FOURIER TRANSFORM PROPERTIES AND APPLICATIONS

Mary June A. Ricaña

SN: 2019-01551

Section: WFU-FX

Table of Contents

Side note:

All of my files may be found in this drive.

Introduction

The Fourier transform (FT) is a linear transform that takes a signal and converts with the help of sinusoidal basis functions. It converts X-dimension signals into 1/X. The FT of an image g(x,y) is given by the spatial frequencies f_x and f_y along the x and y axes:

$$G(f_x, f_y) = \iint g(x,y)\, exp\left(-i2\pi\left(f_x + f_y\right)\right) dxdy$$

In MATLAB, we can apply a Fourier transform using the functions fft( ) for 1D signals and fft2( ) for 2D signals. This can be utilized in different imaging and simulation techniques, which we will be exploring for this activity.

Objectives

The following are the goals of this activity:

- Demonstrate the rotation property of the Fourier transform
- Demonstrate the anamorphic property of the Fourier transform
- Remove image lines and patterns using convolution
- Simulate an imaging device using convolution
- Perform template matching via correlation
- Perform edge detection using convolution

Rotation Property of the Fourier Transform

For a 2D signal, the Fourier transform exhibits a rotational property, which translates to a rotation of the images as well. Suppose we have a constant image. Its FT will be given by $G(f_x, f_y) = \delta(f_x, f_y)$. For a sinusoid along the one direction,

$$g(x, y) = Asin(2\pi f_0 x)$$

and an FT of

$$G(f_x, f_y) = \frac{A}{2}\,\delta(f_x - f_0)\delta(f_y) + \frac{A}{2}\,\delta(f + f_0)\delta(f_y)$$

In this part, we will generate sinusoids of different frequencies and biases, rotate them, and find their Fourier transforms. We will also see what happens to the FT if we combine multiple sinusoids of different patterns together.

Sinusoids of different frequencies

From the first activity, we were tasked to generate a synthetic image of a sinusoid along the x-direction, with amplitude ϵ [0,1] and frequency at 4 cycles/cm. Here, we recreate the same lines of codes and add an fft2( ) and fftshift( ) functions to see what the Fourier transforms of the sinusoid looks like.

```
nx = 200; ny = 200;
x = linspace(-1,1,nx);
y = linspace(-1,1,ny);
[X,Y] = meshgrid(x,y);
f = 4 %frequency - you may change this later.
Z = sin(2*pi*f*X);
mesh(Z);
% Take the FT of the sinusoid
FZ = fft2(Z);
FZmod = fftshift(abs(FZ));
figure(2); mesh(FZmod);
figure(3); imshow(Z);
figure(4); imshow(FZmod);
```

Notice that the FT of our sinusoid looks like two dots, which resembles the Dirac deltas FT of the sinusoid. The Dirac delta function is zero almost everywhere, and the orthogonality of sinusoids can be expressed in terms of the Dirac delta function [2]. Let's see how the FT will be affected if we change the frequency of the sinusoid.

Figure 1. Sinusoid with frequency 4 (left) and its Fourier transform (left)

