Image Processing Project2 Answer Sheet

Name: 廖冠勳 Student ID: 0410893 Department: 電機 4A

1.

Explain how you implement FFT by DFT

利用以下 FFT 的公式：

$$F(u) = F_{even}(u) + F_{odd}(u)W_{2K}^{u} \quad (W_{2K}^{u} = e_{2K}^{-2\pi ju})$$
$$F(u+K) = F_{even}(u) - F_{odd}(u)W_{2K}^{u} \quad (W_{2K}^{u} = e_{2K}^{-2\pi ju})$$

先將$F_{even}(u)$、$F_{odd}(u)$分離，如下的程式碼片段 ：

```
// divide
CArray even = x[std::slice(0, N/2, 2)];
CArray  odd = x[std::slice(1, N/2, 2)];

// conquer
fft(even);
fft(odd);
```

使用 divide、conquer 的遞迴方式將整個數列切出偶數與奇數的兩個成對的 array，並在最後以前半段使用加法，後半段使用減法方式實作，如下程式碼片段 ：

```
// combine
for (size_t k = 0; k < N/2; ++k)
{
    Complex t = std::polar(1.0, -2 * PI * k / N) * odd[k];
    x[k    ] = even[k] + t;
    x[k+N/2] = even[k] - t;
}
```

藉由上述步驟實現該式。

Explain how you implement inverse FFT from FFT

使用 inverse FT 與 FT 互相轉換的性質：

已知：

$$X(j\omega) = \int_{-\infty}^{\infty} x(t)e^{-j\omega t}dt$$

我們在頻域中 apply 一個負號：

$$X(-j\omega) = \int_{-\infty}^{\infty} x(t)e^{j\omega t}dt$$

再做 conjugate :

$$X^{*}(-j\omega) = \int_{-\infty}^{\infty} x^{*}(t)e^{-j\omega t}dt = F\{x^{*}(t)\}$$

即可得其 inverse

$$X(t) = \frac{1}{2}\pi\int_{-\infty}^{\infty} X(\omega)e^{j\omega t}d\omega$$

使用此概念，實作如下程式碼 ：

```
void ifft(CArray& x)
{
    // conjugate the complex numbers
    x = x.apply(std::conj);

    // forward fft
    fft( x );

    // conjugate the complex numbers again
    x = x.apply(std::conj);

    // scale the numbers
    x /= x.size();
}
```

Show the result of problem1.txt after FFT （檔名 ：ans.txt）

因為電腦運算而在 inverse FT 產生的誤差版本

fft	ifft
76.6-4.6i	9+8i
0.22388+17.7693i	7-6i
25.9213-19.0919i	5+4i
7.23463+5.58257i	3-2i
9.6+3.6i	1-1.23269e-16i
-6.15611+15.8029i	8.88178e-16-2i
6.29289+16.9497i	3+8.88178e-16i
-28.583+25.8172i	0-4i
4.6+13.4i	8-2i
12.5241+9.35028i	6+3i
21.6787+19.0919i	9-0.6i
11.1702+24.2684i	7+1i
-16.4+11.6i	0.6+1.23269e-16i
-2.59191-4.52247i	7+7i
7.70711+7.05025i	5-5i
14.1781-14.0681i	6-6i

