filter Implementation

Median Filter computes the median of all the pixels under the kernel window and the central pixel is replaced with this median value. This is highly effective in removing salt-and-pepper noise.

a, b) Apply Median Filter to noisy image

```
def median_filter(data, kernel_size):
    temp = []
    indexer = kernel_size // 2
    data_final = []
    data_final = np.zeros((len(data),len(data[0])))
    for i in range(len(data)):

        for j in range(len(data[0])):

            for z in range(kernel_size):
                if i + z - indexer < 0 or i + z - indexer > len(data) - 1:
                    for c in range(kernel_size):
                        temp.append(0)
                else:
                    if j + z - indexer < 0 or j + indexer > len(data[0]) - 1:
                        temp.append(0)
                    else:
                        for k in range(kernel_size):
                            temp.append(data[i + z - indexer][j + k - indexer])

            temp.sort()
            data_final[i][j] = temp[len(temp) // 2]
            temp = []
    return data_final
```

```
# Load image and convert it to gray scale
file_name = os.path.join('lena1000p.jpg')
img = rgb2gray(plt.imread(file_name))

# Add Gaussian noise
noisy_img_1 = add_gaussian_noise(img, sigma = 10)
noisy_img_2 = add_gaussian_noise(img, sigma = 20)
noisy_img_3 = add_gaussian_noise(img, sigma = 30)

# Apply Median Filter
filtered_img_1 = median_filter(noisy_img_1, 3)
filtered_img_2 = median_filter(noisy_img_2, 3)
filtered_img_3 = median_filter(noisy_img_3, 3)
```

Figure 8. Median Filter implementation using Python.

Figure 8 shows a Median Filter implementation using Python; while figure 9 shows some results of denoising using Median Filter, left-to-right and top-to-bottom, the first three images are added Gaussian noise and the rest images are the results of applying Median Filter respectively to the first three images. An implementation of calculating rms error of Median Filter is described in figure 10.

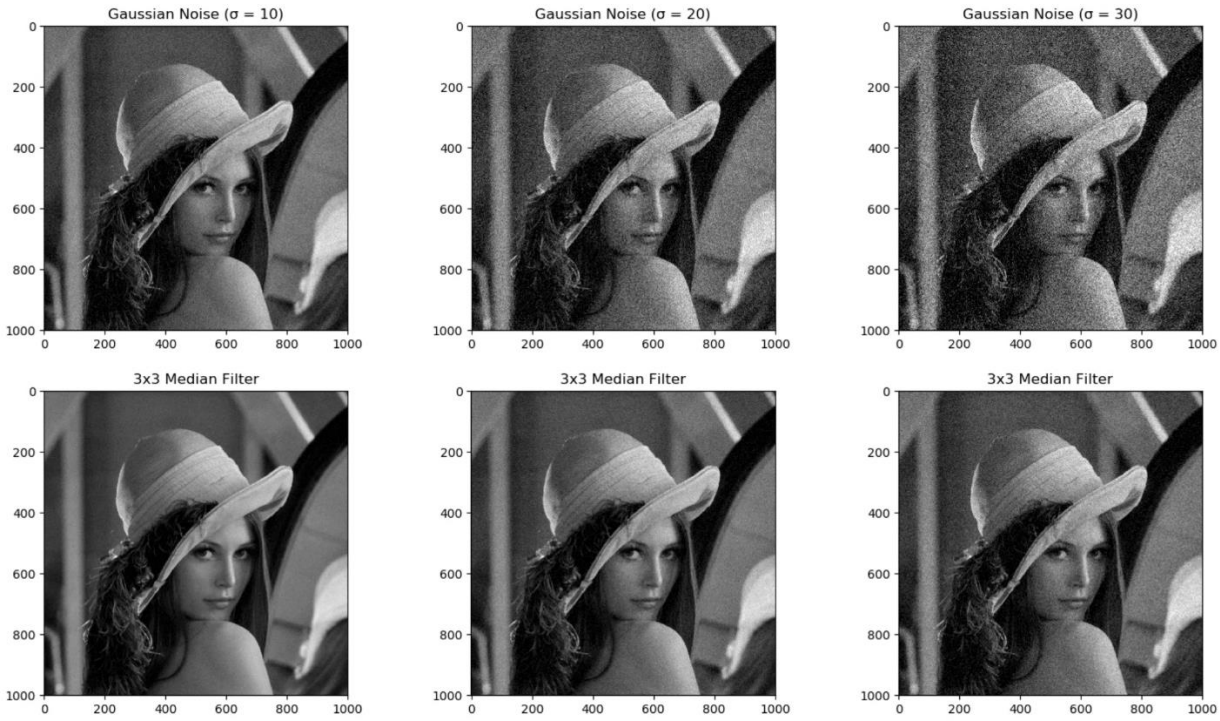

Figure 9. Results of denoising using Median Filter.

