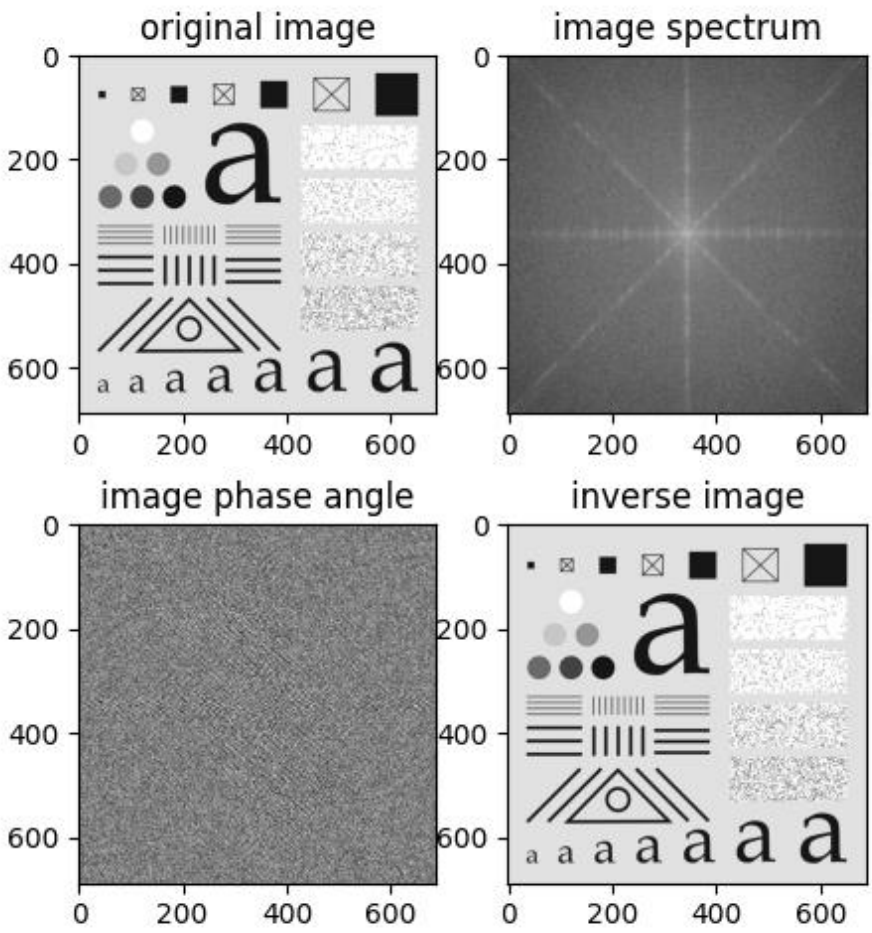
Homework4

R08525116 吳承哲

無實作 GUI

Part 1

```
dft = cv2.dft(np.float32(img), flags=cv2.DFT_COMPLEX_OUTPUT)
dft_shift = np.fft.fftshift(dft)

spectrum = 20*np.log(cv2.magnitude(dft_shift[:, :, 0], dft_shift[:, :, 1]))
phase = np.angle(dft_shift)[:, :, 0]

f_ishift = np.fft.ifftshift(dft_shift)
img_back = cv2.idft(f_ishift)
img_back = cv2.magnitude(img_back[:, :, 0], img_back[:, :, 1])
```



