CS-663 Assignment 5 Q5

Soham Naha (193079003)
Akshay Bajpai (193079002)
Mohit Agarwala (19307R004)

November 16, 2020

5

Read Section 1 of the paper 'An FFT-Based Technique for Translation, Rotation, and Scale-Invariant Image Registration' published in the IEEE Transactions on Image Processing in August 1996. A copy of this paper is available in the homework folder. Implement the technique in Equation 3 of the paper to align two images which are related to each other by a 2D in-plane translation. Test your implementation on images I and J as follows. I is a 300×300 image containing a 50×70 white rectangle (intensity 255) whose top-left corner lies at pixel $(50, 50)$. All other pixels of I have intensity 0. The image J is obtained from a translation of I by values $(t_x = -30, t_y = 70)$. Verify carefully that the predicted translation agrees with the ground truth translation values. Repeat the exercise if I and J were treated with iid Gaussian noise with mean 0 and standard deviation 20. In both cases, display the logarithm of the Fourier magnitude of the cross-power spectrum in Equation 3 of the paper. What is the time complexity of this procedure to predict translation if the images were of size $N \times N$? How does it compare with the time complexity of pixel-wise image comparison procedure for predicting the translation?
Also, briefly explain the approach for correcting for rotation between two images, as proposed in this paper in Section II. Write down an equation or two to illustrate your point. [8+7=15 points]

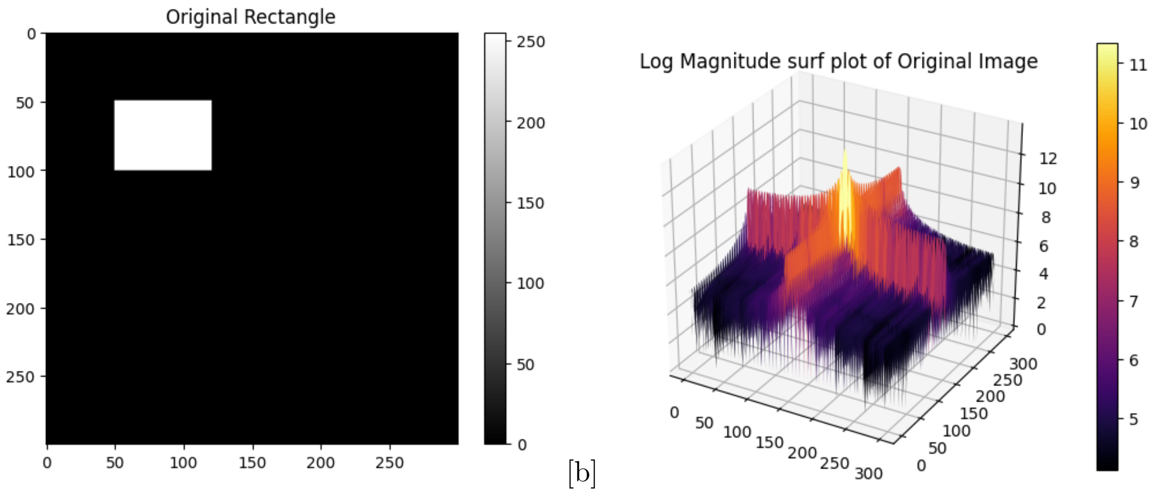
Solution

myPaperImplement.py

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.signal as sg
import cv2
from mpl_toolkits.mplot3d import axes3d

def create_rectangle(size, start , shape, type_, scale=1.0, verbose=True):
    """Creates a rectangular patch in an image
    :param size: the size of the image
    :param start: the starting position of the rectangle
    :param shape: the shape of the rectangular patch
    :param type_: two choices "Noisy" or "Original"
    :param scale: in case of "Noisy", the standard deviation of the noise
    :param verbose: if True plots the patch in the ../images/ folder
    :output I: the rectangular patch image
    """
    if type_=="Original":
        I = np.zeros((size, size))
    elif type_=="Noisy":
        I = np.random.normal(scale=scale,size=size*size).reshape((size,size))

    for i in range(start[0],start[0]+shape[0]+1):
        for j in range(start[1], start[1]+shape[1]+1):
                I[i,j] += 255

