

A report on

FOURIER TRANSFORM MODEL OF IMAGE FORMATION

by: Christian Magsigay

Submitted to : Dr. Maricor Soriano

In partial fulfilment of the requirements for Applied Physics 157



“ Objectives:

- to perform discrete Fourier Transform on synthetic images acting as various apertures to simulate Fraunhofer diffraction patterns.
- to perform convolution by Fourier Transform using synthetic images as apertures with varying diameters.
- to use correlation to match a template with an image.

Results Analysis

1

Familiarization with Discrete FT

We were tasked to use discrete Fourier Transform on synthetic images to simulate Fraunhofer diffraction patterns from these apertures: *Circular aperture, sine wave along x-axis, square aperture, double slit, annular ring, and circular aperture with gaussian transmittance.*



Browser Window

http://www.url.com

Generating some synthetic images

For images not generated in the first activity

```
import numpy as np
import matplotlib.pyplot as plt
import skimage.io

#circular aperture
plt.rcParams['figure.dpi'] = 200
N = 256
x = np.linspace(-2, 2, N)
y = x
X, Y = np.meshgrid(x,y)
R = np.sqrt(X**2 + Y**2)
A = np.zeros(np.shape(R))
A[np.where(R<0.25)]=1.0

#double slit
double_slit=np.zeros(np.shape(R))
for i in range(len(R)):
    for j in range(len(R)):
        if i<len(R)/8*3 and i<len(R)*5/8 and j>(len(R)/8*3) and j<(len(R)*5/8):
            double_slit[i,j]=1.0 #generates the 1x1 white square
        if j>(len(R)*0.4) and j<(len(R)*0.6):
            double_slit[i,j]=0.0 #creates the double slit

#Grating
x = np.linspace(-1,1,N)
y = x
X, Y = np.meshgrid(x,y)
def f(x, y):
    return np.sin(2*k*np.pi*x)
Z = f(x, 5)
grating = np.zeros(np.shape(Z))
grating[np.where(Z>0)] = 1.0
```

Browser Window

http://www.url.com

Generating diffraction patterns by Fourier transform

```
#Transforms the image
def transform(Img, name):
    #Fourier Transform
    FA = np.fft.fft2(Img)
    FAshifted = np.fft.fftshift(FA)
    #plot
    fig, axes = plt.subplots(1, 3, figsize=(11,8));
    axes[0].imshow(Img, cmap = "gray")
    axes[0].set_title(name)
    axes[1].imshow(abs(FA), cmap = "gray")
    axes[1].set_title("Fast Fourier Transform")
    axes[2].imshow(abs(FAshifted), cmap = "hot")
    axes[2].set_title("FFT Shifted")
    axes[0].axis('off')
    axes[1].axis('off')
    axes[2].axis('off')
    plt.savefig(name+".png",bbox_inches = 'tight',
               pad_inches = 0)
#Circular aperture:
transform(A, "Circular aperture")
#Grating:
synth_img=grating
transform(synth_img,"Grating")
#Double slit
synth_img=double_slit
transform(synth_img,"Double slit")
#Square aperture
synth_img=skimage.color.rgb2gray(skimage.io.imread("square.png"))
transform(synth_img,"Square aperture")
#Annular aperture
synth_img=skimage.color.rgb2gray(skimage.io.imread("annulus.png"))
transform(synth_img,"Annular aperture")
#Circular aperture (Gaussian)
synth_img=skimage.color.rgb2gray(skimage.io.imread("Gaussian_2D.png"))
transform(synth_img,"Circular aperture (Gaussian)")
```

Figure 1. Code for generating diffraction patterns by transforming various synthetic images.