經過 Fourier 轉換後的影像，反轉換後看起來差異不大，但數值的大小不同。

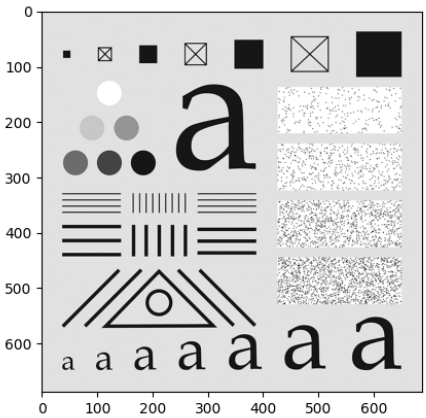
```python
def compute(img, factor):
    size = img.shape[0]*factor
    img = np.resize(img, (size, size))

    start = time.time()
    dft = cv2.dft(np.float32(img), flags=cv2.DFT_COMPLEX_OUTPUT)
    dft_shift = np.fft.fftshift(dft)
    f_ishift = np.fft.ifftshift(dft_shift)
    img_back = cv2.idft(f_ishift)
    img_back = cv2.magnitude(img_back[:, :, 0], img_back[:, :, 1])
    end = time.time()

    T = end - start

    return T
```

```
size X1:  0.017951250076293945
size X2:  0.07081055641174316
size X4:  0.30518412590026855
size X8:  1.2319860458374023
```

Size 每增加 2 倍，時間增加約 4 倍左右。

Part 2

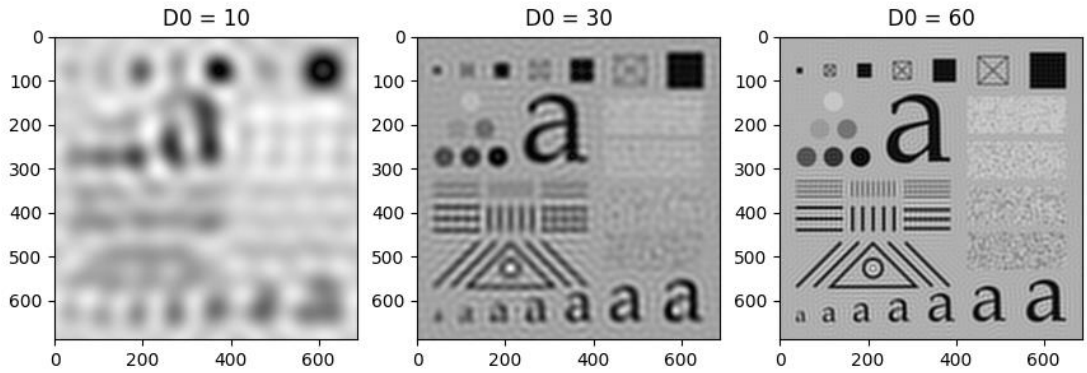
原圖



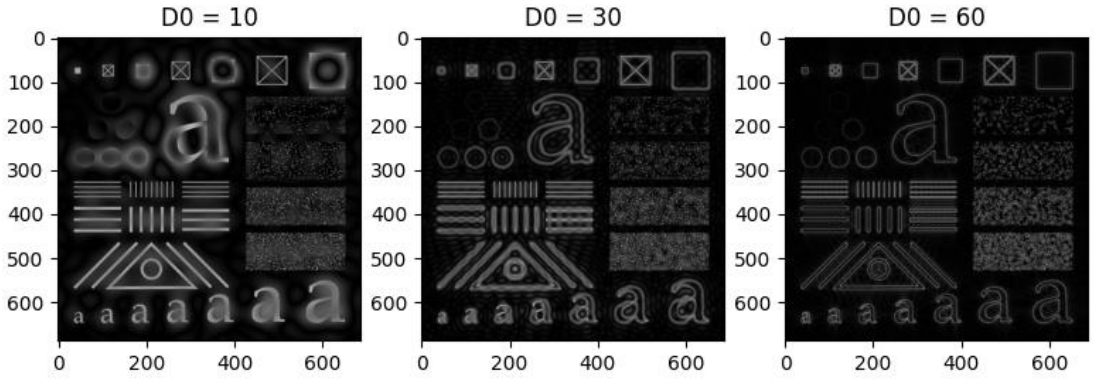
Ideal filter

```python
def Ideal_filter(img, D0, mode="L"):
    dft = cv2.dft(np.float32(img), flags=cv2.DFT_COMPLEX_OUTPUT)
    dft_shift = np.fft.fftshift(dft)

    P = dft.shape[0]
    Q = dft.shape[1]
    H = np.zeros((P, Q, 2))

    if mode == "L":  # low pass
        for u in range(P):
            for v in range(Q):
                distance = ((u - P/2)**2 + (v - Q/2)**2)**(0.5)
                if distance <= D0:
                    H[u][v] = 1
                else:
                    H[u][v] = 0
    elif mode == "H":  # high pass
        for u in range(P):
            for v in range(Q):
                distance = ((u - P/2)**2 + (v - Q/2)**2)**(0.5)
                if distance <= D0:
                    H[u][v] = 0
                else:
                    H[u][v] = 1
    else:
        print("wrong mode")

    fshift = dft_shift*H
    f_ishift = np.fft.ifftshift(fshift)
    img_back = cv2.idft(f_ishift)
    img_back = cv2.magnitude(img_back[:, :, 0], img_back[:, :, 1])

    return img_back
```

