Digital Image Processing (2022 Fall) HW#2.2

r11942137 張舜程

1. Problem 3

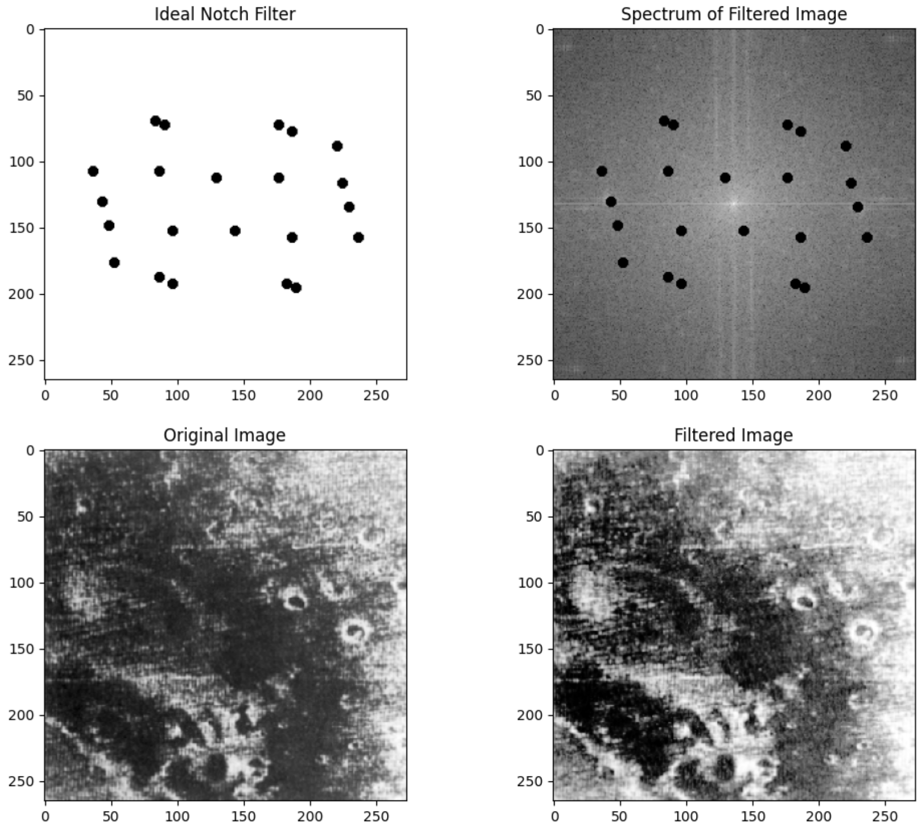
Implement the interference removal technique described in Example 5.7 to understand how a notch filter works. Your program should output the interference pattern as well as the restored image.

a. Notch reject filter

一開始，我先使用 Notch reject filter，將原圖在頻域中星星狀的雜訊濾除，做法比較原始一點，是打算盡可能地把可見的雜訊點慢慢找出來。

```
def notch_reject_filter(shape, d0=9, u_k=0, v_k=0):
    P, Q = shape
    H = np.zeros((P, Q))

    for u in range(0, P):
        for v in range(0, Q):
            D_uv = np.sqrt((u - P / 2 + u_k) ** 2 + (v - Q / 2 + v_k) ** 2)
            D_muv = np.sqrt((u - P / 2 - u_k) ** 2 + (v - Q / 2 - v_k) ** 2)
            if D_uv <= d0 or D_muv <= d0:
                H[u, v] = 0.0
            else:
                H[u, v] = 1.0
    return H
```

```
H1 = notch_reject_filter(img.shape, 4, 20, -40)
    H2 = notch_reject_filter(img.shape, 4, 25, 50)
    H3 = notch_reject_filter(img.shape, 4, 25, 100)
    H4 = notch_reject_filter(img.shape, 4, 2, 93)
    H5 = notch_reject_filter(img.shape, 4, 16, -88)
    H6 = notch_reject_filter(img.shape, 4, 44, -84)
    H7 = notch_reject_filter(img.shape, 4, 60, -40)
    H8 = notch_reject_filter(img.shape, 4, 55, -50)
    H9 = notch_reject_filter(img.shape, 4, 60, 46)
    H10 = notch_reject_filter(img.shape, 4, 63, 53)
    H11 = notch_reject_filter(img.shape, 4, 20, 7)

    H = H1 * H2 * H3 * H4 * H5 * H6 * H7 * H8 * H9 * H10 * H11
```