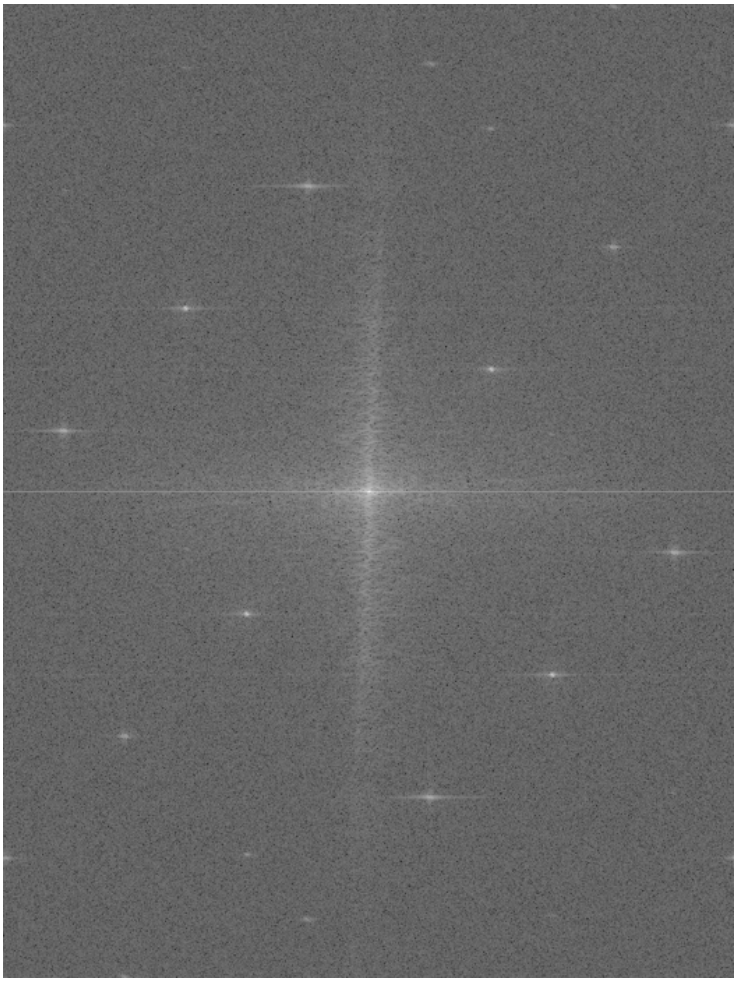
輸出時重新校正誤差的版本：

fft	ifft
76.6-4.6i	9+8i
0.22388+17.7693i	7-6i
25.9213-19.0919i	5+4i
7.23463+5.58257i	3-2i
9.6+3.6i	1
-6.15611+15.8029i	-2i
6.29289+16.9497i	3
-28.583+25.8172i	-4i
4.6+13.4i	8-2i
12.5241+9.35028i	6+3i
21.6787+19.0919i	9-0.6i
11.1702+24.2684i	7+1i
-16.4+11.6i	0.6
-2.59191-4.52247i	7+7i
7.70711+7.05025i	5-5i
14.1781-14.0681i	6-6i

2.

image with noise（檔名 ：periodic_noise_result.bmp）

image with noise after FFT（檔名 ：origin_frequency.bmp）



Explain what kind of the noise is (are) in the image

此 periodic noise 在頻域中因為對稱分佈在座標平面中央，應為一個由sin θ各種相位組合的合成波，只要濾除對稱點即可達到消除 noise 的目的！而由觀察可知，此圖亦包涵細微的 pepper 或是 salt noise，因此我在使用 frequency domain 的 filter（Gaussion filter）前，使用 adaptive mean filter 去除 aperiodic noise 後續再用 frequency domain 的 filter。而除去 aperiodic noise 後的圖如下所示（檔名：after_adaptive.bmp）

The filter(s) you use to process the image

(Hint: you need to show the filter parameter)

為了避免 butterworth 模仿 ideal low pass filter 所造成的 ringing 效應，我使用 Gaussion Low Pass filter。為了使 Gaussion 均勻分布，我使用 7*7 的 kernel。配上 try and error 調出來的標準差 2.3，程式碼片段如下：