Low pass



High Pass

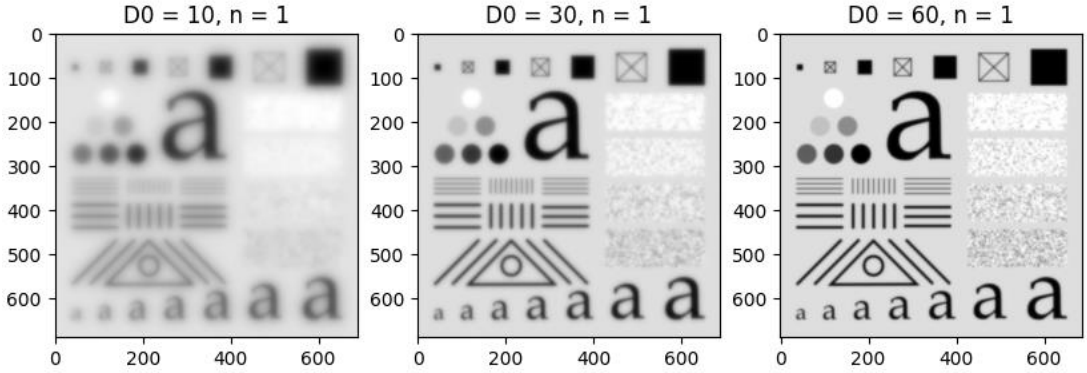


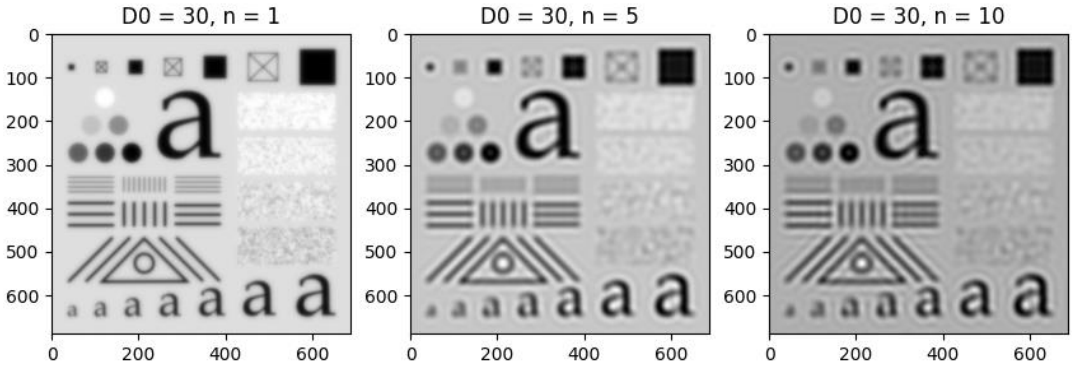
D0 越小，模糊效果越好。

Butterworth filter

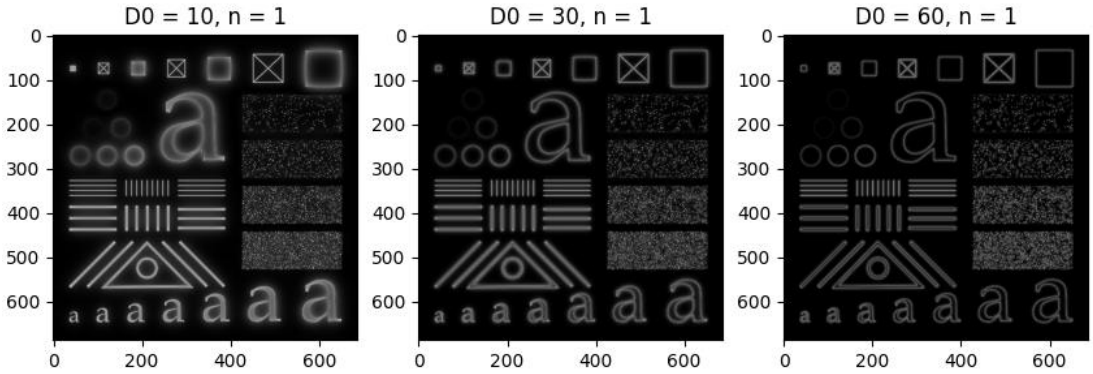
```
def Butterworth_filter(img, D0, n, mode="L"):
    dft = cv2.dft(np.float32(img), flags=cv2.DFT_COMPLEX_OUTPUT)
    dft_shift = np.fft.fftshift(dft)

    P = dft.shape[0]
    Q = dft.shape[1]
    H = np.zeros((P, Q, 2))

    if mode == "L":  # low pass
        for u in range(P):
            for v in range(Q):
                distance = ((u - P/2)**2 + (v - Q/2)**2)**(0.5)
                H[u][v] = 1 / (1 + (distance/D0)**(2*n))
    elif mode == "H":  # high pass
        for u in range(P):
            for v in range(Q):
                distance = ((u - P/2)**2 + (v - Q/2)**2)**(0.5)
                if distance != 0:
                    H[u][v] = 1 / (1 + (D0/distance)**(2*n))
    else:
        print("wrong mode")

    fshift = dft_shift*H
    f_ishift = np.fft.ifftshift(fshift)
    img_back = cv2.idft(f_ishift)
    img_back = cv2.magnitude(img_back[:, :, 0], img_back[:, :, 1])

    return img_back
```

