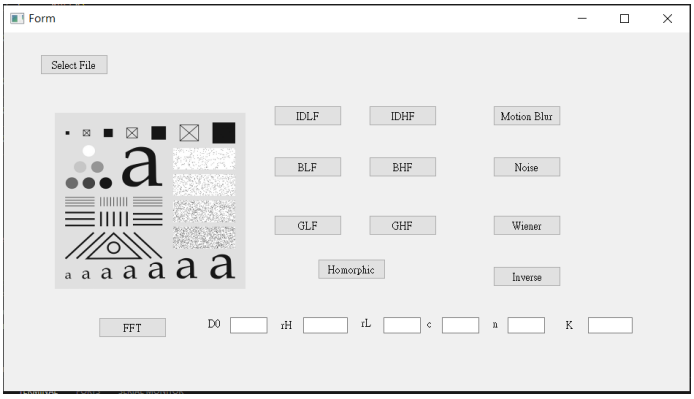
HW4

介面:



程式講解:

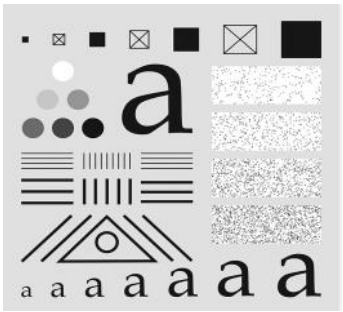
要匯入要處理的圖片，按下介面中的”Select File”即可。

1. 在匯入圖片後，按下”FFT”後會出現四張圖片，分別為圖片的頻譜圖、相位圖、反傅立葉轉換得到的影像以及其與最初影像的差異，並且有進行計時。

256X256 運行時間:

0.008975505828857422s

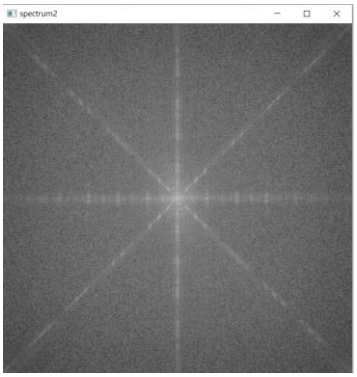
688X688 運行時間:



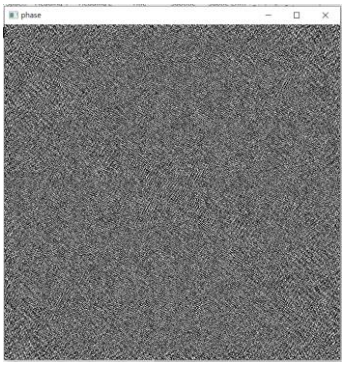
0.10575222969055176s

程式碼:

```python
def FT(self):
    f = np.fft.fft2((self.im)) # fourier transform
    self.f_shift = np.fft.fftshift(f)# shift origin point
    # show specturm photo
    Fmin=np.log(1+np.min(np.abs(self.f_shift)))
    Fmax=np.log(1+np.max(np.abs(self.f_shift)))
    Y=255*(np.log(1+np.abs(self.f_shift))-Fmin)/(Fmax-Fmin)

    #f_normalize=cv2.normalize(self.mag,None,0,1,cv2.NORM_MINMAX)
    Y=cv2.normalize(Y,None,0,1,cv2.NORM_MINMAX)


    phase_angle = np.angle(f) # show phase angle fig

    f_ishift=np.fft.ifftshift(f)
    inverse=np.fft.ifft2(f_ishift)
    inverse=np.abs(inverse)
    inverse = cv2.normalize(inverse,None,0,1,cv2.NORM_MINMAX)
    image_contrast = np.abs(inverse-self.im)


    cv2.imshow("spectrum2",Y)
    cv2.imshow("phase",phase_angle)

    cv2.imshow("Inverse image",inverse)
    cv2.imshow("Contrast after inverse fourier transform",image_contrast)
    return
```

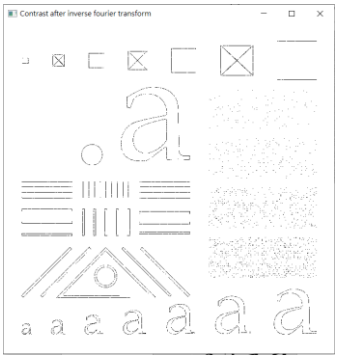


頻譜圖



相位角

反轉換後的圖片



與原圖片的差異

觀察與原圖片的差異時，會發現差異主要是在圖片的邊緣，推測可能是因為在反轉換的過程中失去了部分高頻的訊號所致。

(2)、(3):