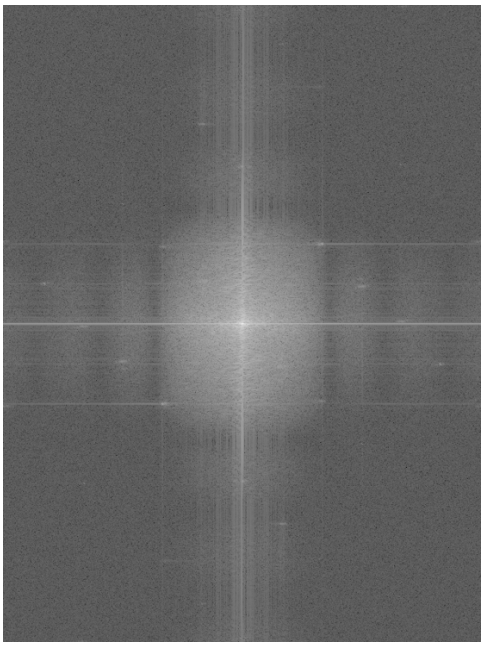
```
large_2d_blur_filter = gauss2D(shape=(7,7), sigma = 2.3)
large_blur_image = my_imfilter(test_image, large_2d_blur_filter);
```

```
def gauss2D(shape=(3,3),sigma=0.1):
    m,n = [(ss-1.)/2. for ss in shape]
    y,x = np.ogrid[-m:m+1,-n:n+1]
    h = np.exp( -(x*x + y*y) / (2.*sigma*sigma) )
    h[ h < np.finfo(h.dtype).eps*h.max() ] = 0
    sumh = h.sum()
    if sumh != 0:
        h /= sumh
    return h
```

The result of image with noise after _____Gaussion_____filter

（檔名：filter_result.bmp）

對應的頻域結果（檔名 ：after_frequency.bmp）



Summary and discussion:

1. 對於 aperiodic noise 而言：

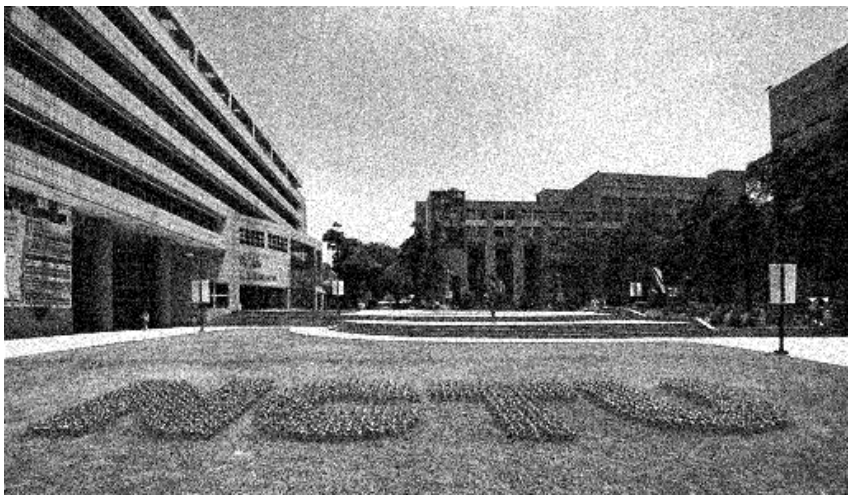
首先，由觀察可知圖形含有部分的 salt noise 與 pepper noise，在考慮因為其含有 periodic noise 而難以判斷此 noise 的 type 下，我使用較 robust 的 adaptive median noise 先將其濾除。

2. 對於 periodic noise 而言：

Gaussion Low Pass Filter 可有效的將 power 鎖在低頻（頻譜的中心點）並且濾除高頻（除了頻譜中心點周圍的其他點）成分，並且利用其對稱性能有效的濾除對稱於中心出現的 periodic noise 的成分。而且相較於 Butterworth 產生的 ringing 效應，進而留下部分高頻段的成分，對於濾除對稱出現在高頻段的 noise 不確定性較高，因此我選擇使用 Gaussion Noise 進行濾波。

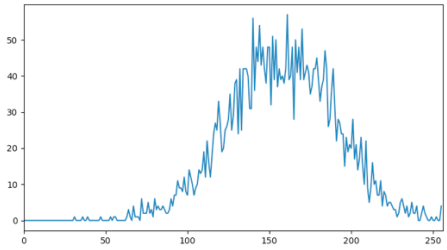
3.