Sinusoids of different frequencies

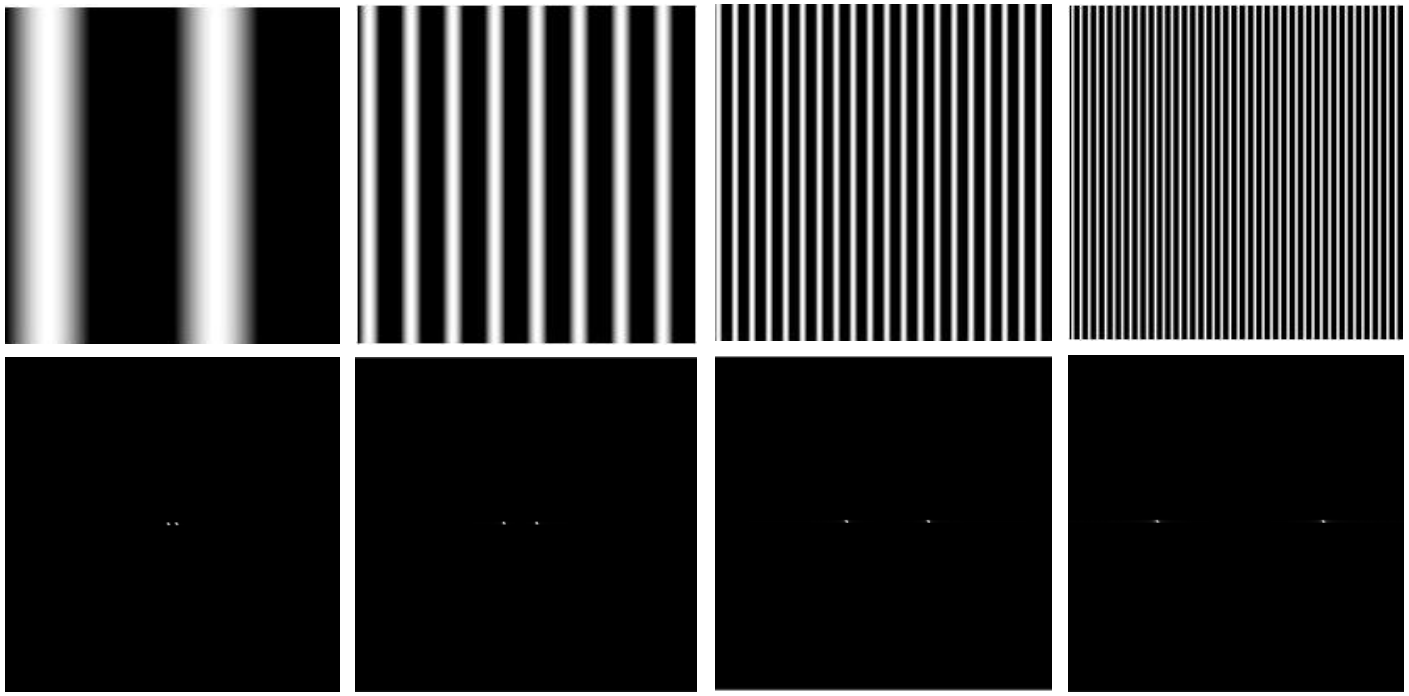
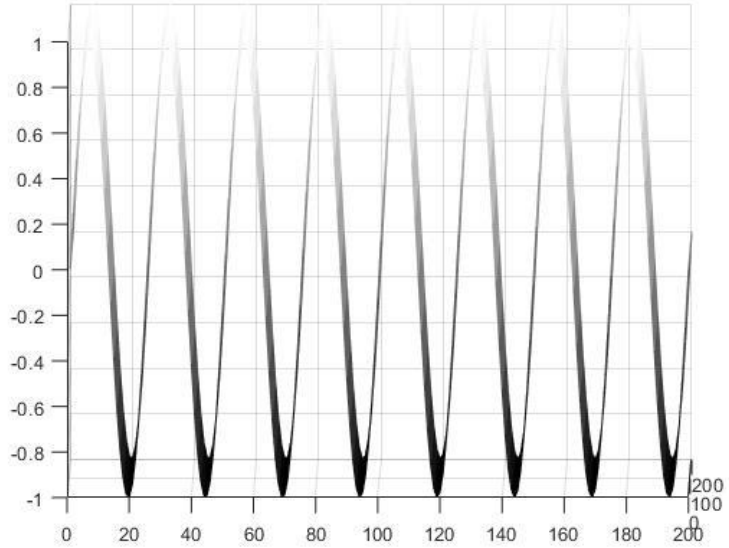


Figure 2. Sinusoid with frequencies 1, 4, 10, and 20 (left to right) and their corresponding Fourier transforms.

Notice that as the frequency increases, the separation distance between the two dots increases as well. This is because the FT of a sinusoid of frequency f is two dotes located at $\pm f$, which means that when the frequency is 20, the separation distance between the dots would be 40, since they will be locates 20 units away from the x axis (center).

Biased sinusoids

Now, notice that when we plot the sinusoid, the amplitude extends up to -1. However, digital images do not have negative values. Let's see how this affects the Fourier transform. Here, instead of generating the sinusoid via MATLAB for the FT, we save it as an image and use imread( ) and then find the FT.