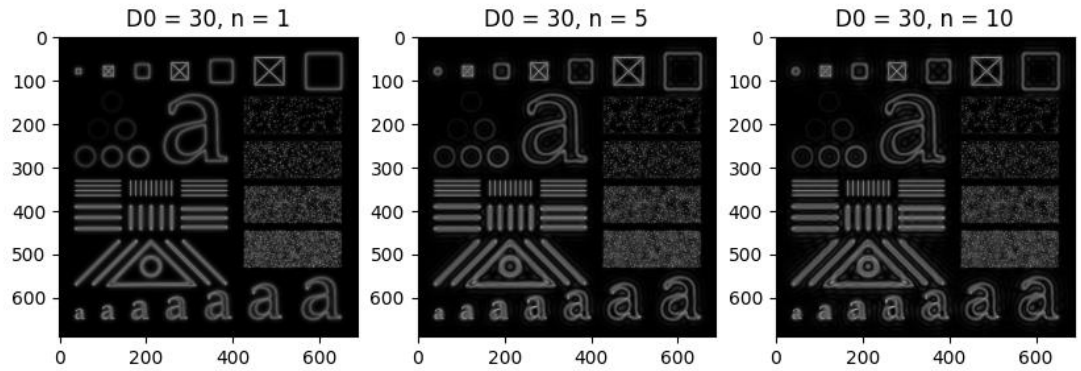
Low pass





High Pass





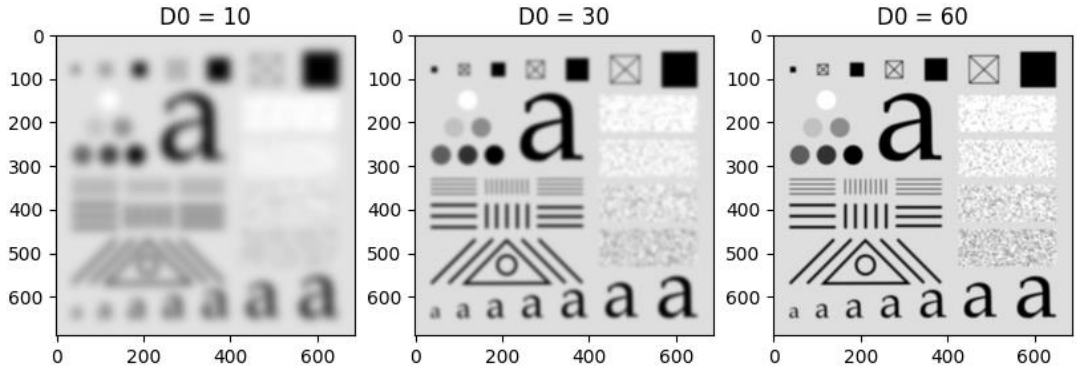
D0 越小，模糊效果越好。

N 越大，圖案有小小的變黑跟模糊的改變。

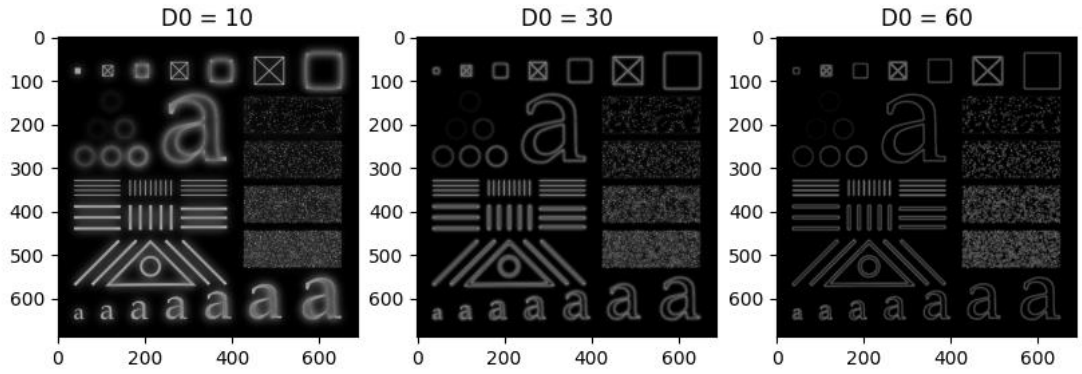
Gaussian filter

```python
def Gaussian_filter(img, D0, mode="L"):
    dft = cv2.dft(np.float32(img), flags=cv2.DFT_COMPLEX_OUTPUT)
    dft_shift = np.fft.fftshift(dft)

    P = dft.shape[0]
    Q = dft.shape[1]
    H = np.zeros((P, Q, 2))

    if mode == "L":  # low pass
        for u in range(P):
            for v in range(Q):
                distance = ((u - P/2)**2 + (v - Q/2)**2)**(0.5)
                H[u][v] = np.exp(-((distance**2)/(2*(D0**2))))
    elif mode == "H":  # high pass
        for u in range(P):
            for v in range(Q):
                distance = ((u - P/2)**2 + (v - Q/2)**2)**(0.5)
                H[u][v] = 1 - np.exp(-((distance**2)/(2*(D0**2))))
    else:
        print("wrong mode")

    fshift = dft_shift*H
    f_ishift = np.fft.ifftshift(fshift)
    img_back = cv2.idft(f_ishift)
    img_back = cv2.magnitude(img_back[:, :, 0], img_back[:, :, 1])

    return img_back
```