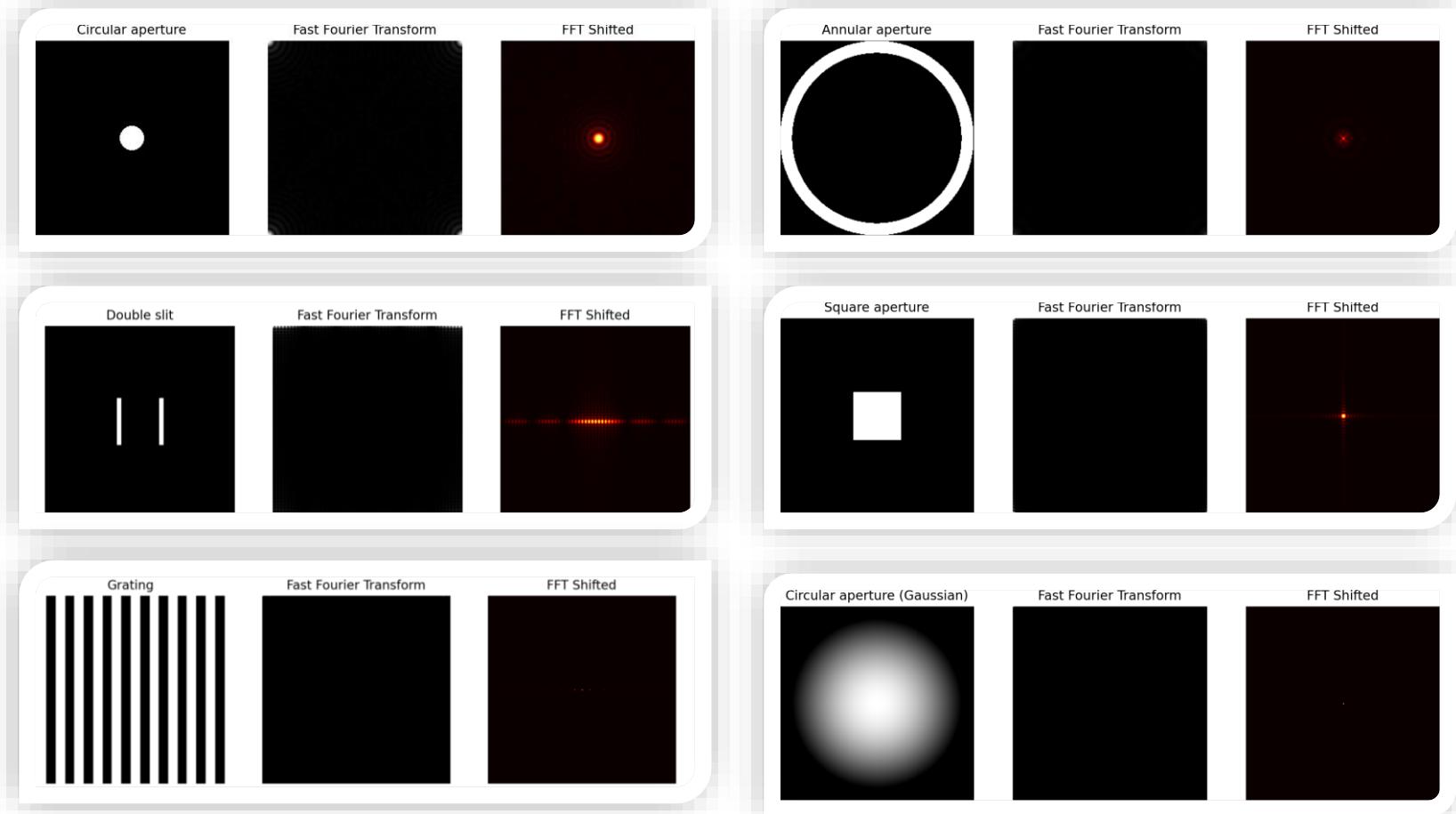
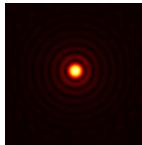


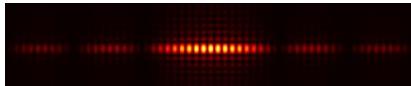
Figure 2. Synthetic images acting as apertures and their corresponding fast Fourier transformation and diffraction pattern.

Diffraction Patterns

Circular aperture:



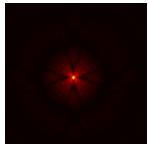
Double slit:



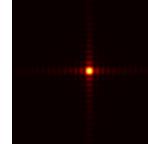
Grating:



Annular aperture:



Square aperture:



Circular aperture (Gaussian):



Initially, various synthetic images from the previous activity were imported. Figure 2 presents the emerging patterns after performing Fast Fourier Transform (FFT) and FFTshift on the synthetic images. Also, figure 3 zooms in on the emerging Fraunhofer diffraction patterns.