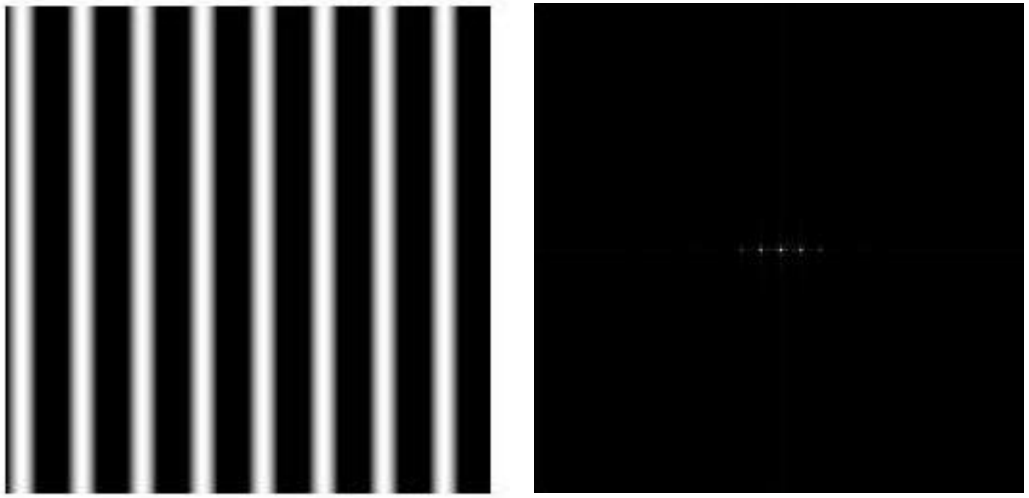


Figure 3. Digital image of a sinusoid with frequency 4 (left) and its Fourier transform (left)

We see that instead of the two dots that we have obtained for the sinusoid at freq = 4, we get five dots with the middle one being the brightest and the farthest from the middle as the faintest. To fix this, we must add a bias to our sinusoid so that there will be no negative values.

Biased sinusoids

After adding a bias of 1 in the y-axis, we now shifted the sinusoid upward so that there will be no negative values. In doing so, we generated a digitally accurate image of the sinusoid. After we got its FT, we see that we obtain the correct FT, which are the two Dirac deltas.

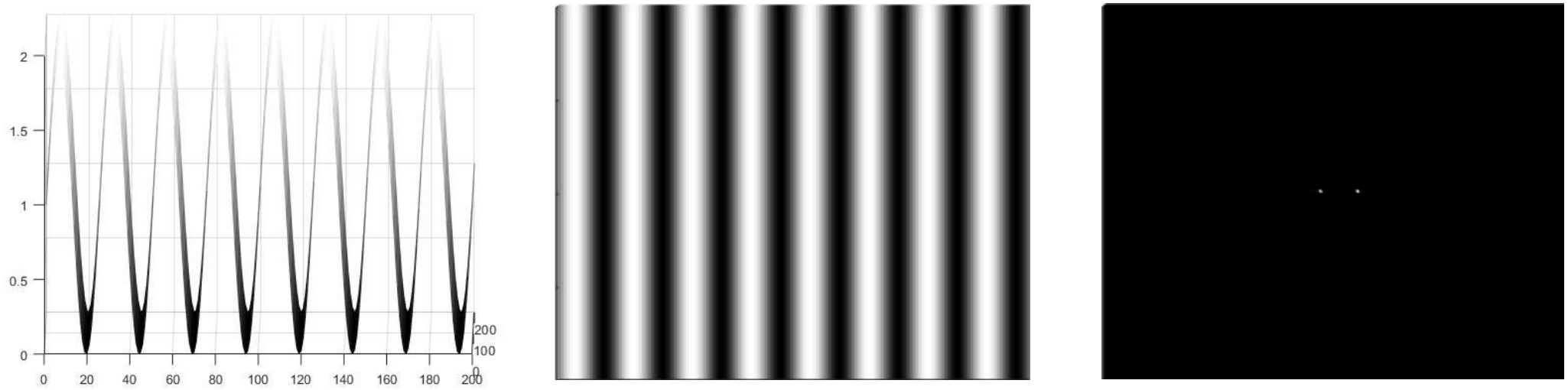


Figure 4. LEFT TO RIGHT: Graph of a sinusoid with a bias of +1 in the y axis, the generated biased sinusoid, and its FT.

Rotated Sinusoids

Now, we try to see what would happen to the FT if we rotate the sinusoid. In our code, we add a theta variable so that we could rotate the object. Then, we take its FT to see how the rotation affected it.

```
nx = 200; ny = 200;
x = linspace(-1,1,nx);
y = linspace(-1,1,ny);
[X,Y] = meshgrid(x,y);
f = 4
% Rotate the sinusoid to an angle
theta = 0;
Z = sin(2*pi*f*(Y*sin(theta) + X*cos(theta)));
mesh(Z);
FZ = fft2(Z);
FZmod = fftshift(abs(FZ));
figure(2); mesh(FZmod); colormap("gray");
figure(3); imshow(Z);
```

Here is a sinusoid rotated at 30 degrees. Notice that we see diagonal strips instead of the vertical strips we had earlier. Now, taking the FT, we still see the same two dots, but now, it is also tilted at an angle. This is expected given that the original image is rotated.