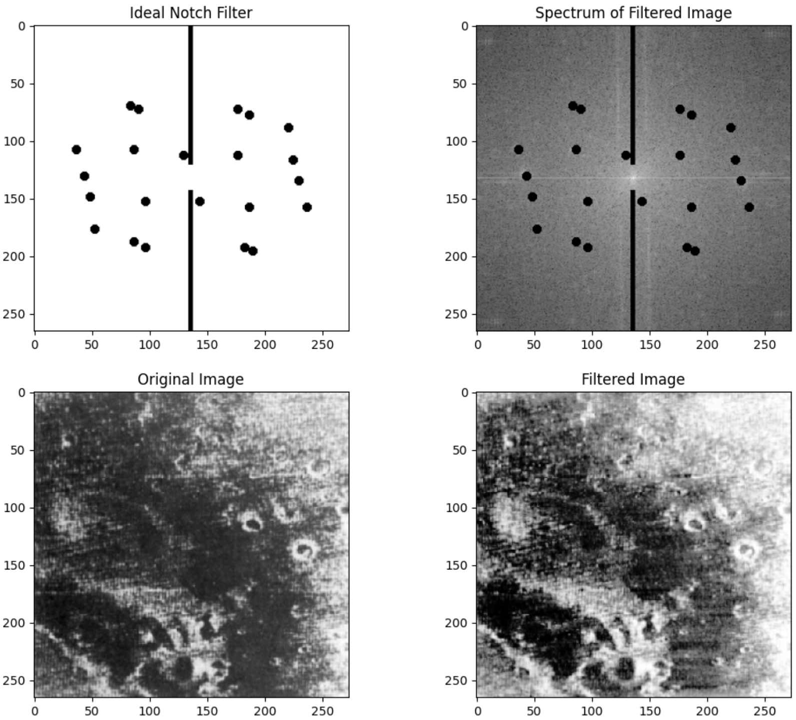
Ideal Notch Filter
Spectrum of Filtered Image
Original Image
Filtered Image
0
50
100
150
200
250

b. Notch Rectangle Filter

因為覺得把星星狀的雜訊點濾除之外，還是感覺圖片有明顯的橫線，因此我採用 Notch Rectangle Filter ，試著將頻域上的一條直線濾除，我認為有成功把原圖中較為明顯的橫線濾除掉了。

```python
class IdealRecNotch:
    def __init__(self, centers, size, height, width=5):
        self.centers = centers
        self.size = size
        self.height = height
        self.width = width
        self.v_filter = self.get_v_filter()
        self.h_filter = self.get_h_filter()

    def get_v_filter(self):
        M, N = self.size
        M_half = M // 2
        N_half = N // 2
        H = np.ones(self.size)
        centers_copy = self.centers.copy()
        for u, v in self.centers:
            centers_copy.append((-u, -v))

        for uk, vk in centers_copy:
            # shift origin to left top
            uk = uk + M_half
            vk = vk + N_half
            # set four point of the filter rectangle
            ustart = uk - self.height // 2 if uk - self.height // 2 > 0 else 0
            uend = uk + self.height // 2 if uk + self.height // 2 < M else M
            vstart = vk - self.width // 2 if vk - self.width // 2 > 0 else 0
            vend = vk + self.width // 2 if vk + self.width // 2 < N else N

            H[ustart: uend, vstart: vend] = 0
        return H
```

```python
H1 = notch_reject_filter(img.shape, 4, 20, -40)
    H2 = notch_reject_filter(img.shape, 4, 25, 50)
    H3 = notch_reject_filter(img.shape, 4, 25, 100)
    H4 = notch_reject_filter(img.shape, 4, 2, 93)
    H5 = notch_reject_filter(img.shape, 4, 16, -88)
    H6 = notch_reject_filter(img.shape, 4, 44, -84)
    H7 = notch_reject_filter(img.shape, 4, 60, -40)
    H8 = notch_reject_filter(img.shape, 4, 55, -50)
    H9 = notch_reject_filter(img.shape, 4, 60, 46)
    H10 = notch_reject_filter(img.shape, 4, 63, 53)
    H11 = notch_reject_filter(img.shape, 4, 20, 7)

    H = H1 * H2 * H3 * H4 * H5 * H6 * H7 * H8 * H9 * H10 * H11 * ideal_rec_notch.v_filter
```


Ideal Notch Filter
Spectrum of Filtered Image
Original Image
Filtered Image
0
50
100
150
200
250

2. Problem 2

The "photographer_degraded.tif" image is corrupted by motion blur and additive Gaussian noise. But we do not know the amount of motion blur and Gaussian noise. The "football player_degraded.tif" image is another degraded image. We do not have any information about the degradation. But it is reasonable to think that a Wiener filter may restore the image.

a. 2-1

Determine the best Wiener filter for each image. You should explicitly provide its mathematical expression and parameters.