Fourier transforms use the sum of cosines and sines in representing a certain signal. In this activity, we extended it to two dimensions to mimic the lens of a camera as a Fourier Transformer represented by the equation:

$$F(f_x, f_y) = \iint f(x, y) \exp(-i2\pi(f_x x + f_y y)) dx dy$$

Another interesting property of FFT is its 2D symmetry. If you zoom in on figure 2, you can see symmetric patterns at each corner of the images. The orientation of these patterns was then corrected by the FFTshift function, which then produced the diffractions patterns in figure 3.

All resulting patterns were the expected outcome in a real-life experiment. The circular aperture produced an airy disk. The double-slit created an interference pattern with the brightest group of spots at the center. The grating also made an interference pattern, but each bright fringe is dominated only by a single spot; the center spot is also the brightest. The annular aperture produced a bright spot with visible scattering around it. The square aperture produced a crosshair pattern. Lastly, The circular aperture with Gaussian transmittance had a single tiny bright spot.

Figure 3. Zoomed in diffraction patterns of various synthetic images

Reflection

The resulting diffraction patterns in this activity are the expected patterns if we were to use the apertures in a real-life experiment, so our results are reasonable.

The activity was a lot of fun. I was amazed that we could simulate various diffraction patterns by using Fourier transforms. Now, I can't even begin to imagine what other things this powerful mathematical tool can do.

While doing the activities, I was curious about certain things. So, I also tried some interesting cases, and my findings are presented in the next slide.

