Homework3

Cho Minsung

2021/02/11

1. GW 4.4. Consider the continuous function f(t) = sin(2πn).

   (a) What is the period of f(t)?
   - The period of f(t) is 1s.

   (b) What is the frequency of f(t)?
   - The frequency is 1Hz.

   The Fourier transform, F(μ), of f(t) is purely imaginary (Problem 4.3), and because the transform of the sampled data consists of periodic copies of F(μ), the transform of the sampled data, $\tilde{F}(\mu)$, will also be purely imaginary. Draw a diagram similar to Fig. 4.6, and answer the following questions based on your diagram (assume that sampling starts at = 0).

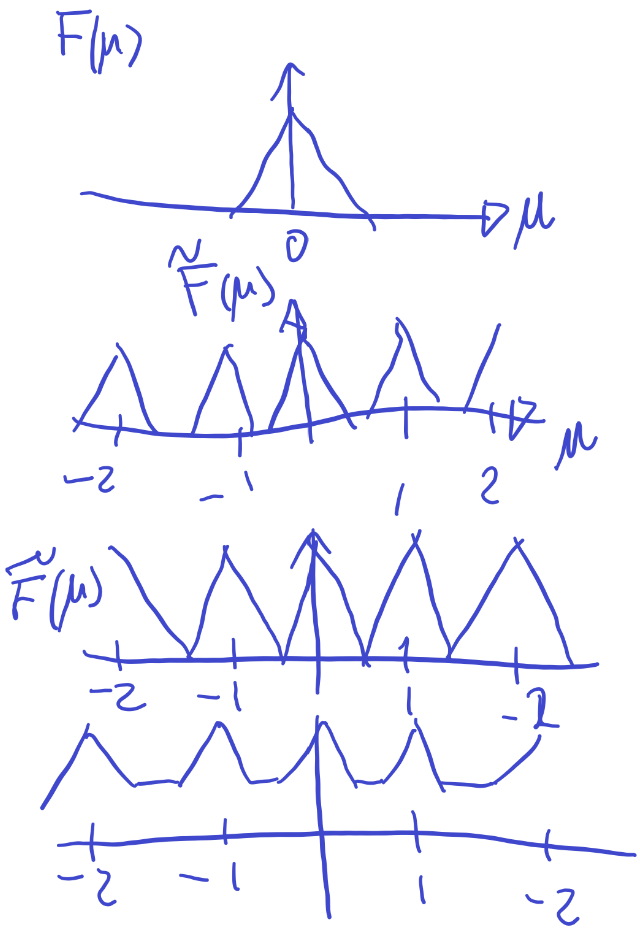


Figure 1: GW 4.4

(c) What would the sampled function and its Fourier transform look like in general if (t) is sampled at a rate higher than the Nyquist rate?

- It would look like last graph if the sampling period is smaller than the Nyquist rate.

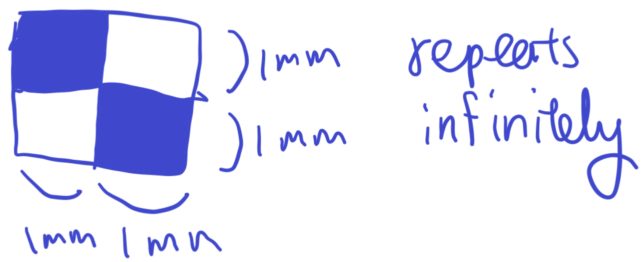
(d) What would the sampled function look like in general if (t) is sampled at a rate lower than the Nyquist rate?

- It would look like the second graph but there would be twice the space in between each periods.

(e) What would the sample function look like if f(t) is sampled at the Nyquist rate with samples taken at t = 0, ΔT, 2 ΔT, ... ?

- It would look like the third graph.

2. GW 4.12. Consider a checkerboard image in which each square is 1×1 mm. Assuming that the image extends infinitely in both coordinate directions, what is the minimum sampling rate (in samples/mm) required to avoid aliasing?