Low pass



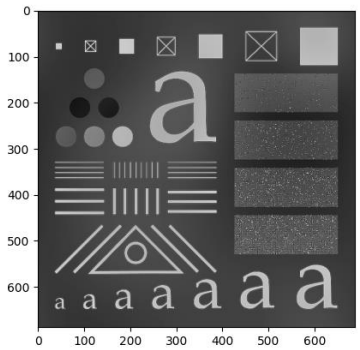
High pass

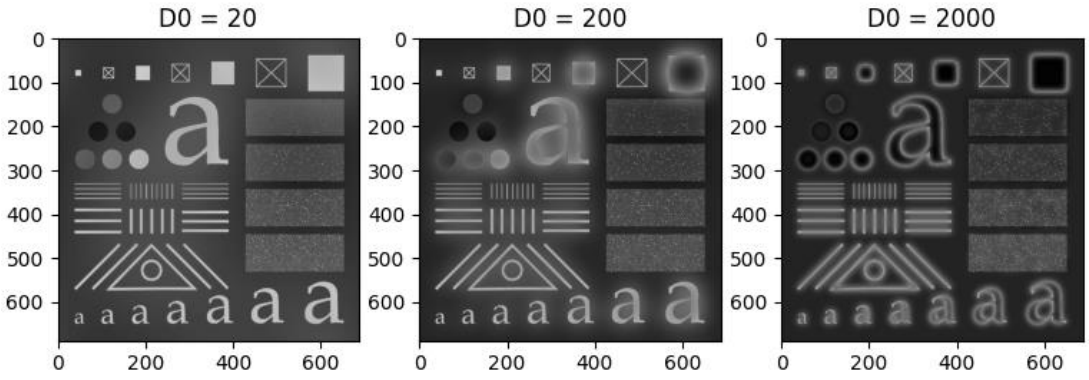


D0 越小，模糊效果越好。

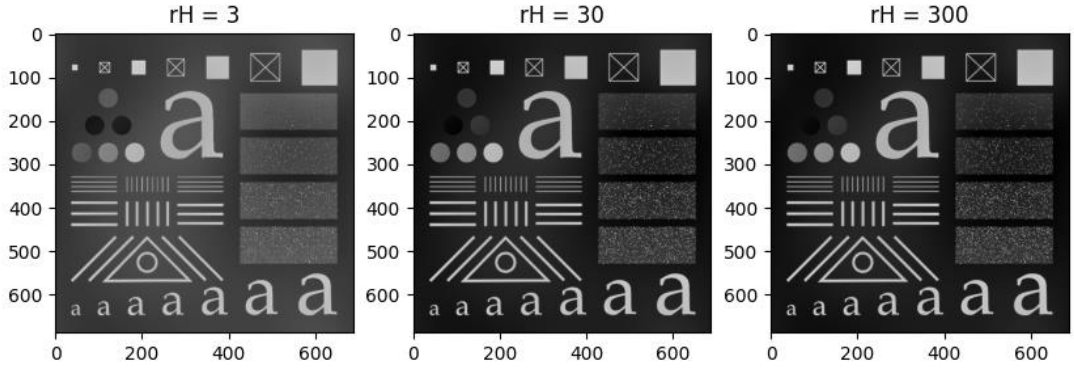
Part 3

D0 = 20, rH = 3, rL = 0.4, c = 5，以下調整數值以外的值接為此固定值。

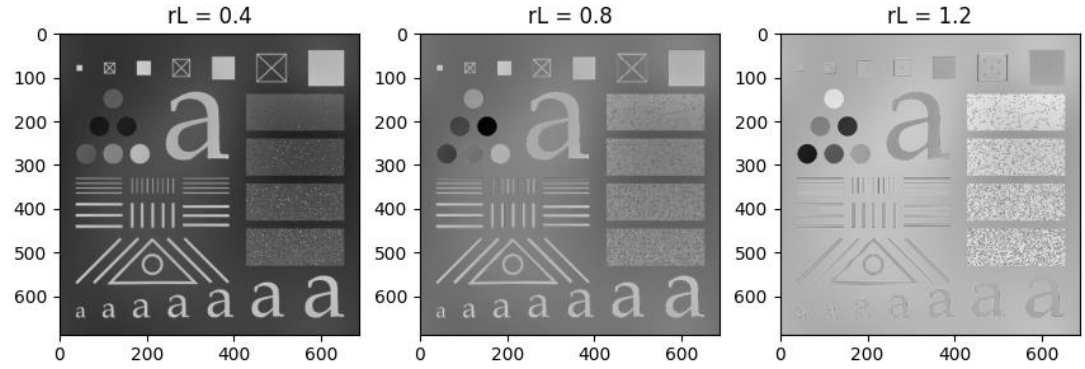




D0 越大，物體邊線越明顯，圖案越暗。



rH 越大，圖案越暗。



rL 越大，圖案越亮。

Part 4

Motion Blur

```
def motiom_blur(img, a, b, T):
    dft = cv2.dft(np.float32(img), flags=cv2.DFT_COMPLEX_OUTPUT)
    dft_shift = np.fft.fftshift(dft)

    M = dft.shape[0]
    N = dft.shape[1]
    H = np.zeros((M, N, 2))

    for u in range(M):
        for v in range(N):
            x = np.pi*(u*a + v*b)
            if x == 0:
                H[u][v] = 0
            else:
                number = (T*np.sin(x)*np.exp(-1j*x)) / x
                H[u][v][0] = np.real(number)
                H[u][v][1] = np.imag(number)

    fshift = dft_shift*H
    f_ishift = np.fft.ifftshift(fshift)
    img_back = cv2.idft(f_ishift)
    img_back = cv2.magnitude(img_back[:, :, 0], img_back[:, :, 1])

    return img_back
```

Inverse filter

```
def Inverse_filter(img, k):