- **Mathematical Expression**

$$\widehat{F}(u,v) = \left[\frac{1}{H(u,v)} \frac{|H(u,v)|^2}{|H(u,v)|^2 + \frac{1}{SNR}}\right] G(u,v)$$

where,

$$G(u,v) = F(u,v)H(u,v) + N(u,v)$$

therefore,

$$\widehat{F}(u,v) = \frac{|H(u,v)|^2}{|H(u,v)|^2 + \frac{1}{SNR}} F(u,v) + \frac{|H(u,v)|^*}{|H(u,v)|^2 + \frac{1}{SNR}} N(u,v)$$

Notice that if H is very small, the noise term can be reduced. i.e:

$$\widehat{F}(u,v) \sim F(u,v)$$

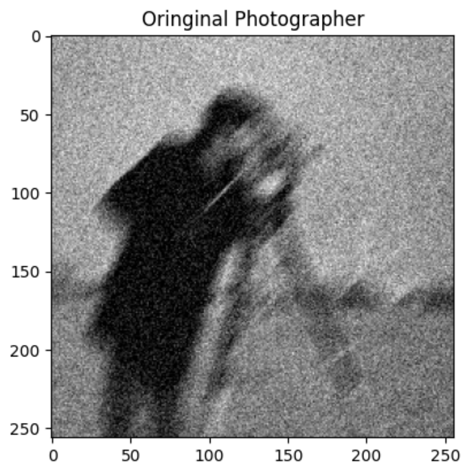
- **Parameters**
  - **Photographer**
    $H(u,v) = Gaussian\ \ filter\ with\ k = 3$
    $SNR = 0.5$
  - **Football players**
    $H(u,v) = Motion\ Blur\ filter\ with\ size = 15$
    $SNR = 2$

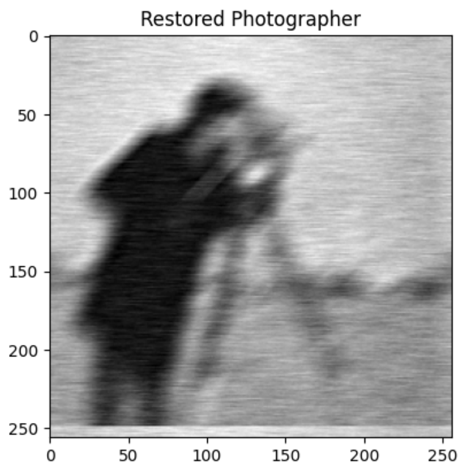
```
photo_img = np.array(Image.open('./images/Photographer_degraded.tif')).astype(np.float64)
foot_img = np.array(Image.open('./images/Football players_degraded.tif')).astype(np.float64

# Gaussian blur
kernel = gaussian_kernel(3)

# motion blur
size = 15
kernel_motion_blur = np.zeros((size, size))
kernel_motion_blur[int((size - 1) / 2), :] = np.ones(size)
kernel = kernel_motion_blur / size

photo_filtered_img = wiener_filter(photo_img, kernel, Noise_Signal_ratio=2)
foot_img_filtered_img = wiener_filter(foot_img, kernel, Noise_Signal_ratio=0.5)
```

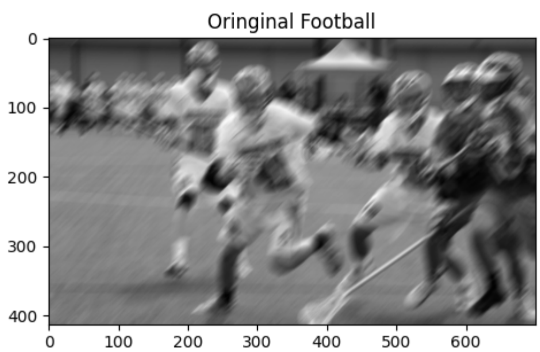
b. 2-2

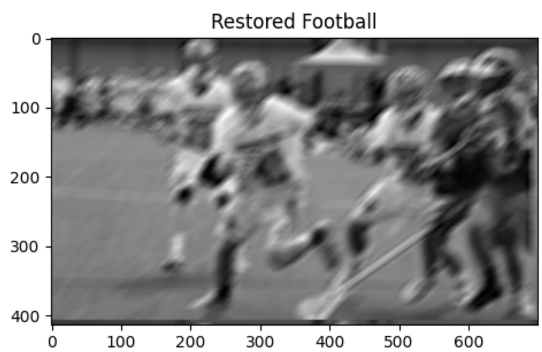
Show the restored image of each Wiener filter.

```
def wiener_filter(img, kernel, Noise_Signal_ratio):
    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 + Noise_Signal_ratio)
    dummy = dummy * kernel
    dummy = np.abs(ifft2(dummy))
    return dummy
```









我認為 Photographer 原圖中的噪點經過 Wiener filter 後有消除掉，經過很多的嘗試下，發現通道的 SNR 設為 0.5 時效果最為恰當；而 Football player 的原圖經過 Wiener filter 後，可以看到本來劇烈晃動造成的殘影有被消除了一些，但是結果還是有點模糊，也是在多次嘗試後，認為 SNR 設為 2 時的結果最佳。