Figure 5. Sinusoid rotated at 30 degrees (right) and its Fourier transform (left)

Rotated Sinusoids

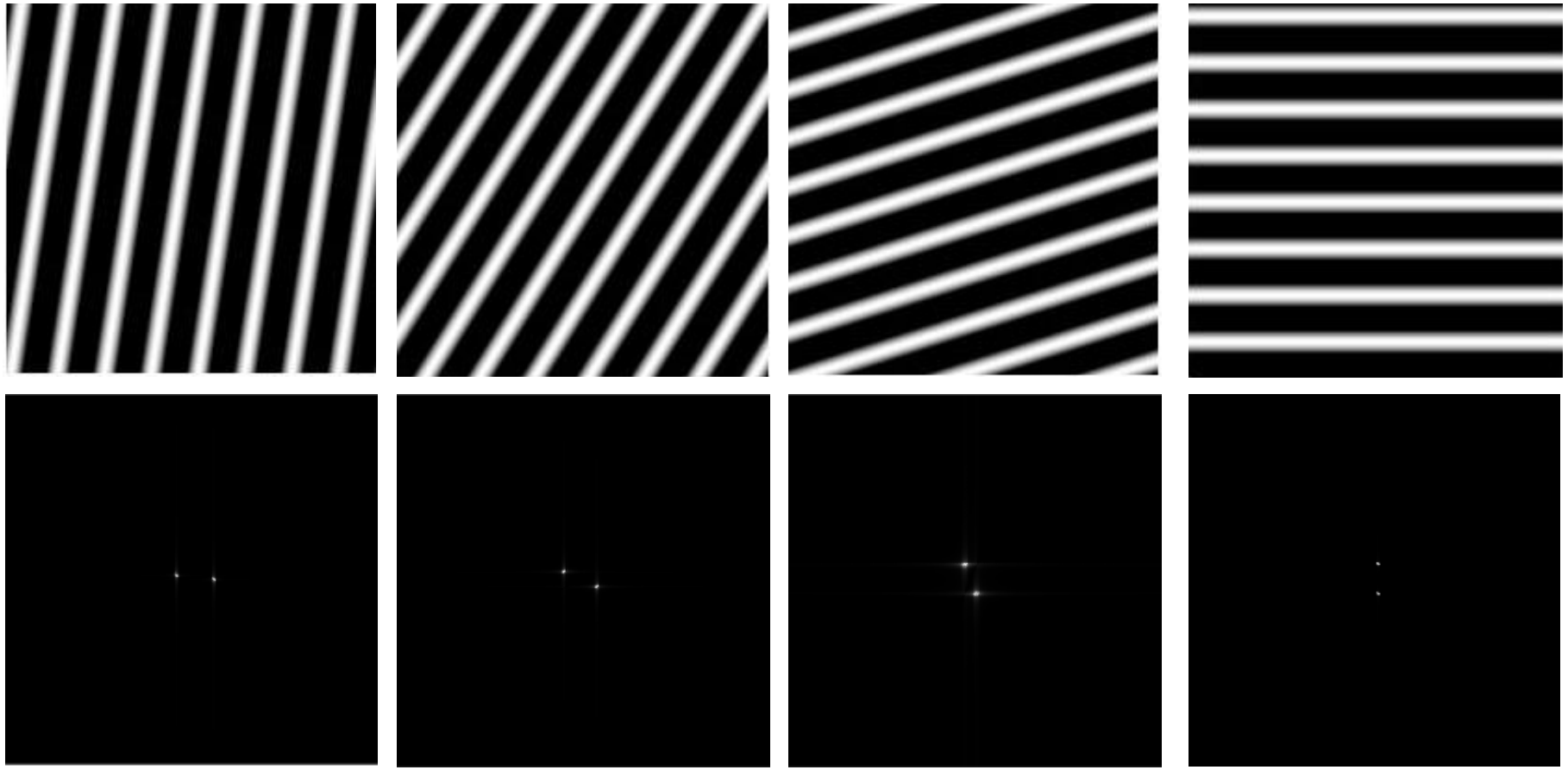


Figure 6. LEFT TO RIGHT: Sinusoid rotated at 30, 45, 60, and 90 degrees and their corresponding Fourier transforms