    dft = cv2.dft(np.float32(img), flags=cv2.DFT_COMPLEX_OUTPUT)
    dft_shift = np.fft.fftshift(dft)

    M = dft.shape[0]
    N = dft.shape[1]
    H = np.zeros((M, N, 2))

    for u in range(M):
        for v in range(N):
            H[u][v] = np.exp((-k)*(((u + M/2)**2 + (v - N/2)**2)**(5/6)))

    fshift = dft_shift*H
    f_ishift = np.fft.ifftshift(fshift)
    img_back = cv2.idft(f_ishift)
    img_back = cv2.magnitude(img_back[:, :, 0], img_back[:, :, 1])

    return img_back
```

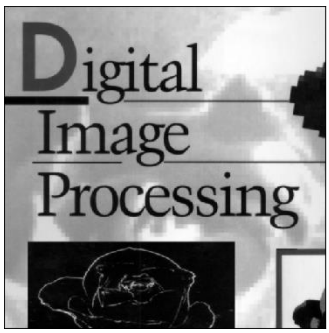
Wiener filter

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


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
```

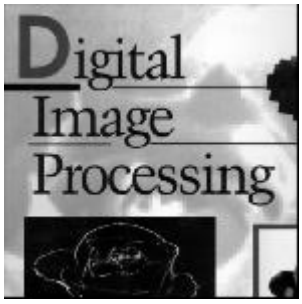
Add Gaussian noise with sigma = 20

```
def add_gaussian_noise(img, sigma):
    gauss = np.random.normal(0, sigma, np.shape(img))
    noisy_img = img + gauss
    noisy_img[noisy_img < 0] = 0
    noisy_img[noisy_img > 255] = 255
    return noisy_img
```

原圖 / motion blur



Inverse filter / wiener filter




Gaussian noise / Inverse filter / Wiener filter



在只有 motion blur 時，inverse filter 的還原表現比較好，不過在加入 Gaussian noise 之後，Wiener filter 的還原表現比較好。