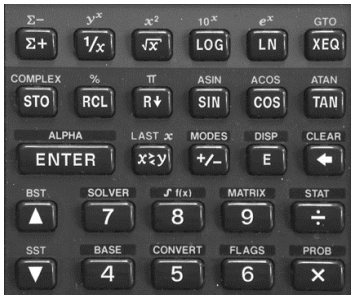
Digital Image Processing (2022 Fall) HW#2.1

r11942137 張舜程

1. Problem 1

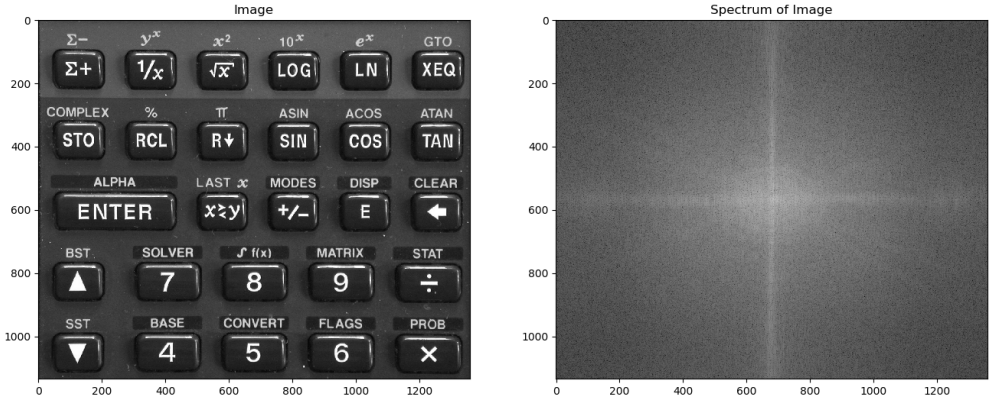
Follow the steps outlined in Section 4.7 to repeat the experiment described in Example 4.15, pp. 271-273, on the vertical Sobel kernel shown in Fig. 4.38(a) and the test image "keyboard.tif." You may use any existing library to compute the Fourier transform.



a. Show the Fourier spectrum of the test image "keyboard.

Use np.fft.fft2 to transform the image to the frequency domain.
Use np.fft.fftshift to center the spectrum.
Use log transformation with c=1 to bring out the detail of the spectrum.

```
img = cv2.imread("./keyboard.tif", cv2.IMREAD_GRAYSCALE)
f = np.fft.fft2(img)
f_shift = np.fft.fftshift(f)
log_mag_spectrum = 1 * np.log(1+ np.abs(f_shift))
```



b. Enforce odd symmetry on the kernel. Show the kernel.

To enforce the Sobel kernel $h(x, y)$ to be odd symmetry, i.e:

$$h(x, y) = -h(M - x, N - y)$$

According to Example 4.15, just adding zeros to be leading row and column.

```
sobel = np.array([[-1,0,1],
                  [-2,0,2],
                  [-1,0,1]])
sobel_odd_sym = np.zeros([4,4])
sobel_odd_sym[1:, 1:] = sobel
print("Original sobel kernel:\n", sobel)
print("After enforcing odd symmetry:\n", sobel_odd_sym.astype(np.int32))
```

```
C:\Users\Johnny\Anaconda3\python.exe D:/台大/課程/DIP/作業/HW/HW#2/p11.py
Original sobel kernel:
 [[-1  0  1]
 [-2  0  2]
 [-1  0  1]]
After enforcing odd symmetry:
 [[ 0  0  0  0]
 [ 0 -1  0  1]
 [ 0 -2  0  2]
 [ 0 -1  0  1]]
```

c. Show the result of frequency-domain filtering of the test image using the vertical Sobel kernel.

I use the shift_mask() function to multiply image by $-(1)^{x+y}$ in spatial domain. Because the kernel size is 3x3, I use zero padding to get PxQ image.

$$P = height + 3 - 1$$
$$Q = width + 3 - 1$$

Let $f(x, y)$ is the padded and shifted image, and $h(x, y)$ is the padded and shifted sobel x kernel.

