NTUT112-1 Digital Image Processing
Homework Assignment 4
Due Date: 12/27(Wed.) 2023

You have already learned how to transform images from the spatial domain into frequency domain. For this assignment, please try to implement **horizontal motion blurring** and **image restoration** through 4 practical exercises. When taking photos with a handheld camera, there is often motion blur due to shaky hands or fast moving objects. The real system is quite complex, but we can simulate it using a mean-like filter. For example, we can directly use an averaging filter with a neighboring distance d to achieve this application. Please download the 'lena.bmp' and 'HW4_demo.mlx' from class website and use them to finish the following functions.

Question 1: (10 %)

Please implement the function `CreateHorizontalMotionBlurredKernel` that creates a kernel of **Horizontal Motion Blur** with d in spatial domain (e.g. $h = \left\{ \frac{1}{2d+1} \mid for - d \le x \le d \right\}$).

(Hint: Built-in function "ones()" could be used.)

```
function h = CreateHorizontalMotionBlurredKernel(d)
% h = CreateHorizontalMotionBlurredKernel(d)
% for example, if d=1, h=[1/3, 1/3, 1/3];if d=3, h=[1/7, 1/7, 1/7, 1/7, 1/7, 1/7, 1/7]
     your implementation here ...
end
```

Question 2: (25%)

Please implement the function `MotionBlurInFrequencyDomain` that performs **Degradation Process in the Frequency Domain**. Note that **all input and output are in spatial domain** but **your processes have to be in frequency domain: any convolution operation is not allowed.**

(Hint: Utilize ".*" to perform element-wise multiplication.)

```
function g = MotionBlurInFrequencyDomain(f, h)
% g = MotionBlurInFrequencyDomain(f, h)
% 'f' is original image, 'h' is kernel of motion blur and 'g' is degraded image
% All inputs and output is in spatial domain but your process has to be in frequency
domain
     your implementation here ...
end
```

Question 3: **(25%)**

Please implement the function `InverseFilterInFrequencyDomain` that performs **Inverse Filter process** with an ε. As in Question 2, **all input and output are in spatial domain** but **your processes have to be in frequency domain: any deconvolution operation is not allowed.**

$$\hat{F}(u,v) = R(u,v)G(u,v), \text{ where } R(u,v) = \begin{cases} \frac{1}{H(u,v)} & ,|H(u,v)| > \varepsilon \\ 0 & ,otherwise \end{cases}$$

```
function f_hat = InverseFilterInFrequencyDomain(g, h, epsilon)
% f_hat = InverseFilterInFrequencyDomain(g, h, epsilon)
% 'g' is degraded image, 'h' is kernel of motion blur, epsilon is for R(u,v)
% All inputs and output is in spatial domain but your process has to be in frequency
domain
     your implementation here ...
end
```

Question 4: **(40%)**

Please implement the function `MyWiennerFilter` that performs basic **Wiener filtering process**. As Question 2, **all input and output are in spatial domain** but **your processes have to be in frequency domain: any deconvolution operation is not allowed.** (built-in function "wiener2()" is not allowed neither.)

From Eq 5-83 in the DIP textbook, K is a specified constant. When K **is equal to reciprocal of SNR (Signal-to-noise ratio)**, the restoration may be good. **Try another 3 different values of K to check their corresponding results. Try to explain/compare the results through MSE, and describe why extremely small K and extremely large K are not good in Wiener filtering processes.** This part will ask you to use degraded, original image and noise to calculate the SNR and incorporate it into the following revised equation from Eq. 5-81.

$$\hat{F}(u,v) = \left[\frac{R(u,v)|H(u,v)|^2}{|H(u,v)|^2+K}\right]G(u,v) \text{ , where } R(u,v) = \begin{cases} \frac{1}{H(u,v)} & ,|H(u,v)| > \varepsilon \\ 0 & ,otherwise \end{cases}$$

```
function f_hat = MyWienerFilter(g, h, K, epsilon)
% f_hat = MyWiennerFilter(g, h, K, epsilon)
% 'g' is degraded image, 'h' is kernel of motion blur, epsilon is for R(u,v)
% All inputs and output is in spatial domain but your process has to be in frequency
domain
    your implementation here ...
end

function snr = ComputeSpectrumSNR(f, n)
    F = fftshift(fft2(f));
    N = fftshift(fft2(n));
    S_n = sum((abs(N).^2), 'all');
    S_f = sum((abs(F).^2), 'all');
    snr = S_f./S_n;
end
```

Please use "HW4_demo.mlx" to finish this work and name it with your studentID.