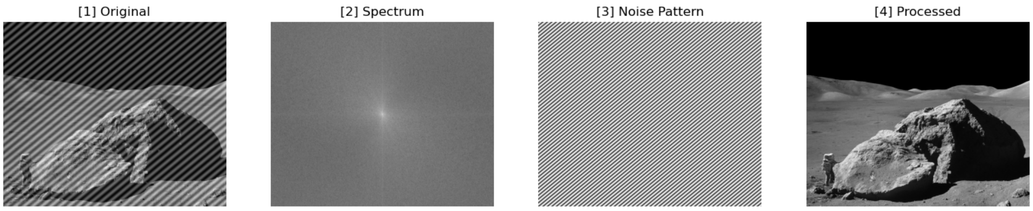
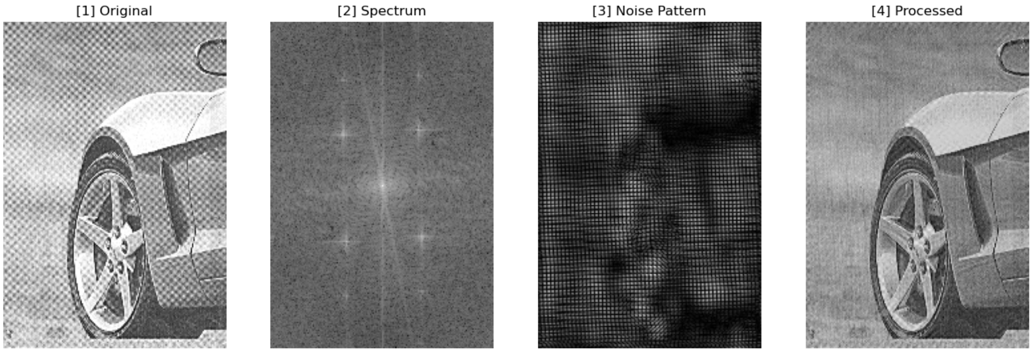
數位影像處理 DIP Chapter4_2 Homework

電機4C 洪愷尹 0710851

- 1 由附圖



- 2 由附圖



- Please **comment** and **compare** your two self-designed filters?

  第一個Filter是直接將頻譜上找到的兩個burst的點直接歸零，再轉回Spatial的資訊，成果上可以發現扣除雜訊後的圖，細節呈現很好，雜訊的Pattern消除的很乾淨。

  第二個Filter是將頻譜的4個Pair的Burst透過Butterworth Notch Reject Filter濾掉，發現扣除雜訊的後的圖，即使原圖解析度很差，效果還是蠻顯著的。

[SOURCE CODE]

```
import argparse
import numpy as np
import cv2
import matplotlib.pyplot as plt
import os
from pandas import *

parser = argparse.ArgumentParser()
parser.add_argument(
    "--image", help = "Image to process.", default="astronaut-interference.tif")
args = parser.parse_args()


def showImage(*img):
    title = ['[1] Original',
             '[2] Spectrum',
             '[3] Noise Pattern',
             '[4] Processed']

    for i in range(len(title)):
        plt.subplot(1,len(title),i+1)
        plt.imshow(img[i], cmap='gray')
        plt.title(title[i])
        plt.axis('off')

    plt.show()


def FFT(img):
    f = np.fft.fft2(img)
    fshift = np.fft.fftshift(f)
    mag = np.abs(fshift)
    log = np.log(1+mag)
    return fshift, mag, log

def IFFT(img):
    ishift = np.fft.ifftshift(img)
    iimg = np.fft.ifft2(ishift)
    imag = np.abs(iimg)
    return imag

def NotchFilter1(size, place):

    filter = np.ones(size, dtype = np.int32)

    for i in place:
        filter[i[0]][i[1]] = 0

    return filter

def Dist(u, v, M, N, uk, vk):
    #Calculate distance
    distance = ((u - M / 2 - uk)**2 + (v - N / 2 - vk)**2)**0.5
    return distance

def NotchFilter2(InterferencePoints, InterferenceSize, pairs, M, N):

    filter = np.ones((M,N), dtype=np.int32)

    for i in range(M):
        for j in range(N):
            for k in range(pairs):
                Dk_plus = Dist(i, j, M, N, InterferencePoints[k][0], InterferencePoints[k][1])
                Dk_minus = Dist(i, j, M, N, -InterferencePoints[k][0], -InterferencePoints[k][1])
                if Dk_plus == 0 or Dk_minus == 0:
                    filter[i][j] = 0
                else:
                    value = (1 /(InterferenceSize[k] / Dk_plus)**pairs) * \
                            (1 / (InterferenceSize[k] / Dk_minus)**pairs)
                    filter[i][j] *= min(1, value / 255)
    return filter

def main():
    # Load img
    path = './' + args.image
    if os.path.isfile(path):
        original = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
    else :
        print ("The file" + path +" does not exist.")

    # Reserved space
    G = np.empty(original.shape)
    Filter = np.empty(original.shape)

    # FFT
    FreqDomain, Magnitude, LogMagnitude = FFT(original)

    # Filter
    if args.image == "astronaut-interference.tif":
        # for x in range(Magnitude.shape[0]):
        #     for y in range(Magnitude.shape[1]):
        #         Magnitude[x][y] = 255 * (Magnitude[x][y] - min) / (max - min)
        # Magnitude = Magnitude.astype(np.uint8)
        # for i in range(Magnitude.shape[0]):
        #     for j in range(Magnitude.shape[1]):
        #         if Magnitude[i][j] == 70:
        #             print(i,j,Magnitude[i][j])
        # How I Found Two Points?
        InterferencePoints = np.array([[437,525], [387, 475]])
        Filter = NotchFilter1(FreqDomain.shape, InterferencePoints)
    else:
        (M, N) = FreqDomain.shape
        pairs = 4
        InterferencePoints = np.array([[205 - M / 2, 111 - N / 2], [205 - M / 2, 56.9
- N / 2],
                                        [165 - M / 2, 111 - N / 2], [165 - M / 2, 56.9
- N / 2]])
        InterferenceSize = [20, 20, 16, 16]
        Filter = NotchFilter2(InterferencePoints, InterferenceSize, pairs, M, N)

    G = FreqDomain * Filter
    FreqNoisePattern = FreqDomain * (1 - Filter)
    # IFFT
    SpatialDomain = IFFT(G)
    SpatialNoisePattern = IFFT(FreqNoisePattern)


    showImage(original, LogMagnitude, SpatialNoisePattern, SpatialDomain)

if __name__ == '__main__':
    main()
```