problem3_4a.bmp



1.Noise type: uniform
2.Noise mean: 157
3.Noise deviation: 25.4626

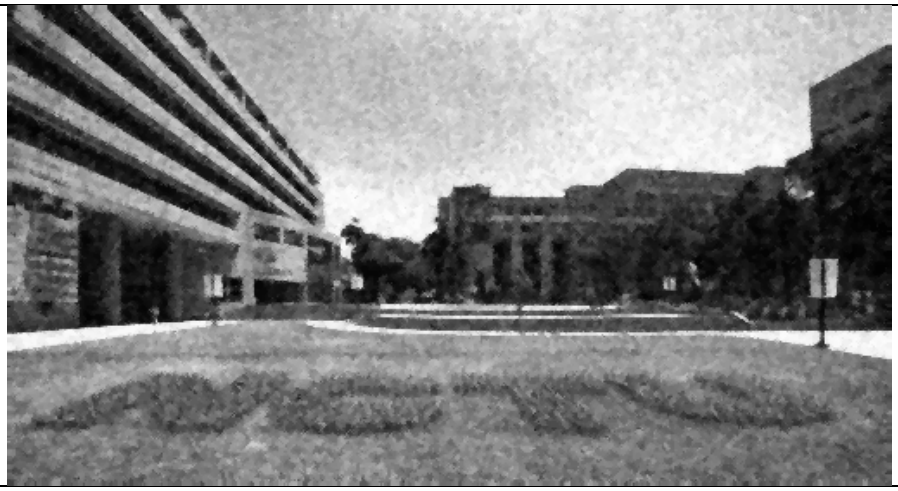
Histogram of noise:



How do you get the noise type?

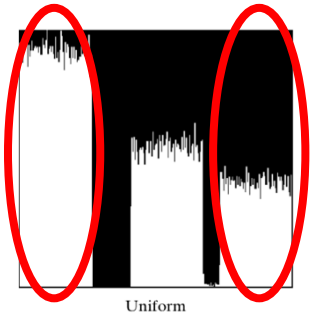
1. 在假設 noise 均勻分布的狀況下，使用一個遮罩，選取原圖中灰色較多的區塊，以避免 gray level 較低或是較高的部分影響判斷 noise type 的種類。
2. 觀察原圖並觀察其 histogram 可知，因為 histogram 在 gray level 100~200 之間均勻分布，所以此 noise 應為 uniform noise。

Result of problem3_4a.bmp after filtering（檔名 ：problem3_4a.bmp_result.bmp）



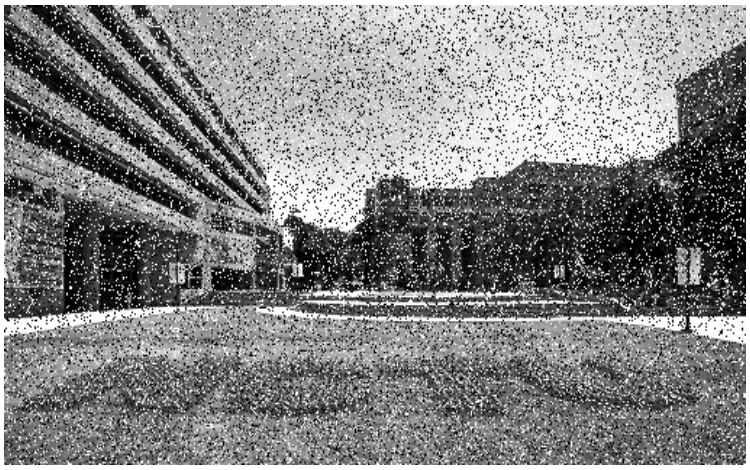
What filter is more suitable to problem3_1.bmp? why?