Period = 2mm

Nyquist = 4 mm

Figure 2: GW 4.12

3. GW 4.32. As expected in Eq. (4.9-1), it is possible to obtain the transfer function, H_{HP}, of a highpass filter from the trasnfer function of a lowpass filter as $H_{HP} = 1 - H_{LP}$

   Using the information given in Problem 4.31, what is the form of the *spatial domain* Gaussian highpass filter?

   - You take the inverse Fourier transform.

   $h_{HP} = \mathcal{F}^1[1 - H_{LP}(u,v)] = \mathcal{F}^1[1] - \mathcal{F}^1[H_{LP}(u,v)] = \delta(0) - \sqrt{2\pi}\sigma e^{-2\pi^2\sigma^2(x^2+y^2}$

4. GW 4.39 As illustrated in Fig. 4.59, combining high-frequency emphasis and histogram equalization is an effective method for achieving edge sharpening and contrast enhancement.

   (a) Show whether or not it matters which process is applied first.
   - If histogram equalization were to be performed first, it will be different form the process where the spacial filter is used to filter the image. So, the order does matter.

   (b) If the order does not matter, give a rationale for using one or the other method first.
   - Histogram equalization smooths out the contrast. High pass filter sharpens the image contrast. Therefore, the image should be filtered and equalized.

5. GW 4.43 A skilled medical technician is assigned the job of inspecting a certain class of images generated by an electron microscope. In order to simplify the inspection task, the technician decides to use digital image enhancement and, to this end, examines a set of representative images and finds the following problems: (1) bright, isolated dots that are of no interest; (2) lack of sharpness; (3) not enough contrast in some images; and (4) shifts in the average intensity, when this value should be V to perform correctly certain intensity measurements.The technician wants to correct these problems and then display in white all intensities in a band between I_1 and I_2, while keeping normal tonally in the remaining intensities. Propose a sequence of processing steps that the technician can follow to achieve the desired goal. You may use techniques from both Chapters 3 and 4.

   - Median filtering can get rid of problem (1).

   - High pass filter can sharpen the image and solve the problem (2).

   - Histogram equalization can solve the contrast problem (3).

   - Shift in average intensity can be solved by finding the average intensity and subtracting it from each pixels like in (4).

6. Spatial Domain Filtering

   The following question operates on the city.jpg image.

   (a) Perform image smoothing using a 7 × 7 averaging filter and a Gaussian filter with $\sigma = 0.5$ and 3. Compare the outputs.

Figure 3: Original(top left), median blur(top right), 0.5 Gaussian blur(bottom left), 3 Gaussian blur(bottom right)

(b) Perform edge enhancement using the Sobel operator (Matlab's default parameters). Repeat using the Laplacian and Laplacian of Gaussian operators. compare the outputs.