    if verbose:
```

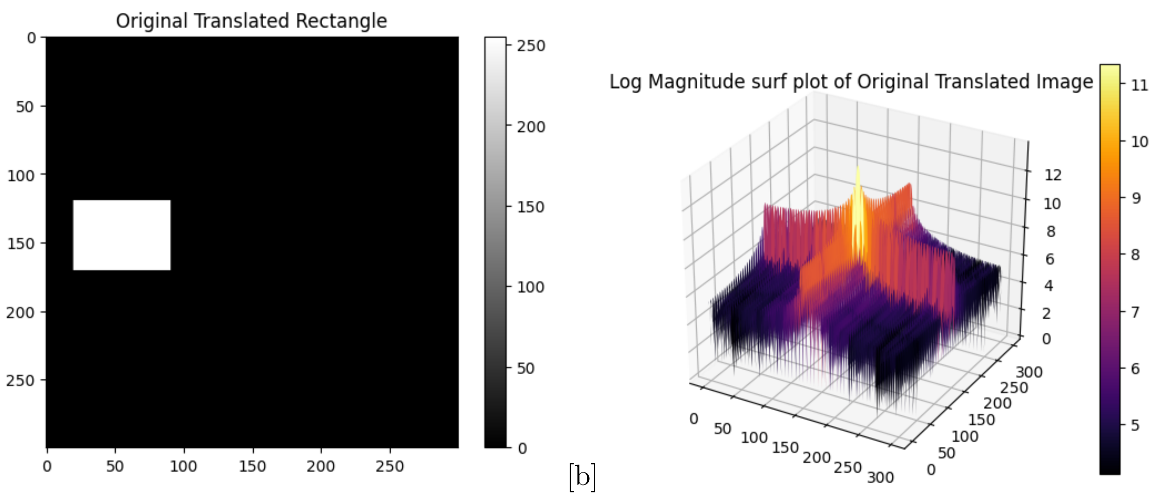
```
        plt.figure()
        plt.title(type_+" Rectangle")
        plt.set_cmap("gray")
        plt.imshow(normalize(I), cmap="gray")
        plt.colorbar()
        plt.savefig("../images/"+type_+"_Rectangle.png", bbox_inches="tight")
    return I

def spatial_translation(image, translation, type_, verbose=True):
    """Creates a spatially translated image of the rectangular patch
    :param image: the input image to perform translation
    :param translation: the translation co-ordinates
    :param type_: choose between "Noisy" and "Original"
    :param verbose: if True saves the plots
    :output translated: the translated image
    """
    translation_matrix = np.array([[1,0,translation[0]],[0,1,translation[1]]]).astype(np.float)
    translated = cv2.warpAffine(image, translation_matrix, image.shape )

    if verbose:
        plt.figure()
        plt.title(type_ + " Translated Rectangle")
        plt.set_cmap("gray")
        plt.imshow(translated, cmap="gray")
        plt.colorbar()
        plt.savefig("../images/" + type_+ "_TranslatedRectangle"+type_+".png",
                                bbox_inches="tight", cmap="gray")

    return translated

def calculate_Fourier(image):
    """Calculates the 2D-fourier transform of the input
    :param image: the 2D input image
    :output: 2D Fourier Transformed image
    """
    return np.fft.fft2(image)

def cross_power_spectrum(orig, translated, type_, verbose=True):
    """Calculates the Cross Power Spectrum of the two images and estimates the translation
    :param orig: the original image
    :param translated: the translated original image
    :param type_: "Noisy" or "Original"
    :param verbose: if True saves the Log-Magnitude of Cross Power Spectrum
    :output (t1,t0): the estimated translation
    """
    orig_fourier = calculate_Fourier(orig)
    trans_fourier = calculate_Fourier(translated)
    cross_power = np.fft.ifft2(orig_fourier*np.conj(trans_fourier))
    eps = 1e-15
    ir = np.abs(np.fft.ifft2((orig_fourier * trans_fourier.conjugate()) / (np.abs(orig_fourier) *
                                                            np.abs(trans_fourier)+eps)))

    if verbose:
        plt.figure()
        plt.imshow(np.log(1+ir),cmap="gray")
        plt.colorbar()
        plt.title(type_ +" Log Cross-Power Spectrum")
        plt.savefig("../images/LogCrossPowerSpectrum_"+type_ +".png",
                            bbox_inches="tight",cmap="gray")

    r,c = orig.shape
    t0, t1 = np.unravel_index(np.argmax(ir), orig.shape)
    if t0 >r//2:
        t0 -= r
    if t1>c//2:
```

