CS 111: Programing Assignment 3: Filtering in Frequency Domain

Submission instructions:

Please **submit your code, output images and a PDF file (containing the output images) in a single zip file** to Canvas. You must also **submit the same PDF file** to Gradescope.
BOTH submissions are required for full points.
Your work is due by **11:59 p.m. on Wednesday, the 8th of May**.

Introduction:

This programming assignment is focused on understanding the frequency domain. You will create some synthetic images and observe their DFTs. You will also apply filters in the frequency domain.

You will be using some OpenCV functions and the functions given to you in **pa3.cpp**. In this assignment, you should work with gray images only. So, use CV_LOAD_IMAGE_GRAYSCALE when reading your images.

Discrete Fourier Transform:

I. DFT

a. DFT is used in signal processing to examine the frequency component of a discrete signal. DFT for a 1-D signal is defined as:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-i\frac{2\pi k}{N}n}$$

for discrete signal $x[n]$ with length of N (non-zero values are from 0 to $N-1$). Note that $X[k]$ is a periodic complex discrete signal with period of N. The lowest frequency component of $x[n]$ is $X[0]$ which is also called DC value. The highest frequency of signal is $X[\frac{N}{2}]$, or where $k = \frac{N}{2}$. If $x[n]$ is a real signal (e.g. images) the values of $X[k]$ for $k = \frac{N}{2} \dots N-1$ is complex conjugate of values of $X[k]$ for $k = 1 \dots \frac{N}{2}$.

b. We can recover our signal using the DFT values. This operation is Inverse Fourier Transform and it is defined as:

$$x[n] = \sum_{k=0}^{N-1} X[k] e^{i\frac{2\pi n}{N}k}$$

c. In image processing, we deal with 2-D signals. The DFT can be extended for 2-D signals as well. If we assume that our image is a 2-D signal as $x[m,n]$ and the size of the image is $M \times N$, then the DFT and IDFT for this signal are:

$$DFT: X[k,l] = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} x[m,n] e^{-i\frac{2\pi l}{N}n} e^{-i\frac{2\pi k}{M}m}$$

$$IDFT: x[m,n] = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X[k,l] e^{-i\frac{2\pi n}{N}l} e^{-i\frac{2\pi m}{M}k}$$

In this assignment, the functions for calculating DFT and IDFT are provided. Functions *DFT(Mat Input, Mat& Real, Mat& Imag)* and *IDFT(Mat& Input, Mat Real, Mat Imag)* calculate the DFT and IDFT, respectively.

d. One of the ways to understand the DFT values is by converting the complex number (Cartesian coordinate) into magnitude and phase (Polar coordinate). Higher values in magnitude tell us that the signal has more of those frequencies. In this assignment, we use the OpenCV functions for converting matrices with complex numbers into matrices with magnitude and phase. The reverse of this is done using a single OpenCV function *polarToCart*.

```
// R is the matrix of real parts (Input)
// I is the matrix of imaginary parts (Input)
magnitude(R, I, M);
phase(R, I, P);
// M is the matrix of magnitude (Output)
// P is the matrix of phase (output)
polarToCart(M, P, R, I)
// M and P are input and R and I are output
```

e. We are more comfortable with visualizing the DFT values such that the DC value ($k = 0, l = 0$) at the center. As mentioned in I.1, the DFT values are periodic and the second half is the complex conjugate of the first half. So, by bringing the DC value at the center the values will be in the range of $k = -\frac{N}{2} \dots 0 \dots \frac{N}{2}$. In this assignment, the function *DFTShift(Mat& I)* is provided that applies this shifting on the DFT values I.

f. The magnitude and phase values are not integers and their dynamic range might be greater than 0-255. In order to save the magnitude and phase as an image we should normalize them. To do so, you can use OpenCV *normalize* function, and then convert it to CV_8UC1.

```
// I input matrix to be normalized
// J output matrix normalized
normalize(I, J, 0, 255, NORM_MINMAX);
// K output matrix of type CV_8UC1
J.convertTo(K, CV_8UC1);
```

II. Creating Images

a. In order to examine how the DFT works, we want to create some 512x512 gray image composed of sinusoids with different frequencies. You should implement the following formulae in images:

$$I_1(x,y) = 1 + sin(0.1\pi x)$$

$$I_2(x,y) = 1 + cos\ cos\ (0.2\pi y)$$

$$I_3(x,y) = 1 + cos\ cos\ (0.4\pi x)$$

$$I_4(x,y) = 1 + sin(0.15\pi \sqrt{x^2 + y^2})$$

$$I_5(x,y) = 1 + sin(0.35\pi \sqrt{x^2 + y^2})$$

where x is the column and y is the row values.

b. Note that the matrix should be of type of CV_32FC1 to keep the floating point values. After implementing the formulae, normalize the Mat and convert it into CV_8UC1 the same way explained in Part-I.f.

Submit the images names "I1.png" to "I5.png". You don't need to provide the code for the generating these images.

c. Once you create the images, you should take the DFT of the images using the functions provided (Part-I.c). Then, convert

the DFT values into magnitude and phase (Part-I.d). Then, apply *DFTShift* to bring the DC value to the center (Part-I.e). Finally, normalize magnitudes and save them as images "mag1.png" to "mag5.png", respectively.