我使用了 adaptive median filter 作為 filter，因為 adaptive 適用於各式各樣的 aperiodic noise，但是在本圖中，uniform noise 均勻散布在各種 gray level 上。然而 adaptive median 只能將原圖或是中間值輸出，因此濾除後的雜訊仍有分佈在 median gray level 的 noise，如下圖所示。



對於這種區塊的 noise 較為束手無策，因為灰色部分必定包含原先圖形的部分，濾除的方式仍須待後續更好的方法探討。

problem3_4b.bmp

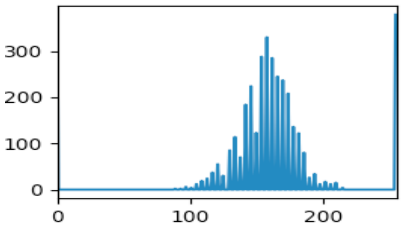


1.Noise type: Rayleigh + salt and pepper noise

2.Noise mean: 151

3.Noise deviation: 25.955

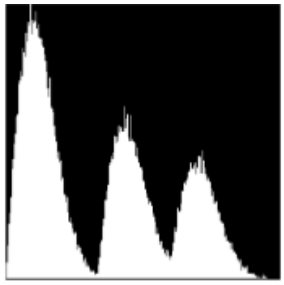
Histogram of noise:



How do you get the noise type?

1. 在假設 noise 均勻分布的狀況下，使用一個遮罩，選取原圖中灰色較多的區塊，以避免 gray level 較低或是較高的部分影響判斷 noise type 的種類。

2. 觀察原圖並觀察其 histogram 可知，因為 histogram 在 gray level 100~200 之間分佈近似 Rayleigh（如下圖所示），而且在 gray level 極大與極小處有分佈，所以此 noise 應為 Rayleigh + salt and pepper noise。



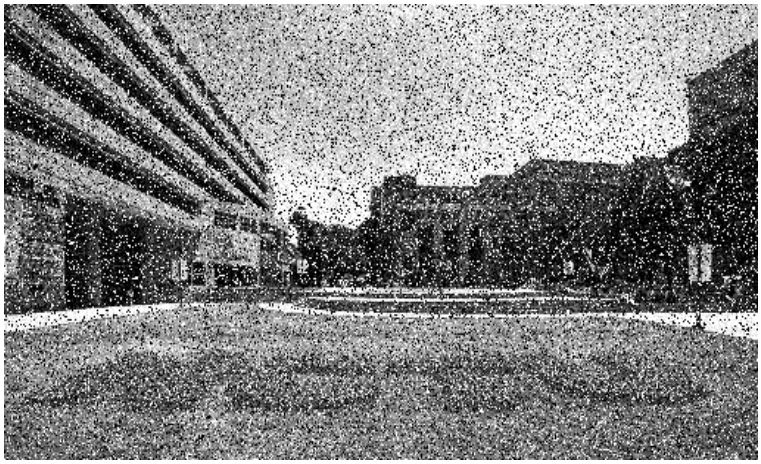
Rayleigh

Result of problem3_4b.bmp after filtering（檔名 ：problem3_4b.bmp_result.bmp）

What filter is more suitable to problem3_2.bmp? why?

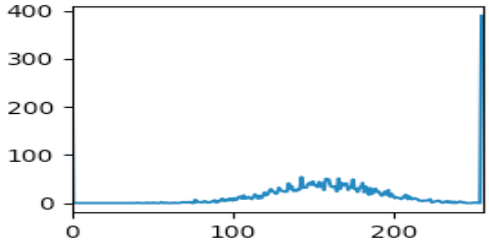
可以使用 adaptive 搭配 min filter，因為由 histogram 可得知 pepper noise 成分也很多，而且留下的雜訊蠻多 pepper noise，所以可以透過 max filter 除去部分 pepper noise，再用 adaptive 即可去除混合式的 noise。

problem3_4c.bmp



1.Noise type: Gaussion + salt and pepper
2.Noise mean: 152
3.Noise deviation: 26.26

Histogram of noise:

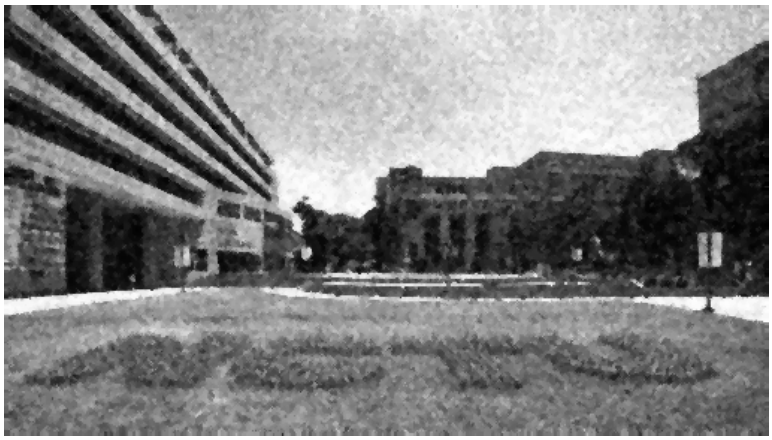


How do you get the noise type?

1. 在假設 noise 均勻分布的狀況下，使用一個遮罩，選取原圖中灰色較多的區塊，以避免 gray level 較低或是較高的部分影響判斷 noise type 的種類。
2. 觀察原圖並觀察其 histogram 可知，因為 histogram 在 gray level 以 Gaussion 方式分布，只是標準差較大，所且在 gray level 極值也有分佈，此 noise 應為 Gaussion 加上 salt and pepper noise。

Result of problem3_4c.bmp after filtering

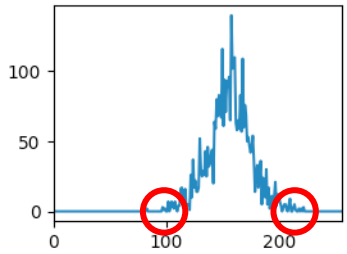
第一輪 adaptive-（檔名 ：problem3_4c.bmp_result.bmp）



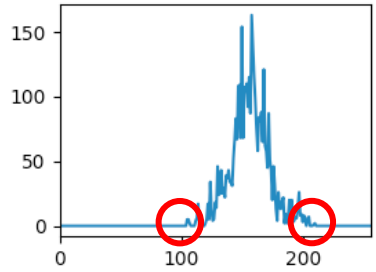
第二輪 adaptive-（檔名 ：problem3_4c.bmp_result.bmp_result.bmp）

What filter is more suitable to problem3_3.bmp? why?

Salt and pepper 對 adaptive median filter 來說較好解決，但比較麻煩的是 Gaussion noise 的部分。如下圖所示為第一輪的 histogram。

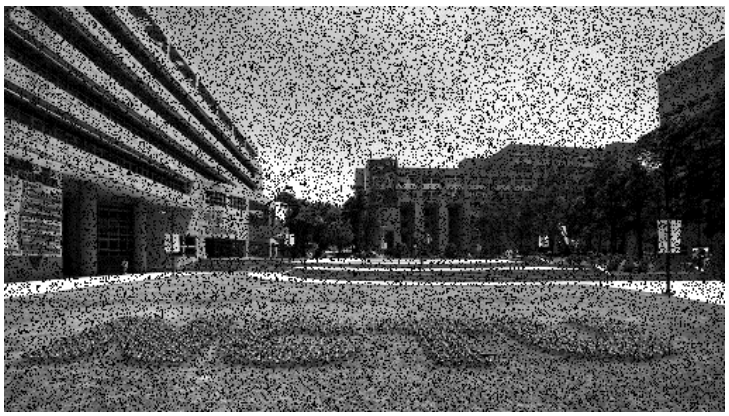


因此我做了第二輪的 adaptive 企圖降低 Gaussion noise 的效應，第二輪 adaptive 的 histogram 如下圖所示。



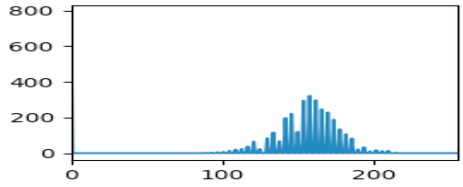
前後差異其實不太大，僅在偏離均值區域因 median filter 的效應，所以有降低 noise 的效應（如紅圈所示），應使用其他如 max 或 min filter 會較為妥當。

problem3_4d.bmp



1.Noise type: Gaussion+ pepper noise
2.Noise mean: 124
3.Noise deviation: 22.4195

Histogram of noise:



How do you get the noise type?