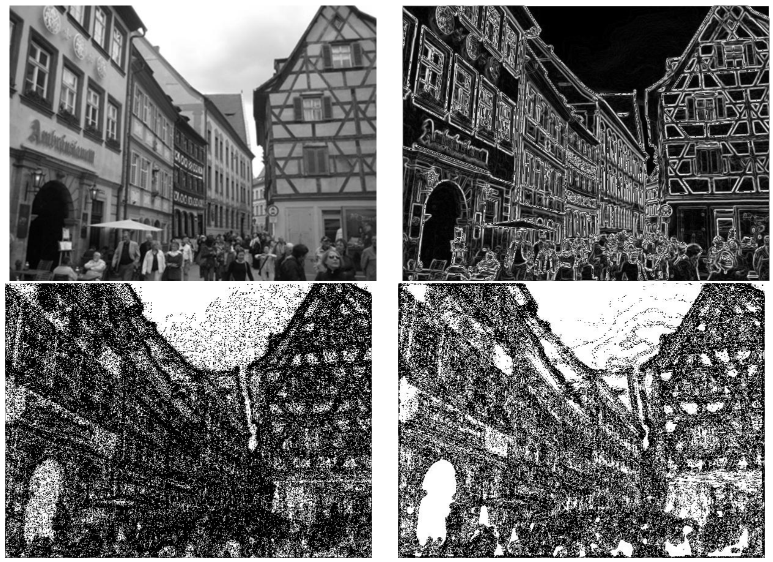


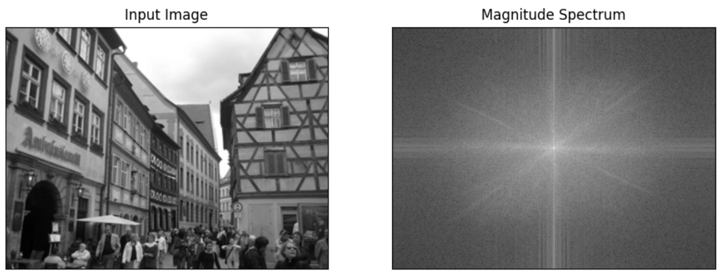
Figure 4: Original(top left), Sobel edge detection(top right), Laplacian edge detection(bottom left), 3 Gaussian blur and Laplacian edge detection(bottom right)

The laplacian edge detection only used the 3x3 size kernel to compare with Sobel edge detection one-to-one. More detail can be found in the code.

7. Frequency Domain Filtering

   The following question operates on the city.jpg image.

   (a) Find the Fourier transform of the image. Be sure to center the frequencies.



(b) Perform image smoothing in the frequency domain using the filters defined in the previous problem. Compare the output images from the two methods (spatial and frequency) and the time for operation.

(c) Perform edge enhancement using the filters defined in the previous problem.

1.4844024181365967 seconds took for the execution.

(d) Define a lowpass filter in the frequency domain with radius of 1/4 the height. Show the result. Repeat with a similar sized Gaussian and compare the results. Give the σ parameter you used and show the output transform image.

Sigma of 7 is used.

(e) Repeat with a rectangular filter with the same dimension as the ideal lowpass. Compare the results between the ideal filter and the rectangular approximation.