Combination of Sinusoids

From the first activity, we were tasked to generate a synthetic image of a sinusoid along the x-direction, with amplitude ϵ [0,1] and frequency at 4 cycles/cm. Here, we recreate the same lines of codes and add an fft2( ) and fftshift( ) functions to see what the Fourier transforms of the sinusoid looks like.

```
nx = 200; ny = 200;
x = linspace(-1,1,nx);
y = linspace(-1,1,ny);
[X,Y] = meshgrid(x,y);
f = 4
% Combination of sinusoids
Z = sin(2*pi*4*X).*sin(2*pi*4*Y);
mesh(Z);
FZ = fft2(Z);
FZmod = fftshift(abs(FZ));
figure(2); mesh(FZmod); colormap("gray");
figure(3); imshow(Z);
```

Here, I combined two corrugated roofs, one running in the X-direction, the other in Y. We see that their FT is a binary image of four dots forming a square. This shows the combination of the FTs of the sin functions that we have created earlier.

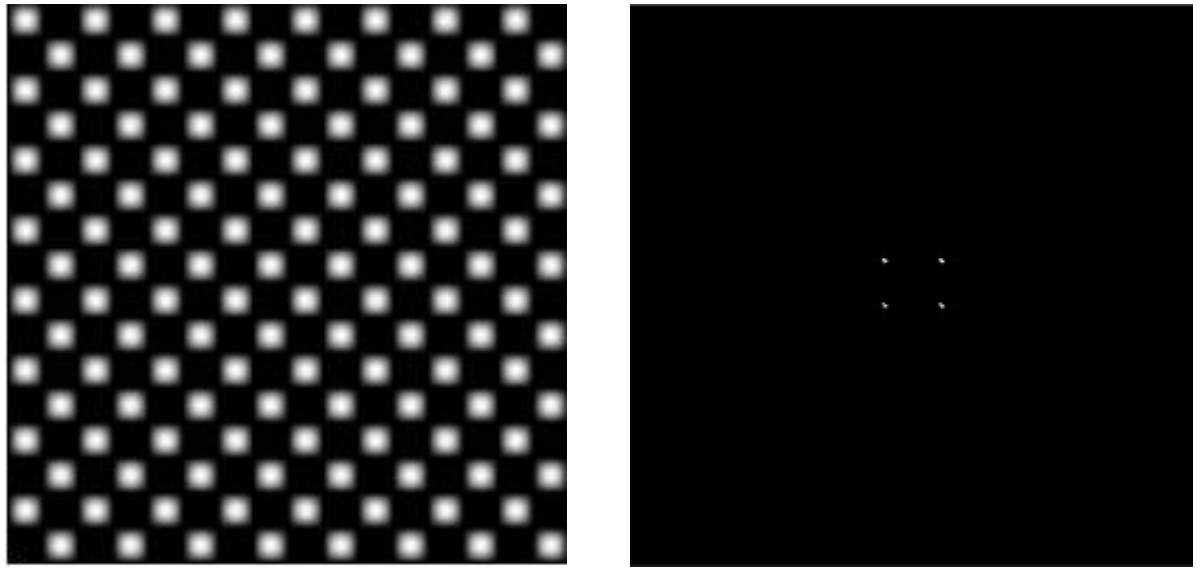


Figure 7. Combination of two corrugated roofs, one running in the X-direction, the other in Y (left) and its FT (right)

Combination of Rotated Sinusoids

Here, we just combine the rotated sinusoids that we have obtained previously and revise our code accordingly. I chose to combine a horizontal sinusoid with another one rotated at 30 degrees, and 60 degrees. I expect to get a diamond like pattern due to the tilting and four tilted dots for the FT.

```
nx = 200; ny = 200;
x = linspace(-1,1,nx);
y = linspace(-1,1,ny);
[X,Y] = meshgrid(x,y);
f = 4
theta = 30 ;
% Combine rotated sinusoids
Zs = sin(2*pi*4*X*theta);
Zr = sin(2*pi*4*(Y*sin(theta) + X*cos(theta)));
Z = Zs + Zr;
mesh(Z); colormap("gray"); view(0,90);
FZ = fft2(Z);
FZmod = fftshift(abs(FZ));
figure(2); mesh(FZmod); colormap("gray"); view(0,90);
figure(3); imshow(Z);
figure(4); imshow(FZmod);
```

I also tried to combine a sinusoid rotated at 30 degrees from the horizontal and another sinusoid rotated at 30 degrees. I expect a series of diamond or kite-like shape, tilted at 30 degrees for this combination.