Step1. Converting $f(x, y)$ and $h(x, y)$ into frequency domain.

$$f(x, y) \rightarrow F(u, v)$$
$$h(x, y) \rightarrow H(u, v)$$

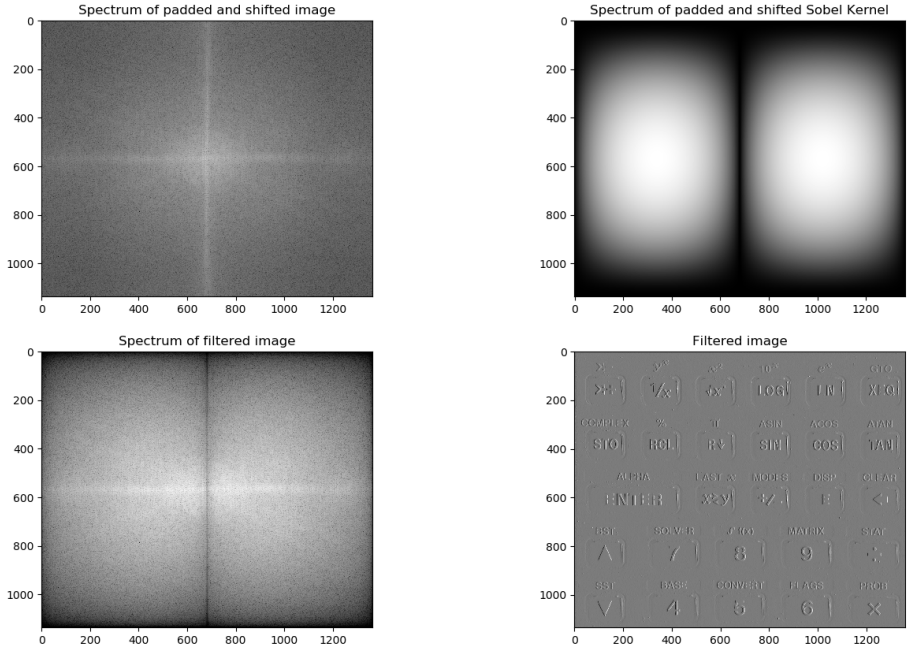
Step2. Multiplying the two spectrum to generate filtered spectrum $G(u, v)$.

$$G(u, v) = F(u, v)\, H(u, v)$$

Step3. Convert back to spatial domain.

$$G(u, v) \rightarrow g(x, y) = f(x, y) * h(x, y)$$

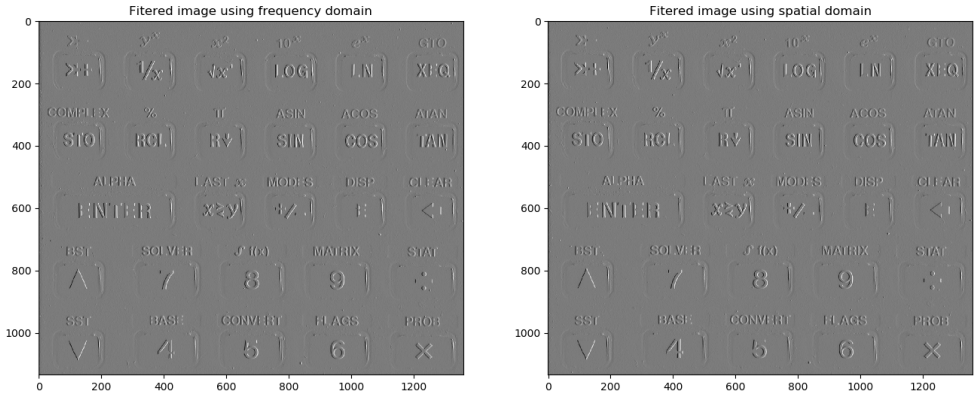
```
# create F(u,v)
padded_img = np.zeros([P,Q])
padded_img[:h, :w] = img
shift = shift_mask(h, w)
padded_shifted_img = np.zeros([P,Q])
padded_shifted_img[:h, :w] = img * shift
f_padded_shifted_img = np.fft.fft2(padded_shifted_img)
log_f_padded_shifted_img = 1 * np.log(1+ np.abs(f_padded_shifted_img))

# create H(u,v)
kernel_size = sobel_odd_sym.shape[0]
shifted_sobel = np.zeros([P,Q])
shifted_sobel[:kernel_size, :kernel_size] = sobel_odd_sym * shift_mask(kernel_size,k
f_shifted_sobel = np.fft.fft2(shifted_sobel)
log_f_shifted_sobel = 1 * np.log(1+ np.abs(f_shifted_sobel))

# G(u,v) = F(u,v)*H(u,v)
f_filtered_img = f_padded_shifted_img * f_shifted_sobel
log_f_filtered_img = 1 * np.log(1 + np.abs(f_filtered_img))

# Get back to spatial domain
img_filtered = np.fft.ifft2(f_filtered_img).real * shift_mask(P, Q)
img_filtered = img_filtered[:h, :w]
```