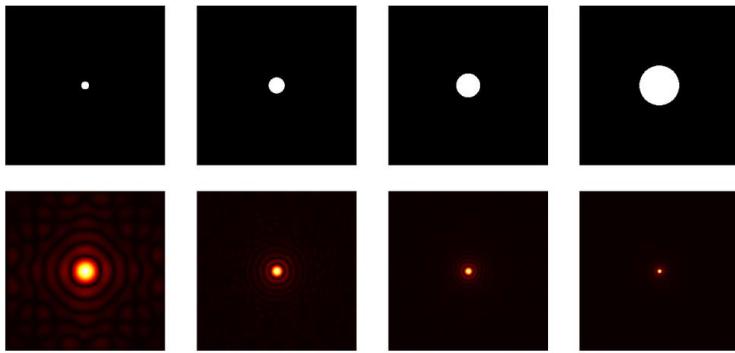


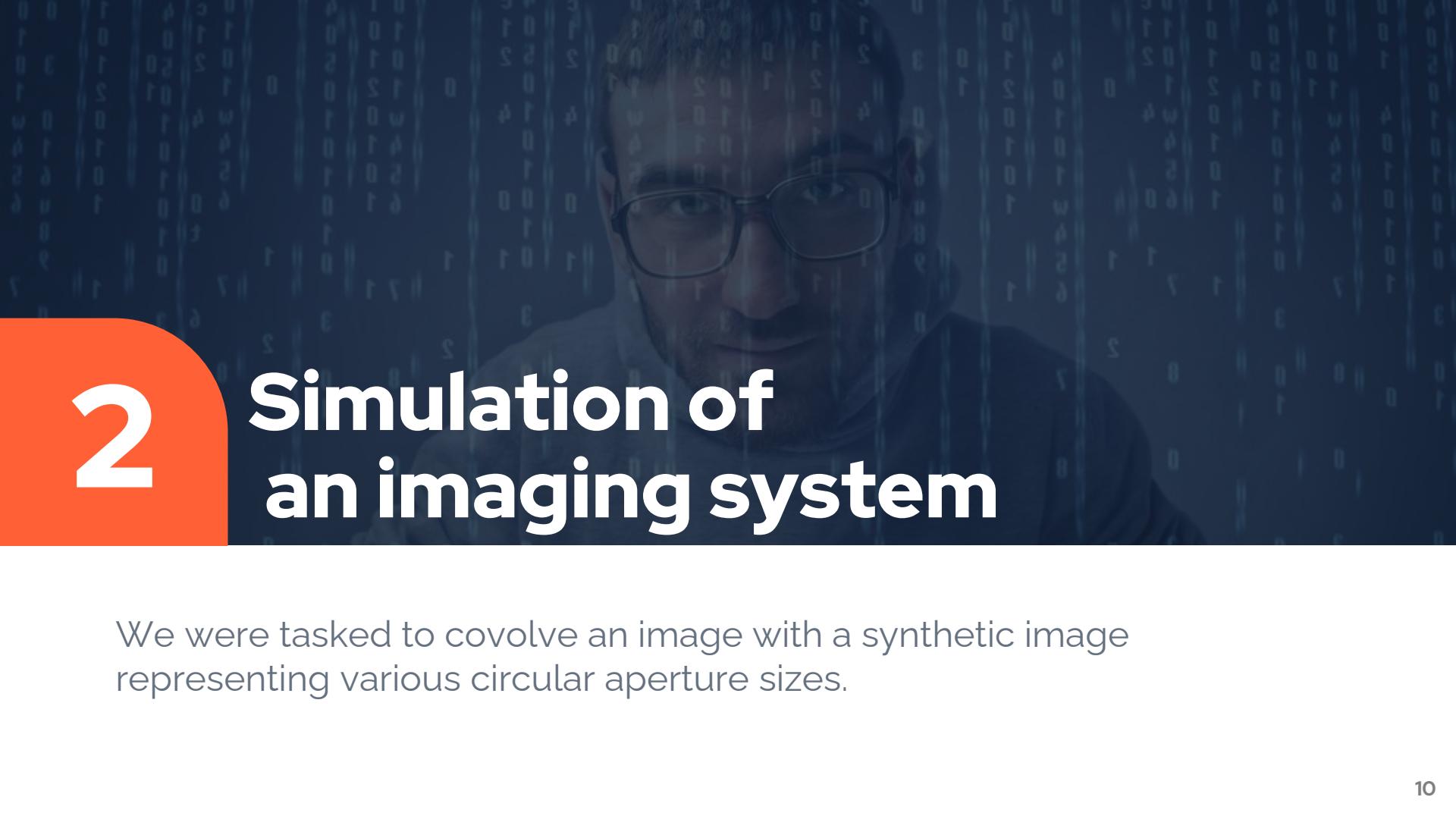
Figure 4. Diffraction patterns for different aperture sizes.



Figure 5. Diffraction pattern from an uncentered aperture

It was observed that as you increase the size of the aperture, the emerging pattern will, in turn, decrease in size.

I was also surprised to find out that the Fourier transform does not depend on the position of the aperture as seen in figure 5. The resulting pattern is still identical to the case where the aperture was at the center.

A person wearing glasses and a hoodie, standing in front of a dark background with binary code falling down like rain.

2

Simulation of an imaging system

We were tasked to convolve an image with a synthetic image representing various circular aperture sizes.



Generating circular apertures with various sizes

```
def circ(percent_D):
    N = 256
    x = np.linspace(-2, 2, N)
    y = x
    X, Y = np.meshgrid(x, y)
    R = np.sqrt(X**2 + Y**2)
    A = np.zeros(np.shape(R))
    A[np.where(R < (percent_D * 2))] = 1.0
    return A

D_25P,D_50P,D_75P,D_100P=circ(0.25),circ(0.5),\circic(0.75),circ(1.0)

#Display figures
fig, axes = plt.subplots(1, 4, figsize=(11,8));
axes[0].imshow(D_25P, cmap = "gray")
axes[0].set_title("25% diameter")
axes[1].imshow(D_50P, cmap = "gray")
axes[1].set_title("50% diameter")
axes[2].imshow(D_75P, cmap = "gray")
axes[2].set_title("75% diameter")
axes[3].imshow(D_100P, cmap = "gray")
axes[3].set_title("100% diameter")
axes[0].axis('off')
axes[1].axis('off')
axes[2].axis('off')
axes[3].axis('off')
plt.savefig("circs.png",bbox_inches = 'tight',pad_inches = 0)
plt.show()
```