I did not have time to finish this part.

```python
from PIL import Image
import numpy as np
import math

IN_FILE = 'hw3/city.jpg'

pillow_img = Image.open(IN_FILE).convert('L')
pillow_img.save('hw3/city_grayscale.jpg')
IMG_W, IMG_H = pillow_img.size
img = np.array(pillow_img)

output_img = np.zeros(shape=(IMG_H, IMG_W), dtype=np.int8)
output_img[0:3,:] = img[0:3,:]
output_img[IMG_H-3:IMG_H,:] = img[IMG_H-3:IMG_H:]
output_img[:,0:3] = img[:,0:3]
output_img[:,IMG_W-3:IMG_W] = img[:,IMG_W-3:IMG_W]

temp = 0
for i in range(3, IMG_H-3):
    for j in range(3, IMG_W-3):
        for k in range(7):
            for l in range(7):
                temp += img[i+3-k, j+3-l]
        output_img[i, j] = temp / (7 * 7)
        temp = 0

output_file = Image.fromarray(np.uint8(output_img))
```

```python
output_file.save('hw3/city_output.jpg')

# Gaussian kernel with sigma of 0.5.
gaussian_point5 = np.zeros(shape=(7,7), dtype=float)
# 2nd distribution
gaussian_point5[1,1] = 0.000002
gaussian_point5[1,5] = 0.000002
gaussian_point5[5,1] = 0.000002
gaussian_point5[5,5] = 0.000002
gaussian_point5[2,1] = 0.000212
gaussian_point5[4,1] = 0.000212
gaussian_point5[1,2] = 0.000212
gaussian_point5[5,2] = 0.000212
gaussian_point5[1,4] = 0.000212
gaussian_point5[5,4] = 0.000212
gaussian_point5[2,5] = 0.000212
gaussian_point5[4,5] = 0.000212
gaussian_point5[3,1] = 0.000922
gaussian_point5[1,3] = 0.000922
gaussian_point5[5,3] = 0.000922
gaussian_point5[3,5] = 0.000922
# 1st distribution
gaussian_point5[2,2] = 0.024745
gaussian_point5[4,2] = 0.024745
gaussian_point5[2,4] = 0.024745
gaussian_point5[4,4] = 0.024745
gaussian_point5[3,2] = 0.10739
gaussian_point5[2,3] = 0.10739
gaussian_point5[4,3] = 0.10739
gaussian_point5[3,4] = 0.10739
# Center
gaussian_point5[3,3] = 0.466064

# Gaussian kernel with sigma of 3.
gaussian_3 = np.zeros(shape=(7,7), dtype=float)
# 3rd distribution
gaussian_3[0,0] = 0.011362
gaussian_3[0,6] = 0.011362
gaussian_3[6,0] = 0.011362
gaussian_3[6,6] = 0.011362
gaussian_3[0,1] = 0.014962
gaussian_3[0,5] = 0.014962
gaussian_3[1,0] = 0.014962
gaussian_3[1,6] = 0.014962
gaussian_3[5,0] = 0.014962
gaussian_3[5,6] = 0.014962
gaussian_3[6,1] = 0.014962
gaussian_3[6,5] = 0.014962
gaussian_3[0,2] = 0.017649
gaussian_3[0,4] = 0.017649
gaussian_3[2,0] = 0.017649
gaussian_3[2,6] = 0.017649
gaussian_3[4,0] = 0.017649
gaussian_3[4,6] = 0.017649
gaussian_3[6,2] = 0.017649
gaussian_3[6,4] = 0.017649
gaussian_3[3,0] = 0.018648
```

```
gaussian_3[0,3] = 0.018648
gaussian_3[3,6] = 0.018648
gaussian_3[6,3] = 0.018648
# 2nd distribution
gaussian_3[1,1] = 0.019703
gaussian_3[1,5] = 0.019703
gaussian_3[5,1] = 0.019703
gaussian_3[5,5] = 0.019703
gaussian_3[2,1] = 0.02324
gaussian_3[4,1] = 0.02324
gaussian_3[1,2] = 0.02324
gaussian_3[5,2] = 0.02324
gaussian_3[1,4] = 0.02324
gaussian_3[5,4] = 0.02324
gaussian_3[2,5] = 0.02324
gaussian_3[4,5] = 0.02324
gaussian_3[3,1] = 0.024556
gaussian_3[1,3] = 0.024556
gaussian_3[5,3] = 0.024556
gaussian_3[3,5] = 0.024556
# 1st distribution
gaussian_3[2,2] = 0.027413
gaussian_3[4,2] = 0.027413
gaussian_3[2,4] = 0.027413
gaussian_3[4,4] = 0.027413
gaussian_3[3,2] = 0.028964
gaussian_3[2,3] = 0.028964
gaussian_3[4,3] = 0.028964
gaussian_3[3,4] = 0.028964
# Center
gaussian_3[3,3] = 0.030603

gauss_point5_img = np.zeros(shape=(IMG_H, IMG_W), dtype=np.int8)
gauss_point5_img[0:3,:] = img[0:3,:]
gauss_point5_img[IMG_H-3:IMG_H,:] = img[IMG_H-3:IMG_H:]
gauss_point5_img[:,0:3] = img[:,0:3]
gauss_point5_img[:,IMG_W-3:IMG_W] = img[:,IMG_W-3:IMG_W]

for i in range(3, IMG_H-3):
    for j in range(3, IMG_W-3):
        for k in range(7):
            for l in range(7):
                gauss_point5_img[i, j] += int(img[i+3-k, j+3-l] * gaussian_point5[k, l])

output_file = Image.fromarray(np.uint8(gauss_point5_img))
output_file.save('hw3/city_gauss_point5_output.jpg')

gauss_3_img = np.zeros(shape=(IMG_H, IMG_W), dtype=np.int8)
gauss_3_img[0:3,:] = img[0:3,:]