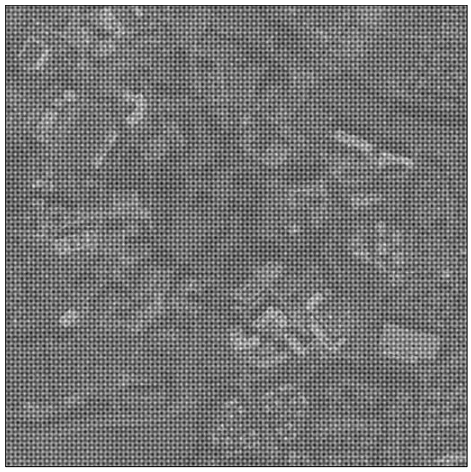
Submit the images "mag1.png" to "mag5.png". You don't need to provide the code for the steps you did to create these images

Notch Filter

III. Interference

a. Due to some reason the signal may be contaminated with some periodic noise. In medical signal processing, it is very common that the vital signals (e.g. ECG, EKG) being corrupted by addition of some noise coming from the AC power. In the United States, the AC power has frequency of 60 Hz, and the medical signal has some added noise of this frequency. So, it is common that this signal should be filtered out before further processing.

b. In this assignment, we intentionally corrupted a gray image (uci.jpg) by adding a certain frequency of sinusoids. By, taking the DFT of the noisy image and looking into the magnitude of the DFT, you can estimate the frequency of the added noise.

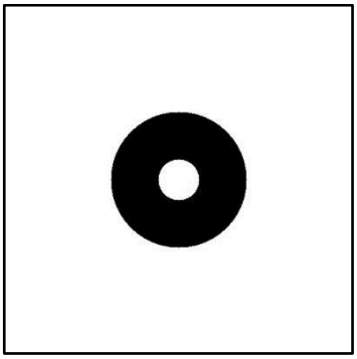


IV. Multiplying in Frequency Domain

a. Recall that convolution and multiplication have the property of duality in spatial and frequency (DFT) domains. This means that convolution in the spatial domain is equal to multiplication in frequency domain and vice versa. In frequency domain, since

we can find the noise frequencies, we can remove those frequencies by setting their magnitude to zero. We can do that by multiplying a mask on DFT magnitude to set the unwanted frequencies to zero.

b. As discussed in section I, DFT is an invertible mapping, so we can recover the image by applying IDFT on the DFT values. So, after any manipulation in the frequency domain we can apply IDFT and create the image. For this reason, we can create masks that block unwanted frequencies. In this assignment, we want a mask to set the magnitude of the unwanted noise to zero. So, this mask should act as a Notch filter. A Notch filter is the opposite of a bandpass filter (also known as bandstop filter), therefore, it stops some bands of frequencies. A 2-D Notch filter is like a black ring in a white square, where white means 1, and black means 0, shown in the image below:



c. In this assignment, function *NotchFilter(int s, int lowerCutOff, int upperCutOff)* returns a *Mat* of type 'float' with size *sxs*, and the given cut-off frequencies. The *lowerCutOff* is the radius of inner circle and the *upperCutOff* is the radius of the outer circle. The values for cut-off frequency should be up to the half of the image size. You should create the appropriate Notch filter to be multiplied by the magnitude of the DFT of the noisy image, to cancel the periodic noise from the image.

d. However, it is not as simple as canceling the unwanted noise and recovering the the original image. Recall that box functions are the dual of Sinc functions. That is why if you convolve your image with a box filter, you will introduce some leakage frequencies, since the DFT of a box filter is a Sinc. Here, if you multiply the DFT by the mask (which is like a box function), it is similar to convolving the image with the Sinc function. Since the Sinc function has "rings", you will see ringing effects in your image after IDFT.

e. In order to prevent introducing ringing effects into our image, we should smooth our mask by applying some Gaussian filter.

Here, you don't need to write the Gaussian filter, and you can use the OpenCV function *GaussianBlur*, to do that for you.

```
// I input mask
// J output smoothed mask
// s size of flter (e.g. 9)
// sigma: std of the gaussian (e.g. 5)
GaussianBlur(I, J, Size(s,s), sigma);
```

f. Here is the steps you should do:

i. Calculate DFT of "uci.jpg"
ii. Calculate magnitude from real part and imaginary part
iii. Apply *DFTShift* to center the DC value
iv. Using *normalize* and *convertTo*, visualize the normalized magnitude of the image (don't overwrite the magnitude since the original magnitude is needed for IDFT)
v. Spot the bright points in the magnitude image
vi. Use *NotchFilter* to create a ring that removes the bright spots
vii. Smooth the ring using *GaussianBlur* function
viii. Multiply the smoothed ring with magnitude (the actual magnitude, not the normalized one)
ix. Check if the bright spots are gone in magnitude image (if not go to Part-vi)
x. Use *DFTShift* to bring the magnitude back to its original, unshifted form
xi. Use *polarToCart* to convert magnitude and phase into real part and imaginary parts
xii. Use IDFT function to calculate the image from the manipulated DFT
xiii. If the image looks good, save and submit it, otherwise repeat the process again, by changing the parameters.

Submit the recovered image "uci_denoised.jpg" and your smooth mask "mask.png".