c) Calculate rms error

```
# Load image and convert it to gray scale
file_name = os.path.join('lena1000p.jpg')
img = rgb2gray(plt.imread(file_name))

# Add Gaussian noise
noisy_img_1 = add_gaussian_noise(img, sigma = 10)
noisy_img_2 = add_gaussian_noise(img, sigma = 20)
noisy_img_3 = add_gaussian_noise(img, sigma = 30)

# Apply Median Filter
filtered_img_1 = median_filter(noisy_img_1, 3)
filtered_img_2 = median_filter(noisy_img_2, 3)
filtered_img_3 = median_filter(noisy_img_3, 3)
```

```
def median_rms(original, median):
    return np.sqrt(np.mean((median - original)**2))
```

```
# Calculate rms
rms_1 = median_rms(img, filtered_img_1)
rms_2 = median_rms(img, filtered_img_2)
rms_3 = median_rms(img, filtered_img_3)

print('rms_1 = ' + str(rms_1))
print('rms_2 = ' + str(rms_2))
print('rms_3 = ' + str(rms_3))
```

```
rms_1 = 6.117000407600381
rms_2 = 9.397816173914253
rms_3 = 13.146347485618199
```

Figure 10. Calculate rms error of Median Filter.

d) Comparison of Wiener Filter and Median Filter

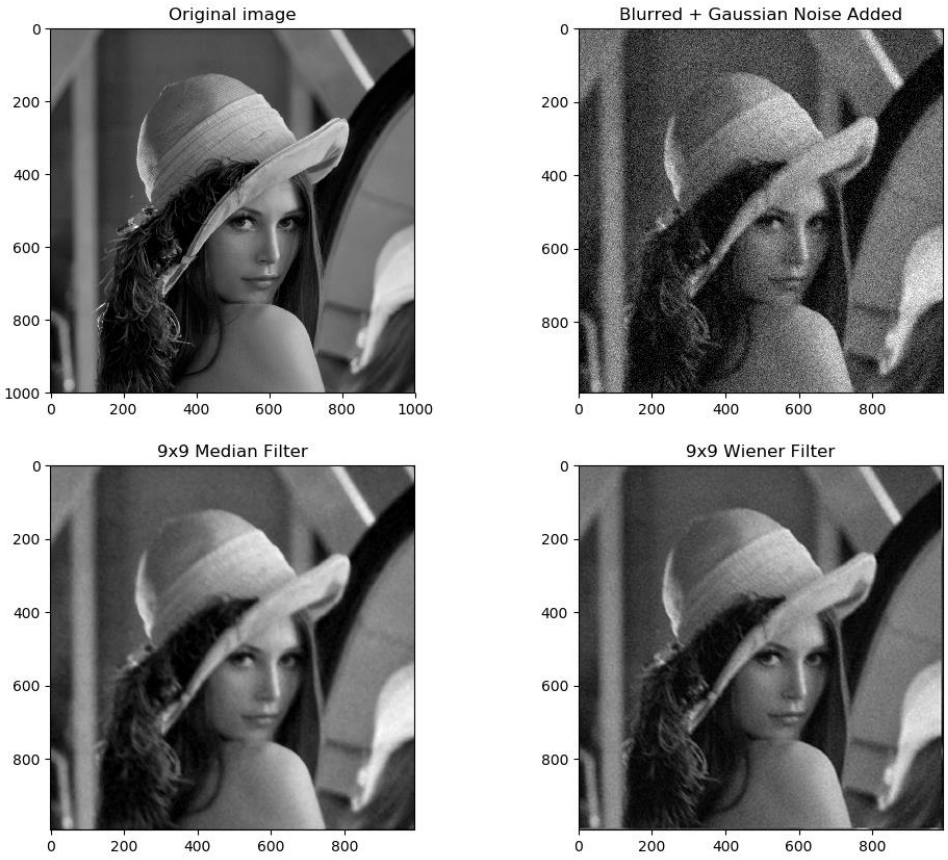

Figure 11. Comparison of Wiener Filter and Median Filter.

Figure 11 demonstrates the effectiveness of Median Filter and Wiener Filter in restoring degraded images. Previously, the 3x3 and 5x5 kernel sizes were used to denoise the images, but they might yield the same effect. Thus, the kernel size was increased up to 9x9 and the result showed the difference more clearly. It could be said that Wiener Filter outperforms Median Filter, Median Filter can reduce noise well but the output image would be less clear than that of Wiener Filter, meanwhile, Wiener Filter produced a favorable result for simultaneously deblurring and denoising.