gauss_3_img[IMG_H-3:IMG_H,:] = img[IMG_H-3:IMG_H:]
gauss_3_img[:,0:3] = img[:,0:3]
gauss_3_img[:,IMG_W-3:IMG_W] = img[:,IMG_W-3:IMG_W]

for i in range(3, IMG_H-3):
    for j in range(3, IMG_W-3):
        for k in range(7):
```

```
            for l in range(7):
                gauss_3_img[i, j] += int(img[i+3-k, j+3-l] * gaussian_3[k, l])

output_file = Image.fromarray(np.uint8(gauss_3_img))
output_file.save('hw3/city_gauss_3_output.jpg')

sobelx_kern = np.zeros(shape=(3,3), dtype = np.int16)
sobelx_kern[0,0]=1
sobelx_kern[0,2]=1
sobelx_kern[0,1]=2
sobelx_kern[2,0]=-1
sobelx_kern[2,2]=-1
sobelx_kern[2,1]=-2

sobely_kern = np.zeros(shape=(3,3), dtype = np.int16)
sobely_kern[0,0]=1
sobely_kern[2,0]=1
sobely_kern[1,0]=2
sobely_kern[0,2]=-1
sobely_kern[2,2]=-1
sobely_kern[1,2]=-2

sobel_img = np.zeros(shape=(IMG_H, IMG_W), dtype = np.int16)
sobel_img[0:1,:] = img[0:1,:]
sobel_img[IMG_H-1:IMG_H,:] = img[IMG_H-1:IMG_H:]
sobel_img[:,0:1] = img[:,0:1]
sobel_img[:,IMG_W-1:IMG_W] = img[:,IMG_W-1:IMG_W]

sobelx = 0
sobely = 0
for i in range(1, IMG_H-1):
    for j in range(1, IMG_W-1):
        for k in range(3):
            for l in range(3):
                sobelx += img[i-k+1, j-l+1] * sobelx_kern[k, l]
                sobely += img[i-k+1, j-l+1] * sobely_kern[k, l]
        sobel_img[i, j] = math.sqrt(sobelx**2+sobely**2)
        sobelx = 0
        sobely = 0

output_file = Image.fromarray(np.uint8(sobel_img))
output_file.save('hw3/city_sobel_output.jpg')

laplacian_img = np.zeros(shape=(IMG_H, IMG_W), dtype = np.int8)
laplacian_img[0:1,:] = img[0:1,:]
laplacian_img[IMG_H-1:IMG_H,:] = img[IMG_H-1:IMG_H:]
laplacian_img[:,0:1] = img[:,0:1]
laplacian_img[:,IMG_W-1:IMG_W] = img[:,IMG_W-1:IMG_W]

first_der = np.zeros(shape=(IMG_H, IMG_W, 4), dtype = np.int8)

for i in range(1, IMG_H-1):
    for j in range(1, IMG_W-1):
            first_der[i, j, 0] = int((img[i,j+1]-img[i,j-1])/2)
            first_der[i, j, 1] = int((img[i+1,j]-img[i-1,j])/2)
            first_der[i, j, 2] = int((img[i+1,j+1]-img[i-1,j-1])/2)
```

```
            first_der[i, j, 3] = int((img[i-1,j+1]-img[i+1,j-1])/2)

for i in range(1, IMG_H-1):
    for j in range(1, IMG_W-1):
            if (first_der[i,j+1, 0]-first_der[i,j-1,0])/2 == 0 or (first_der[i+1,j, 1]-first_der[i-1,j, 1])/2 == 0 or ((first_der[i+1,j+1, 2]-first_der[i-1,j-1, 2])/2) == 0 or ((first_der[i-1,j+1, 3]-first_der[i+1,j-1, 3])/2) == 0:
                laplacian_img[i, j] = 255
            else:
                laplacian_img[i, j] = 0

output_file = Image.fromarray(np.uint8(laplacian_img))
output_file.save('hw3/city_laplacian_output.jpg')

gaussian_laplacian_img = np.zeros(shape=(IMG_H, IMG_W), dtype = np.int8)
gaussian_laplacian_img[0:1,:] = img[0:1,:]
gaussian_laplacian_img[IMG_H-1:IMG_H,:] = img[IMG_H-1:IMG_H:]
gaussian_laplacian_img[:,0:1] = img[:,0:1]
gaussian_laplacian_img[:,IMG_W-1:IMG_W] = img[:,IMG_W-1:IMG_W]

first_der = np.zeros(shape=(IMG_H, IMG_W, 4), dtype = np.int8)

for i in range(1, IMG_H-1):
    for j in range(1, IMG_W-1):
            first_der[i, j, 0] = int((gauss_3_img[i,j+1]-gauss_3_img[i,j-1])/2)
            first_der[i, j, 1] = int((gauss_3_img[i+1,j]-gauss_3_img[i-1,j])/2)
            first_der[i, j, 2] = int((gauss_3_img[i+1,j+1]-gauss_3_img[i-1,j-1])/2)
            first_der[i, j, 3] = int((gauss_3_img[i-1,j+1]-gauss_3_img[i+1,j-1])/2)

for i in range(1, IMG_H-1):
    for j in range(1, IMG_W-1):
            if (first_der[i,j+1, 0]-first_der[i,j-1,0])/2 == 0 or (first_der[i+1,j, 1]-first_der[i-1,j, 1])/2 == 0 or ((first_der[i+1,j+1, 2]-first_der[i-1,j-1, 2])/2) == 0 or ((first_der[i-1,j+1, 3]-first_der[i+1,j-1, 3])/2) == 0:
                gaussian_laplacian_img[i, j] = 255
            else:
                gaussian_laplacian_img[i, j] = 0

output_file = Image.fromarray(np.uint8(gaussian_laplacian_img))
output_file.save('hw3/city_gaussian_laplacian_output.jpg')
```