d. Compare your result in (c) with the result of space-domain filtering.

```
# Compare with filtering in spatial domain
img_spatial_filtered = conv2d(img, sobel)
img_spatial_filtered_magnitude = np.abs(img_spatial_filtered)
```

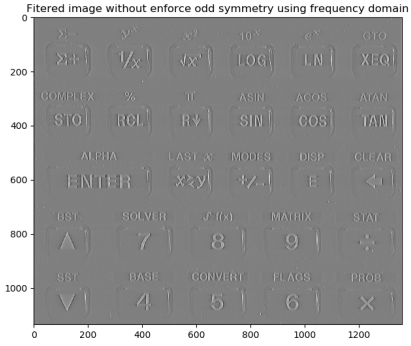
The result in (c) is the same as the result of space-domain filtering.

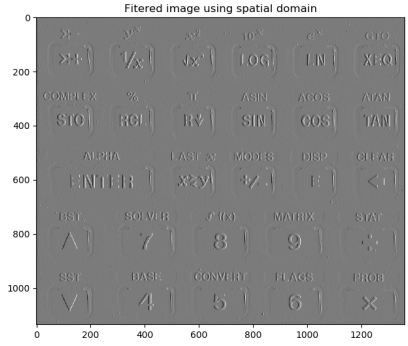


e. Show the result of frequency-domain filtering without enforcing odd symmetry on the kernel.

```
kernel_size = sobel.shape[0]
shifted_sobel = np.zeros([P, Q])
shifted_sobel[:kernel_size, :kernel_size] = sobel * shift_mask(kernel_size, kernel_size)
f_shifted_sobel = np.fft.fft2(shifted_sobel)
f_shifted_sobel_mag = np.abs(f_shifted_sobel)

f_filtered_img = f_padded_shifted_img * f_shifted_sobel_mag

img_filtered = np.fft.ifft2(f_filtered_img).real * shift_mask(P, Q)
img_filtered = img_filtered[:h, :w]
```

After using the sobel kernel without adding a leading row and columns of zeros, the filtered image is different from the result using spatial domain filtering.



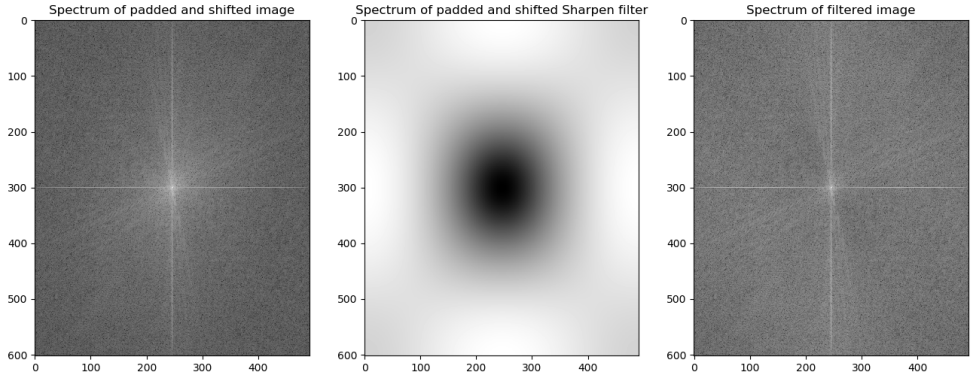


2. Problem 2

Design a frequency-domain filtering algorithm to enhance the following two test images. You should clearly describe your idea and verify how it works. The grading will be based on the quality of your description and results.