```python
        t1 -= c
    return [t1, t0]

def normalize(image):
    """Min-Max normalization
    :param image : input image to normalize
    :output min-max scaled image
    """
    max_ = np.max(image)
    min_ = np.min(image)

    return ((image-min_)/(max_-min_))*255.0

def plot_log_magnitude(image,type_):
    r,c = image.shape
    x = np.array([i for i in range(r)])
    y = np.array([i for i in range(c)])
    X,Y = np.meshgrid(x,y)
    fft_image = np.fft.fftshift(np.fft.fft2(image))
    log_mag = np.log(1+np.abs(fft_image))

    fig = plt.figure()
    ax = plt.axes(projection ='3d')
    plt.set_cmap("inferno")
    surf = ax.plot_surface(X, Y, log_mag, cmap="inferno")
    fig.colorbar(surf, ax=ax)
    plt.title("Log Magnitude surf plot of "+type_ + " Image")
    plt.savefig("../images/LogMagSurfPlotof"+type_+"_image.png",
                bbox_inches="tight",cmap="inferno")


if __name__=="__main__":
    size = 300
    start = (50,50)
    shape = (50,70)
    std_dev = 20

    orig_rect = create_rectangle(size, start, shape, "Original")
    noisy_rect = normalize(create_rectangle(size, start, shape, "Noisy", scale=20.0))
    plot_log_magnitude(orig_rect,"Original")
    plot_log_magnitude(noisy_rect,"Noisy")

    orig_trans = spatial_translation(orig_rect, (-30, 70),"Original")
    noisy_trans = spatial_translation(noisy_rect, (-30, 70), "Noisy")
    plot_log_magnitude(orig_trans,"Original Translated")
    plot_log_magnitude(noisy_trans,"Noisy Translated")

    t0,t1 = cross_power_spectrum(orig_rect, orig_trans, "Original")
    print("tx: {}, ty: {} for Original Rectangle.".format(t0, t1))