Listing 1: number 6 code

```
from scipy import fftpack
from PIL import Image, ImageDraw, ImageFilter
import numpy as np
from matplotlib import pyplot as plt
from numpy.fft import fft2, ifft2, fftshift, ifftshift
import math
import cv2
import time
```

```

IN_FILE = 'hw3/city.jpg'

pillow_img = Image.open(IN_FILE).convert('L')
IMG_W, IMG_H = pillow_img.size
img = np.array(pillow_img)

dft = cv2.dft(np.float32(img),flags = cv2.DFT_COMPLEX_OUTPUT)
dft_shift = np.fft.fftshift(dft)
magnitude_spectrum = 20*np.log(cv2.magnitude(dft_shift[:,:,0], dft_shift[:,:,1]))
plt.figure(figsize=(11,6))
plt.subplot(121),plt.imshow(img, cmap = 'gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(magnitude_spectrum, cmap = 'gray')
plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])
plt.show()


start_time = time.time()

figure_size = 7

new_image = cv2.medianBlur(img, figure_size)
plt.figure(figsize=(11,6))
plt.subplot(121), plt.imshow(img, cmap='gray'),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(122), plt.imshow(new_image, cmap='gray'),plt.title('Median Filter')
plt.xticks([]), plt.yticks([])
plt.show()

print("--- %s seconds ---" % (time.time() - start_time))

new_image = cv2.Laplacian(img,cv2.CV_64F)
plt.figure(figsize=(11,6))
plt.subplot(131), plt.imshow(img, cmap='gray'),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(132), plt.imshow(new_image, cmap='gray'),plt.title('Laplacian')
plt.xticks([]), plt.yticks([])
plt.subplot(133), plt.imshow(img + new_image, cmap='gray'),plt.title('Resulting image')
plt.xticks([]), plt.yticks([])
plt.show()

rows, cols = img.shape
crow,ccol = rows//2 , cols//2
mask = np.zeros((rows,cols,2),np.uint8)
mask[crow-30:crow+30, ccol-30:ccol+30] = 1
fshift = dft_shift*mask
f_ishift = np.fft.ifftshift(fshift)
img_back = cv2.idft(f_ishift)
img_back = cv2.magnitude(img_back[:,:,0],img_back[:,:,1])
plt.figure(figsize=(11,6))
```

```
plt.subplot(121),plt.imshow(img, cmap = 'gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(img_back, cmap = 'gray')
plt.title('Low Pass Filter'), plt.xticks([]), plt.yticks([])
plt.show()

image = cv2.imread('hw3/city.jpg')
image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

new_image = cv2.GaussianBlur(image, (figure_size, figure_size),0)
plt.figure(figsize=(11,6))
plt.subplot(121), plt.imshow(cv2.cvtColor(image, cv2.COLOR_HSV2RGB)
    ),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(122), plt.imshow(cv2.cvtColor(new_image, cv2.
    COLOR_HSV2RGB)),plt.title('Gaussian Filter')
plt.xticks([]), plt.yticks([])
plt.show()
```

Listing 2: number 7 code