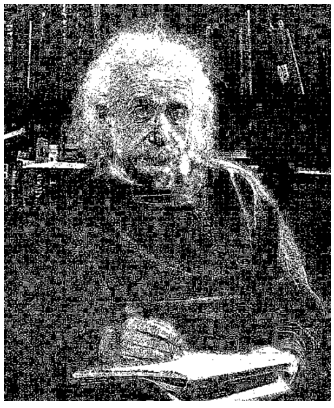
a. Einstein

Because the region of interest in this picture is high frequency part. So I use the sharpen filter to get the high frequency components. The way I filter image is the same as Problem1's procedures.

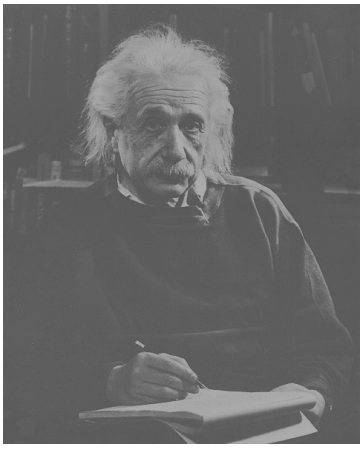
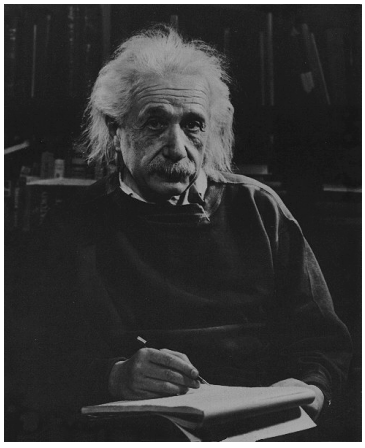


Then, I normalize the filtered image and remove the background noise

```
img_eq = np.clip(img_eq, 0, 255)
img_eq = img_eq/img_eq.max()
img_eq *= img.max()
img_eq[img_eq - img_eq.mean() < 0.15] = 0
cv2.imshow("edge_Einstein", img_eq)
```

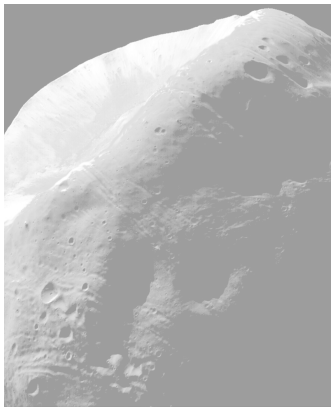


After getting the high frequency components, I iteratively add the original image back and do normalization with 10 times. In other words, I emphasized the detail part (high frequency) of the picture, So we can see that Einstein's hair, face and hands are much clearer than the original image.

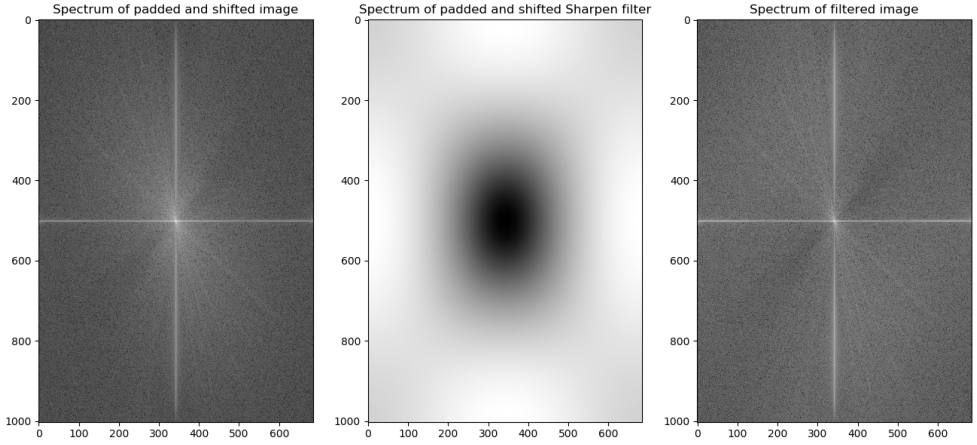



b. Phobos

Because the color of original image is very imbalanced, So I firstly do histogram equalization. After doing HE, the ignored detail in this picture are revealed.



The situation is likely to 2.(a), so I do the same procedures to extract the high frequency components.



Finally, I iteratively add the original image back and do normalization with 10 times. In other words, I emphasized the detail part (high frequency) of the picture.