    t0,t1 = cross_power_spectrum(noisy_rect, noisy_trans, "Noisy")
    print("tx: {}, ty: {} for Noisy Rectangle.".format(t0, t1))
```



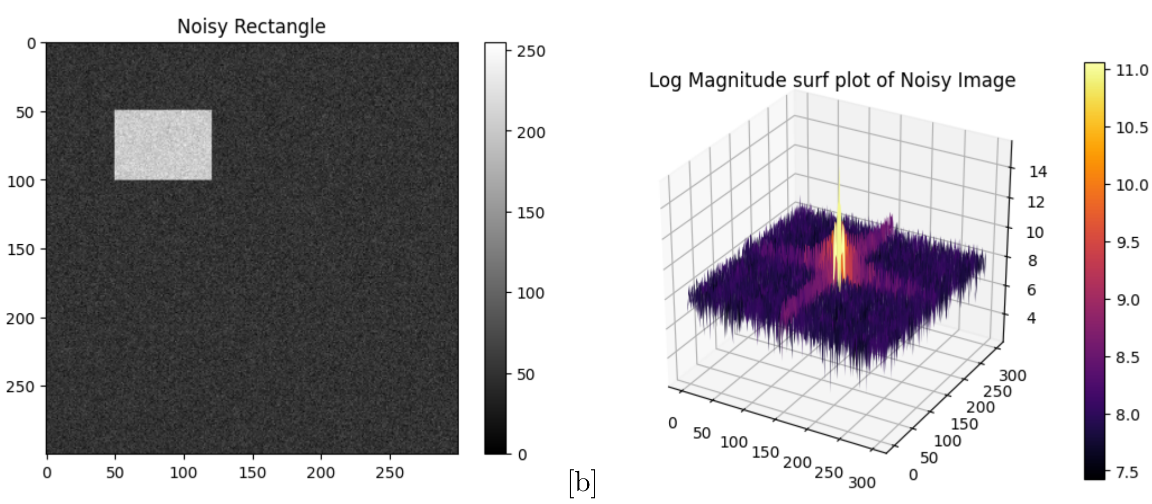
[a] [b]

Figure 1: (a) Original Rectangular patch image (b) Log-Magnitude Surface Plot of (a)



[a] [b]

Figure 2: (a) Translated Original image (b) Log-Magnitude Surface Plot of (a)



[a] [b]

Figure 3: (a) Noisy Rectangular patch image (b) Log-Magnitude Surface Plot of (a)

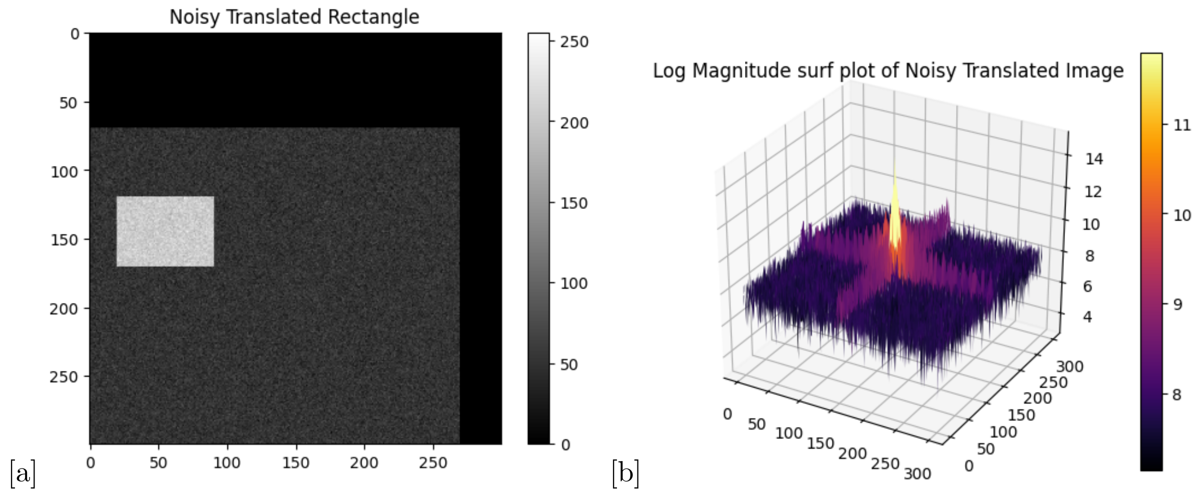


Figure 4: (a) Translated Noisy image (b) Log-Magnitude Surface Plot of (a)

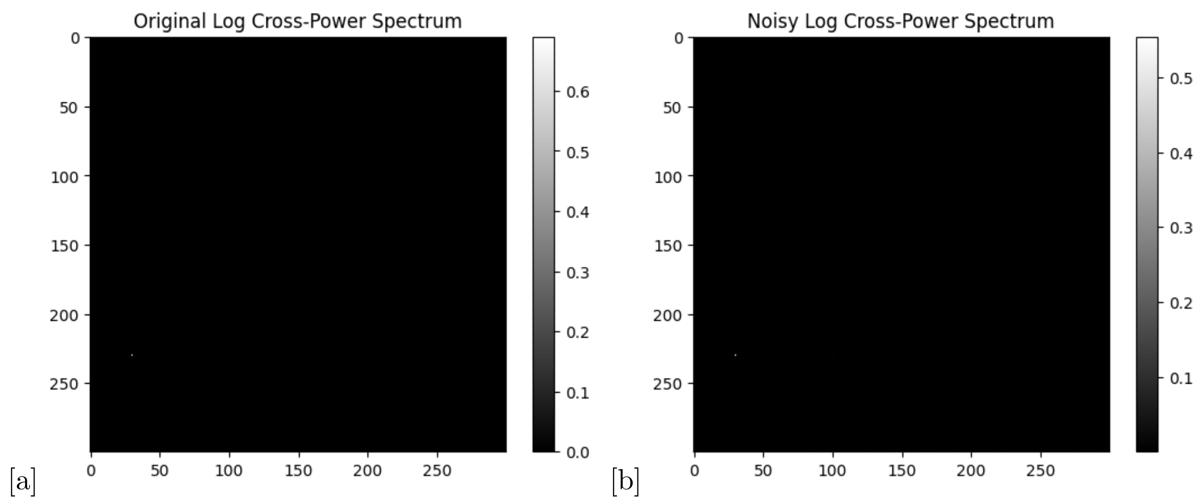


Figure 5: (a) Log-Magnitude Cross Power Spectrum of Translated Original Image (b) Log-Magnitude Cross Power Spectrum of Translated Noisy Image

Discussions :

- For the plots above Image J = Noisy Image and Image I = Original Image
- For both the images (with and without noise, the translations are predicted as follows:

```
tx: 30, ty: -70 for Original Rectangle(I).
tx: 30, ty: -70 for Noisy Rectangle(J).
```

- So the noisy image(J) has to be translated by ($t_x = 30andt_y = -70$) to get back to the original image(I).
- The time complexity of the approach followed in the mentioned paper is $O(N\log(N))$ where NxN is the dimensions of the image. The algorithm involves calculation of Fourier Transform of order $O(N\log(N))$, conjugate of one fft-image $O(N)$ and point-wise multiplication and division $O(1)$.
- he time complexity of pixel-wise image comparison procedure for predicting the translation is $O(N^2)$.

Discussion of Section II of the paper.

- (Considering only normal rotation without considering translation or scaling along with rotation)
  If $f_2(x,y)$ is a rotated version of $f_1(x,y)$ [with a rotation of θ_0], doing a Fourier Transform in the Cartesian coordinates would yield $F_2(u,v) = F_1(u\cos(\theta_0) + v\sin(\theta_0), -u\sin(\theta_0) + v\cos(\theta_0))$. Clearly, their magnitudes are the same. So, we can use the same concept of cross-power spectrum as before by converting the rotation by θ_0 into a translation. This can be achieved by converting the images into polar coordinates taking their Fourier Transform:

$$f_2(r,\theta) = f_1(r, \theta - \theta_0)$$

$$F_2(m,n) = exp(-2\pi j(n\theta_0)) * F_1(m,n)$$

  Thus, cross-power spectrum of $F_1(m,n)$ $F_2(m,n)$ would yield $\exp(2\pi j(n\theta_0))$, using which we can calculate the rotation. Any translation in x y would lead to a change in r by r_0, such that the cross power spectrum would yield $exp(2\pi j(mr_0 + n\theta_0))$.
  Hence, displacement rotation can be figured out. The exact (x, y) translations can be figured out using the original cross-power spectrum in the Cartesian coordinates.

- In case of translation + rotation,

$$f_2(r,\theta) = f_1(\rho - \rho_0, \theta - \theta_0)$$

  , i.e the translation will lead to change in ρ by ρ_0.
  We will get $exp(2\pi j(u\rho_0 + v\theta_0)$ after applying the cross power spectrum.
  Thus, we can get both translation and rotation values. The exact (x,y) coordinates can be deduced from the polar coordinates using the corresponding relations.

- In case of scale change (without rotation and translation), if f_1 is a scaled replica of f_2 with scale factors (a,b) then,

$$F_2(x,y) = \frac{F_1(\frac{x}{a}, \frac{y}{b})}{|ab|}$$

$$or, F_2(\log x, \log y) = F_2(\log x - \log a, \log y - \log b)$$

  , where we can find $\log(a)$ and $\log(b)$ using the phase correlation technique mentioned above.

- If f_2 is translated, rotated, and scaled replica of f_1, then we have in polar coordinates :

$$M_2(\log(\rho), \theta) = M_1(\log(\rho) - \log(a), \theta\theta_0)$$

  where θ_0 is the rotation and a is the scale magnitude. Thus, we can find the rotation and scale using phase correlation technique. The translation is inside the ρ term, which can be found out by applying phase correlation technique on the image which we have scaled by a and rotated by θ_0, i.e removed the effects of scaling and rotating.