1. 在假設 noise 均勻分布的狀況下，使用一個遮罩，選取原圖中灰色較多的區塊，以避免 gray level 較低或是較高的部分影響判斷 noise type 的種類。
2. 觀察原圖可知此圖大部分的 noise 應為 pepper 並觀察其 histogram 可知。此外因為 histogram 在 gray level 以 Gaussion 方式分布，在 gray level 極小值也有分佈，而且 mean 比前幾張的 mean 都來得小，此 noise 應為 Gaussion 加上 pepper noise。

Result of problem3_4d.bmp after filtering

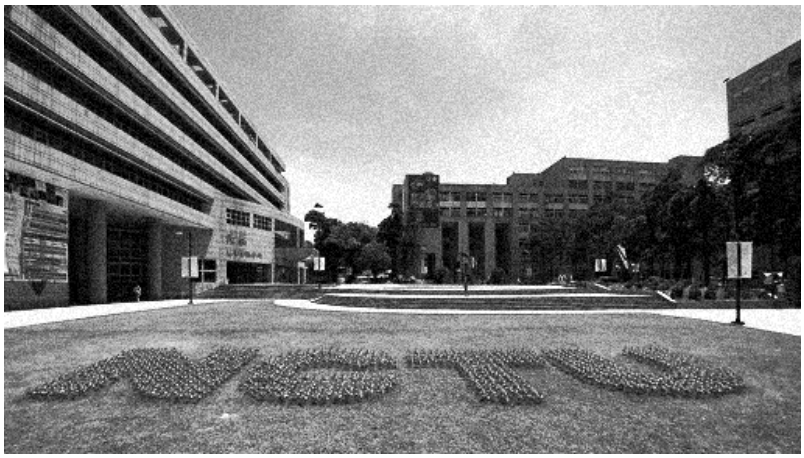
第一輪 adaptive-（檔名 ：problem3_4d.bmp_result.bmp）

第二輪 adaptive-（檔名 ：problem3_4d.bmp_result.bmp_result.bmp）

What filter is more suitable to problem3_4.bmp? why?

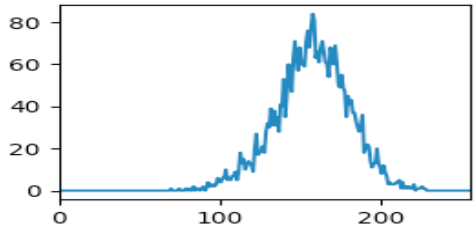
使用 max filter 可較有效率消除此圖的 pepper noise，但是考慮到階梯、走道、天空等區域大部分為白色區段，若用 max filter 會局部放大此區段，因此我使用兩次 adaptive filter 藉以消除圖形中的 pepper noise。

problem3_4e.bmp



1.Noise type: Gaussion
2.Noise mean: 157
3.Noise deviation: 25.25098

Histogram of noise:



How do you get the noise type?

1. 在假設 noise 均勻分布的狀況下，使用一個遮罩，選取原圖中灰色較多的區塊，以避免 gray level 較低或是較高的部分影響判斷 noise type 的種類。
2. 觀察原圖並觀察其 histogram 可知，因為 histogram 在 gray level 以 Gaussion 方式分布，此 noise 應為 Gaussion noise。

Result of problem3_4e.bmp after filtering（檔名 ：problem3_4e.bmp_result.bmp）

What filter is more suitable to problem3_4e.bmp? why?

使用 adaptive filter 對 Gaussion noise 來說，只能局部柔化其被 noise 污染的區域， 其效果如 problem3_4c.bmp 中所討論，僅在偏離均值區域因 median filter 的效應，所以有降低分佈於極值 noise 的效果，median filter 的效果只能局部縮小偏離中央的雜訊效應。