In the next slide, we will see that we got what we expected for the combination and their FTs.

Combination of Rotated Sinusoids

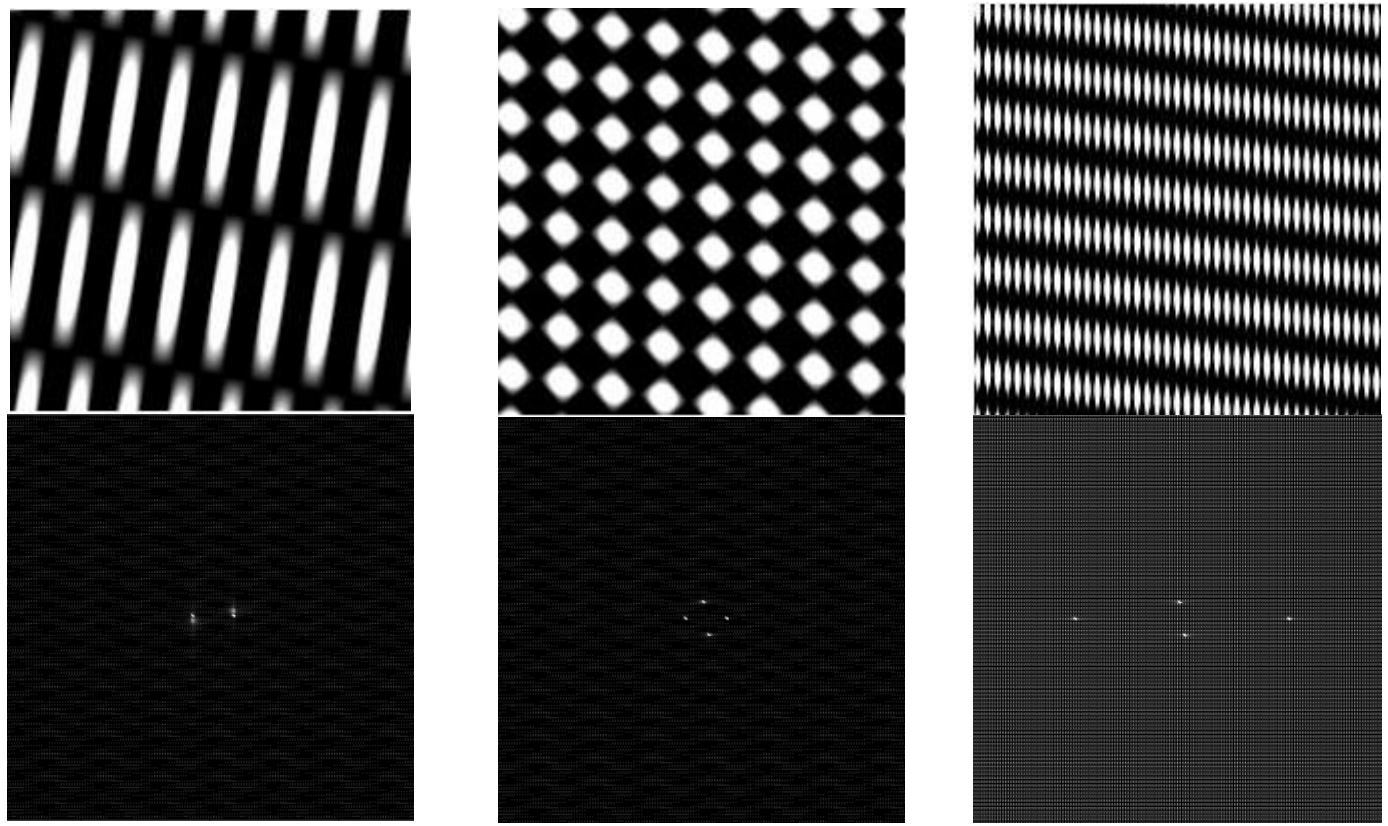


Figure 8. LEFT TO RIGHT: Combination of a (1) horizontal sinusoid and a sinusoid tilted at 30 degrees, (2) horizontal sinusoid and one tilted at 60 degrees, and (3) tilted 30 degrees from the horizontal and another one tilted at 30 degrees, and their corresponding FTs.