Convolution

```
import skimage.io
import matplotlib.pyplot as plt
import numpy as np

Img = skimage.io.imread("157images/NIP.bmp")

def convolve(image,aperture):
    A = aperture
    Ashift = np.fft.fftshift(A)
    FImg = np.fft.fft2(image[:, :, 1])
    H = Ashift[:, :, 1]*FImg
    h = np.fft.ifft2(H)
    return abs(h)

A,B,C,D=convolve(Img,D_25P),convolve(Img,D_50P),\convolve(Img,D_75P),convolve(Img,D_100P)

#Display figures
fig, axes = plt.subplots(1, 4, figsize=(11,8));
axes[0].imshow(A, cmap = "gray")
axes[0].set_title("25% diameter")
axes[1].imshow(B, cmap = "gray")
axes[1].set_title("50% diameter")
axes[2].imshow(C, cmap = "gray")
axes[2].set_title("75% diameter")
axes[3].imshow(D, cmap = "gray")
axes[3].set_title("100% diameter")
axes[0].axis('off')
axes[1].axis('off')
axes[2].axis('off')
axes[3].axis('off')
plt.savefig("convolved.png",bbox_inches = 'tight',pad_inches = 0)
plt.show()
```

Figure 6. Codes for generating the synthetic images and the convolution process of an image and a synthetic image acting as an aperture.

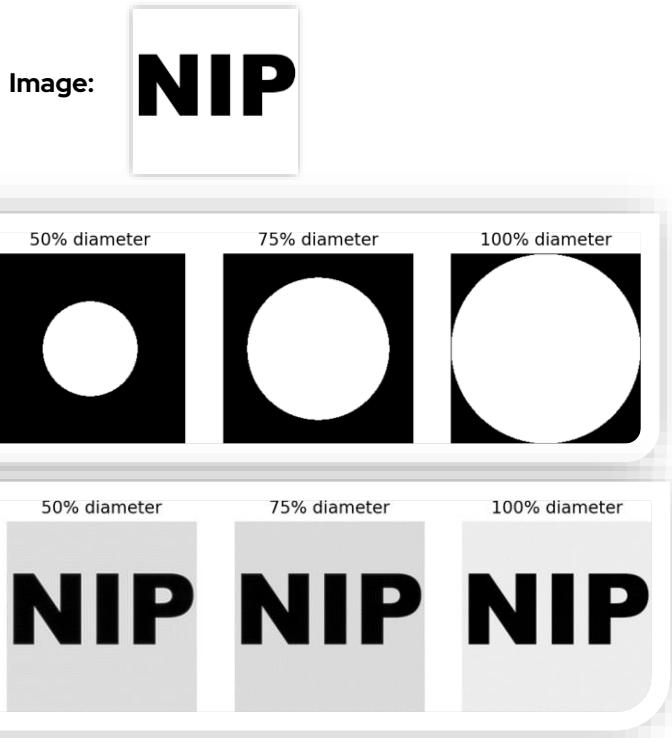


Figure 7. circular apertures with various diameters and their resulting convolution with an image.

Initially, synthetic images of a circular aperture with various sizes were generated. These were then convolved with an image using a series of Fourier transforms, and figure 7 presents the results.

In the process of convolution, a function is 'smeared' to another function by linear transformations. The convolution between two 2-D functions f and g is given by:

$$h(x, y) = f \otimes g = \int \int f(x', y') g(x - x', y - y') dx' dy'.$$

The synthetic images acted like camera lenses. A smaller aperture meant that it could only gather a small bundle of rays from an object; therefore, the resulting image has an evident loss of quality; however, as the aperture increases, the quality of the resulting image also increases. The stated phenomenon is evident in figure 7, therefore verifying our result.

Reflection

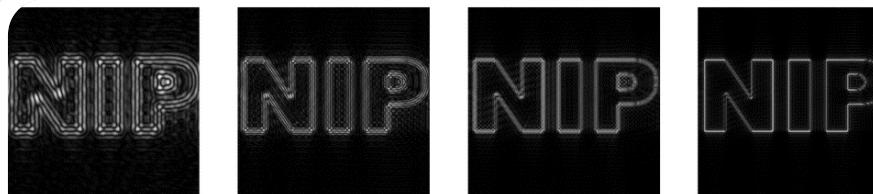
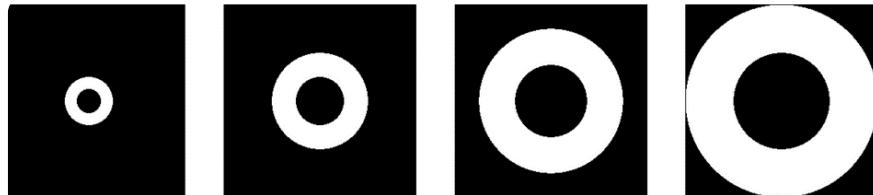
Convolution really is a powerful tool. This concept of generating a new array by relating two distinct arrays of the same dimensionality can be very useful not only in image processing but I reckon it could also be useful in data analysis.