在 D0 的空格填入數值以及在實施 Butterworth filter 時的 n 值、homomorphic

的參數後，按下介面中想要的濾波即可出現處理後的圖片。

程式碼:

```
def IdealLowpass(self):
    cv2.destroyAllWindows()
    self.Type="IDLP"
    self.processImage()
    return
def IdealHighpass(self):
    cv2.destroyAllWindows()
    self.Type="IDHP"
    self.processImage()
    return
def butterworthLow(self):
    cv2.destroyAllWindows()
    self.Type="BLF"
    self.processImage()
    return
def butterworthHigh(self):
    cv2.destroyAllWindows()
    self.Type="BHF"
    self.processImage()
    return
```

```
def gaussianLow(self):
    cv2.destroyAllWindows()
    self.Type="GLF"
    self.processImage()
    return
def gaussianHigh(self):
    cv2.destroyAllWindows()
    self.Type="GHF"
    self.processImage()
    return
```

隨著按下不同的鍵，程式會將 self.Type 改成不同的參數，再呼叫函數

self.processImage()

```
def processImage(self):
    D0 = int(self.cuttoff.text())
    image = np.fft.fft2((self.im))
    image = np.fft.fftshift(image)
    h,w =image.shape
    cx = int(w/2)
    cy = int(h/2)
    for i in range(h):
        for j in range(w):
            distance = math.sqrt((i-cx)**2+(j-cy)**2)
            if(self.Type=="IDLP"):
                if(distance>D0):
                    image[i][j]=0
            elif(self.Type=="IDHP"):
                if(distance<D0):
                    image[i][j]=0
            elif(self.Type=="BLF"):
                n = int(self.n_.text())
                image[i][j] *=1/((1+(distance/D0))**(2*n))
            elif(self.Type=="BHF"):
                n = int(self.n_.text())
                image[i][j] *=(1-1/((1+(distance/D0))**(2*n)))
            elif(self.Type=="GLF"):
                image[i][j]*=np.exp(-(distance)**2/(2*D0**2))
            elif(self.Type=="GHF"):
                image[i][j]*=(1-np.exp(-(distance)**2/(2*D0**2)))
            elif(self.Type=="Homorphic"):
                try:
                    rH=float(self.gammaH.text())
                    rL=float(self.gammaL.text())
                    c=float(self.c_.text())
                    image[i][j]*=(rH-rL)*(1-np.exp(-c*pow(distance,2)/pow(D0,2)))+rL)
                except:
                    Error_Text = "parameters haven't been input"
                    QtWidgets.QMessageBox.information(None, 'Read Me', Error_Text)
    #f_ishift=np.fft.ifftshift(image)
    image=np.fft.ifft2(image)
    image=np.abs(image)
    image = cv2.normalize(image,None,0,1,cv2.NORM_MINMAX)
```

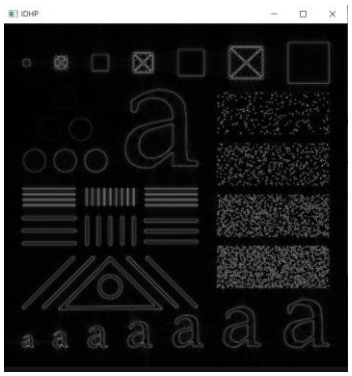
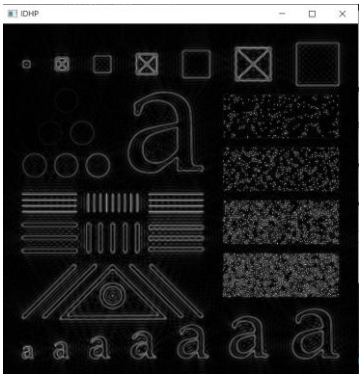
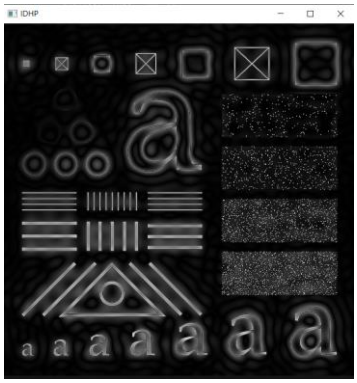
將圖片轉換到頻率域後，該函式會根據不同的 self.Type 來進行不同的處理。

Ideal Low Pass filter:



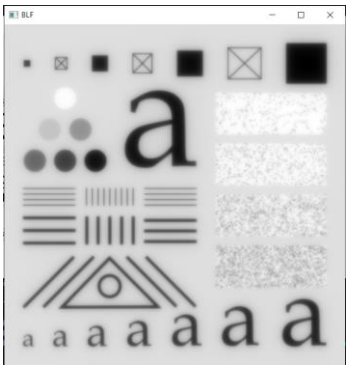
D0 =100 D0 = 60 D0 = 20

Ideal High Pass filter:

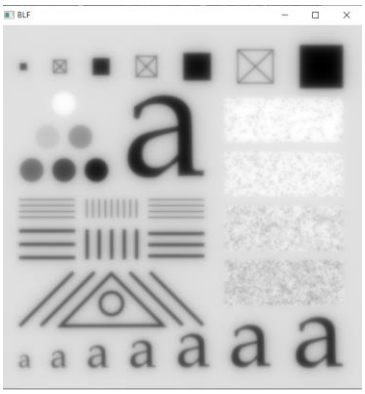




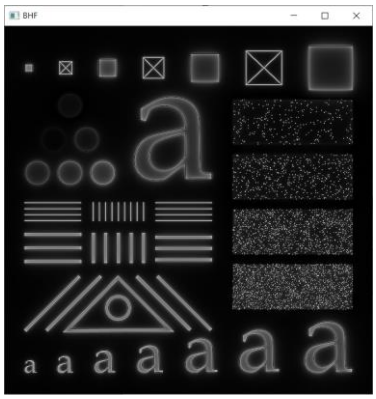
D0 =100 D0 = 60 D0 = 20

Butterworth D0=60,n=1




D0=40,n=1





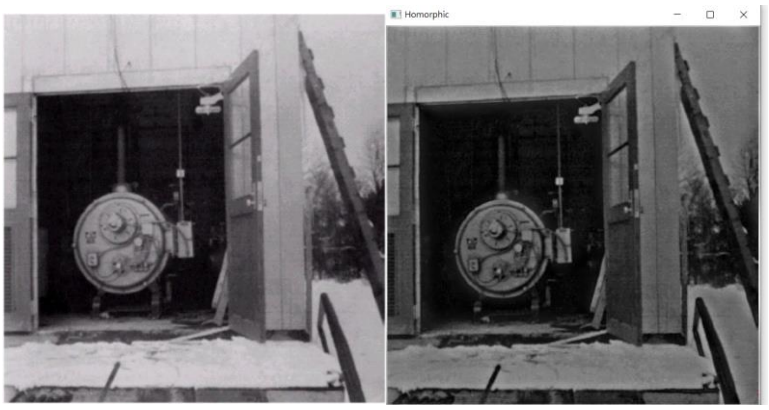
Gaussian D0=60

D0=40

從上方的結果可以發現，隨著 D0 越小，以低通濾波來說代表不為零的頻率部分越少，反轉回空間域時的資訊就越小，圖片越模糊，但對 Gaussian 以及 Butterworth 來說比較像是潤化的效果，因為它們是以一個較平緩的曲線來下降;而以高頻來說雖然仍可將輪廓表現出來，但由於可保留的頻率愈多，因此可以發現隨著 D0 越小輪廓跟細節有越來越多或越來越粗的趨勢。

Homomorphic :

根據課本，當$\gamma L < 1$ 且 $\gamma H \geq 1$時，該濾波會削弱低頻率的部分並且增強高頻率的部分，因此會使圖片的對比度增強



D0=20、 γH=4、γL=0.3、c=5

(4)

按下"Select File"選擇好圖片後,先輸入 K 值,接著選擇按下”Motion Blur”,或者”Noise”即可。程式會顯示出只有模糊或者是有雜訊及模糊的影像:

原圖　　Motion Blur　　Motion Blur + Noise

程式碼:

```
def motionblur(self):
    cv2.destroyAllWindows()
    image = np.fft.fft2(self.im)
    image = np.fft.fftshift(image)
    M,N=image.shape
    T=1
    a=b=0.1
    H = np.zeros((M + 1, N + 1), dtype = np.complex128)
    for u in range(1, M + 1):
        for v in range(1, N + 1):
            w = np.pi * (u * a + v * b)
            H[u, v] = (T / w) * np.sin(w) * np.exp(-1j * w) # formular 5-77
    H = H[1 : , 1 : ]
    G = H*image
    blur=np.fft.ifft2(G)
    g=np.abs(blur)
    blur_normalize = cv2.normalize(g,None,0,1,cv2.NORM_MINMAX)
    cv2.imshow("Motion Blur",blur_normalize)

    #Wiener
    image = np.fft.fft2(blur)
    blur = np.fft.fftshift(image)
    k= float(self.K.text())
    F_wiener=(1 / H) * (np.abs(H)**2 / (np.abs(H)**2 + k)) * G
    wiener_image=np.fft.ifft2(F_wiener)
    wiener_image=np.abs(wiener_image)
    contrast_wiener = np.abs(self.im-wiener_image)
    self.wiener_image = cv2.normalize(wiener_image,None,0,1,cv2.NORM_MINMAX)
    self.weiner_contrast = cv2.normalize(contrast_wiener,None,0,1,cv2.NORM_MINMAX)
    #inverse
    blur = np.copy(G)
    F_inverse = blur/H
    F_inverse = np.fft.ifftshift(F_inverse)
    inverse_image=np.fft.ifft2(F_inverse)
    inverse_image=np.abs(inverse_image)
    contrast_inverse = np.abs(self.im-inverse_image)
```

```
def addnoise(self):
    cv2.destroyAllWindows()
    image = np.fft.fft2((self.im))
    image = np.fft.fftshift(image)
    M,N=image.shape
    T=1
    a=b=0.1
    H = np.zeros((M + 1, N + 1), dtype = np.complex128)
    for u in range(1, M + 1):
        for v in range(1, N + 1):
            w = np.pi * (u * a + v * b)
            H[u, v] = (T / w) * np.sin(w) * np.exp(-1j * w) # formular 5-77
    H = H[1 : , 1 : ]
    G_noise = H*image
    G_noise=np.fft.ifftshift(G_noise)
    blur_noise=np.fft.ifft2(G_noise)
    blur_noise=np.abs(blur_noise)
    mean = 0
    variance = 20 # 題目設定
    noise = np.random.normal(mean, variance, self.im.size)
    noise = noise.reshape(self.im.shape[0], self.im.shape[1]).astype('uint8')
    blur_noise=blur_noise
    blur_normalize = cv2.normalize(blur_noise,None,0,1,cv2.NORM_MINMAX)
    cv2.imshow("Motion Blur",blur_normalize)

    image = np.fft.fft2(blur_noise)
    blur_noise_wiener = np.fft.fftshift(image)
    k= float(self.K.text())
    F_wiener=(1 / H) * (np.abs(H)**2 / (np.abs(H)**2 + k)) * blur_noise_wiener
    wiener_image=np.fft.ifft2(F_wiener)
    wiener_image=np.abs(wiener_image)
    contrast_wiener= np.abs(self.im-wiener_image)
    self.wiener_image = cv2.normalize(wiener_image,None,0,1,cv2.NORM_MINMAX)
    self.weiner_contrast = cv2.normalize(contrast_wiener,None,0,1,cv2.NORM_MINMAX)
```

```
image = np.fft.fft2(blur_noise)
blur_noise_inverse = np.fft.fftshift(image)
F_inverse = blur_noise_inverse/H
F_inverse = np.fft.ifftshift(F_inverse)
inverse_image=np.fft.ifft2(F_inverse)
inverse_image=np.abs(inverse_image)
contrast_inverse = np.abs(self.im-inverse_image)
```

Deblurring:

在取得模糊後的照片後，直接按下”Inverse”或者”Wiener”，程式會顯示在前段步驟的圖片結果。

```
def wienerfilter(self):

    cv2.imshow("Wiener Image",self.wiener_image)
    cv2.imshow("contrast",self.weiner_contrast)
def inversefilter(self):

    cv2.imshow("Inverse Image",self.inverse_image)
    cv2.imshow("contrast",self.inverse_contrast)
```

程式碼:

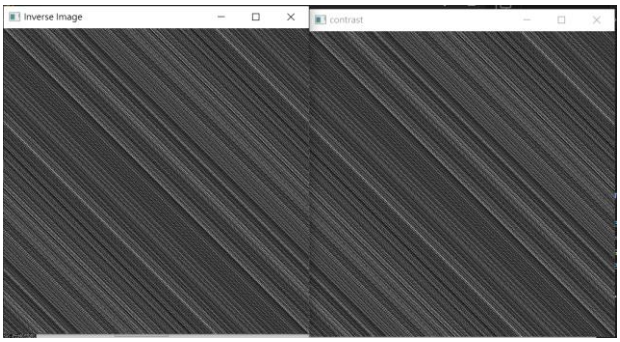
Reverse Image and Contrast between Original(Wiener,K=0.0001)

Reverse Image and Contrast between Original(Inverse)

可以發現,在沒有噪音的情形下 Inverse Filter 會有比較好的效果，這是因為該圖片的 degradation filter 已經被我們得知,並且也沒有噪音干擾，所以從原本的式子除回去就可以得到原來的圖片。而 Wiener Filter 是假設在有噪音的情形下使用的,當其沒有噪音時,從式子可發現 Wiener Filter 會相等於 Inverse Filter。下面兩張比對可發現當加上噪音的干擾,Wiener Filter 會有比較好的效果,由於噪音會影響整個 model,所以單純的使用 Inverse Filter 會得到很差的結果:


Wiener Image
contrast
Digital
Image
Processing
Digital
Image
Processing

K=0.0001


Inverse Image
contrast