I had encountered the use of this technique before in Applied Physics 155, where we used it to enhance a blurred image. It then revealed details we could not see before; it was astounding.

I was also curious what would happen if I used a different aperture. My results are presented in the next slide.

Image:

NIP



By using an annular aperture with various diameters, the resulting images only captured the outline of the original image. Nonetheless, the same conclusion can be drawn that when the lens is bigger the higher the quality of the resulting image, where less scattering is observed.

Figure 8. Annular apertures with various diameters and their resulting convolution with an image.

3

Template matching using correlation

We were tasked to match a template with an image.

CODE

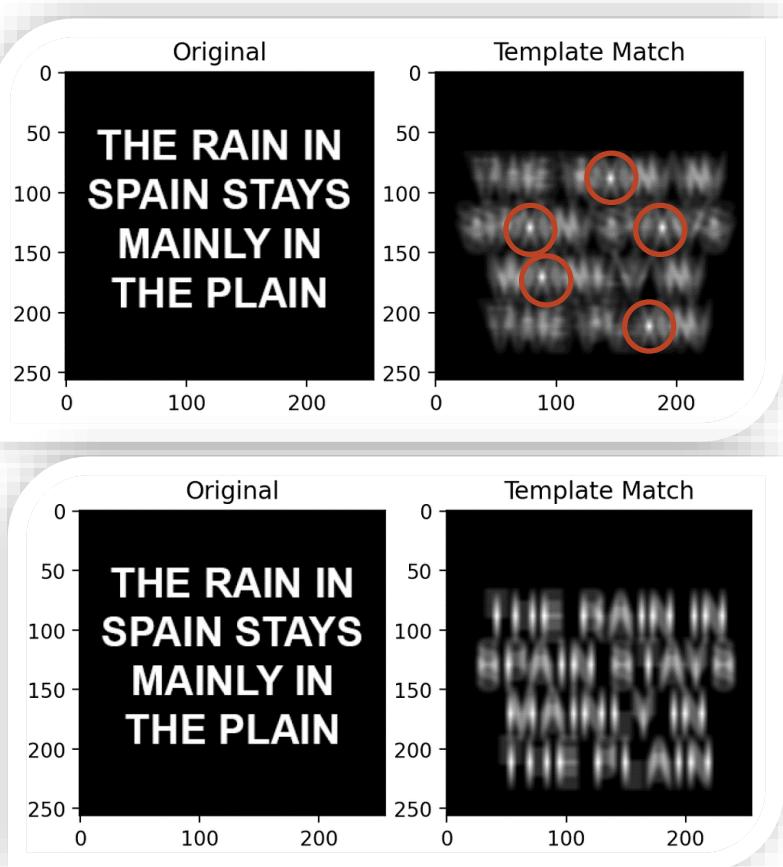


The image shows a screenshot of a web browser window titled "Browser Window". The address bar contains "http://www.url.com". The main content area displays a Python script titled "Template Matching algorithm". The code uses the skimage library to read images A and I, and then performs template matching using the FFT. It includes subplots for the original images and the template match results.

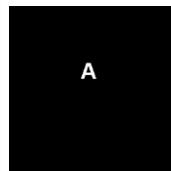
```
A = skimage.color.rgb2gray(skimage.io.imread("157images/A.bmp"))
#template image
I = skimage.color.rgb2gray(skimage.io.imread("157images/I.bmp"))
#template image
Img = skimage.color.rgb2gray(skimage.io.imread("157images/the rain.bmp"))
#phrase image
def template_match(image,template):
    FImg = np.fft.fft2(image)
    AFFT = np.fft.fft2(template)
    P = np.conjugate(FImg)*AFFT
    smallp = np.fft.ifft2(P)

    fig, (ax1, ax2) = plt.subplots(1, 2)
    ax1.imshow(Img, cmap="gray")
    ax1.set_title("Original")
    ax2.imshow(np.fft.fftshift(abs(smallp)))
    ax2.set_title("Template Match")
    plt.show()
template_match(Img,A) #template match "A"
template_match(Img,I) #template match "I"
```

Figure 9. Code for template matching



Template image:



The template image which contains the letter to be matched and the phrase image had undergone a correlation process. A series of Fourier transforms were used to relate the two images.

Template image:



The resulting image, as seen in figure 10, shows evident bright spots at locations where the letter 'A's are found in the phrase image. However, the letter 'I' seems to be not distinct enough to have a reliable match, since the correlation process also highlighted other letters that have vertical lines in them. Therefore, this algorithm works best for unique template images.

Figure 10. Results generated by the template matching algorithm

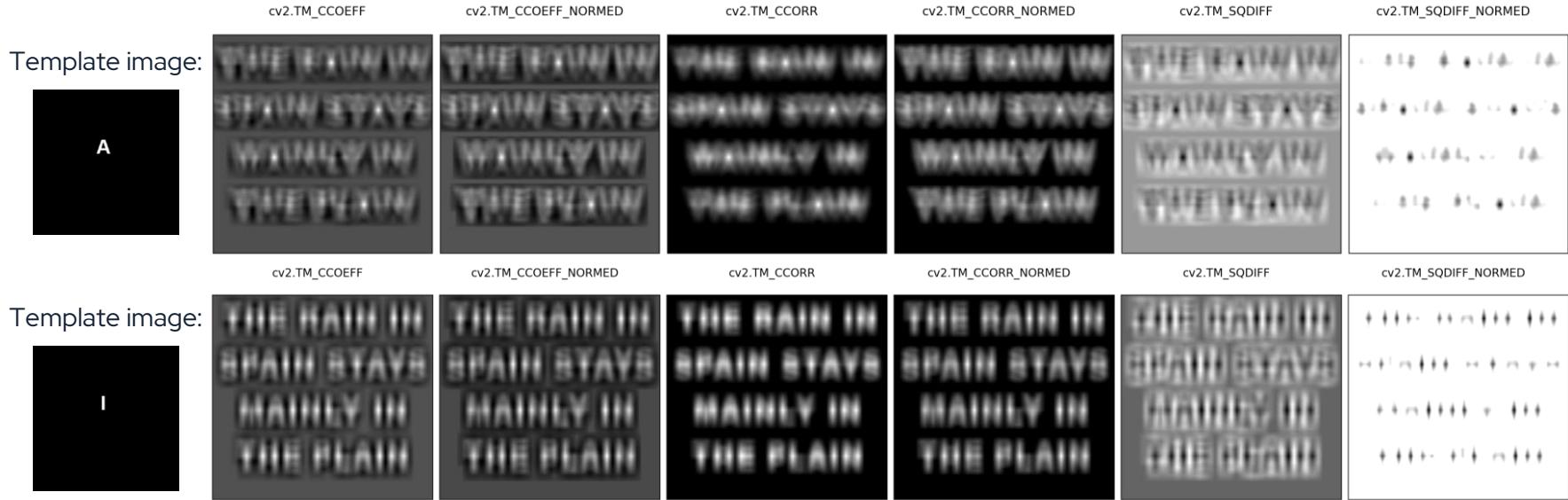


Figure 10. Output images of various template-matching algorithms using the cv2 package [1]

To verify the results of our template-matching algorithm, I have also conducted various template-matching processes using an OpenCV package. As you can see in figure 10, The results are fairly similar to our results. They also share the same limitation where they require a more distinct template image to produce more reliable results. Therefore, the results of our algorithm is valid.

Reference:

[1] "Template Matching – OpenCV-Python Tutorials 1 documentation." 29 Jan. 2021, opencv-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_template_matching/py_template_matching.html.

Reflection



Template matching is one cool algorithm. I am sure this can be extended to colored images or maybe even videos. I feel that this concept or something similar is used by video editing software to track objects in our videos. I also think the concept can be used in finding certain patterns in a huge array of data.

At this point, I have learned to appreciate the application of Fourier transform, which was responsible for making these ingenious algorithms possible. Admittedly, I have grown very fond of the equation.

120/100

I would give myself at least a 120 because I understood all the concepts and were able to satisfy all the needed objectives of this activity. I also went beyond on certain topics and have pointed out their limitations