Anamorphic Property of FT of Different 2D patterns

Anamorphism is the distortion of a projection or perspective. The Fourier Transform has this property as it maps a 2D function with X and Y dimensions into 1/X and 1/Y dimensions. Thus, we expect that when we perform FT to an object, the resulting image would be distorted. We could use this to our advantage in imaging as we display our wanted pattern.

For this activity, our goal is to display the anamorphic property of the FT using a rectangular aperture with varying dimensions (tall and wide), and two dots symmetric along the x-axis with different spacing. We use a black background and white objects for this activity.

Rectangular apertures

For this activity, we create a tall rectangular aperture using MATLAB and take its FT using the fft2( ) function. Using the same process, we also what happens when we switch the X and Y dimensions and create a wide rectangular aperture instead.

```
f=zeros(200,200);
f(20:180,96:103)=1; %for tall aperture
% f(96:103,20:180)=1; for wide aperture
imshow(f);
F=fft2(f, 256, 256);
F2=fftshift(F);
F2=abs(F2);
figure,imshow(F2,[])
```

Figure 9. Tall rectangular aperture (left) and its FT (left)

As seen on the FT, the tall aperture that appears vertical now looks horizontal. This is expected as we know that the FT maps out the X dimensions as 1/X and the Y dimensions as 1/Y. The result of the transform also showed what looks like a diffraction patter or Airy pattern.

Meanwhile, when we changed the aperture to a wider one, the FT now appears vertical, with the same pattern as the FT of the tall aperture.

Figure 10. Wide rectangular aperture (left) and its FT (left)

Two dots along an axis of symmetry

```
f=zeros(200,200);
f(99:100,71:72)=1;
f(99:100,131:132)=1;
imshow(f);
F=fft2(f, 256, 256);
F2=fftshift(F);
F2=abs(F2);
figure,imshow(F2,[])
```

For this activity, we make two white dots against a black background. The dots are spaced equally from the y-axis. We again take the transform of the image to see what it would look like.

Notice that the two white dots have a FT that looks like a corrugated roof, as in a sine wave. This is because the FT of a 1D sinusoidal function is a pair of Dirac deltas. Thus, if we take the FT of the two dots, it would result to what seems like taking the inverse FT, resulting to a sinusoid.

In the next part, we will see how changing the distance between the two dots would affect the Fourier transform.

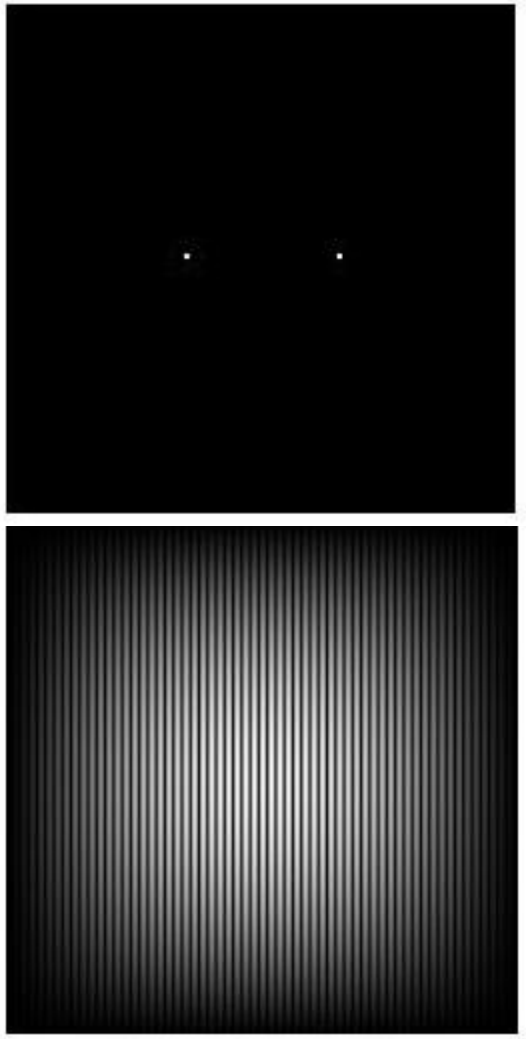


Figure 11. Two dots along an axis of symmetry (up) and its FT (down)

Two dots along an axis of symmetry

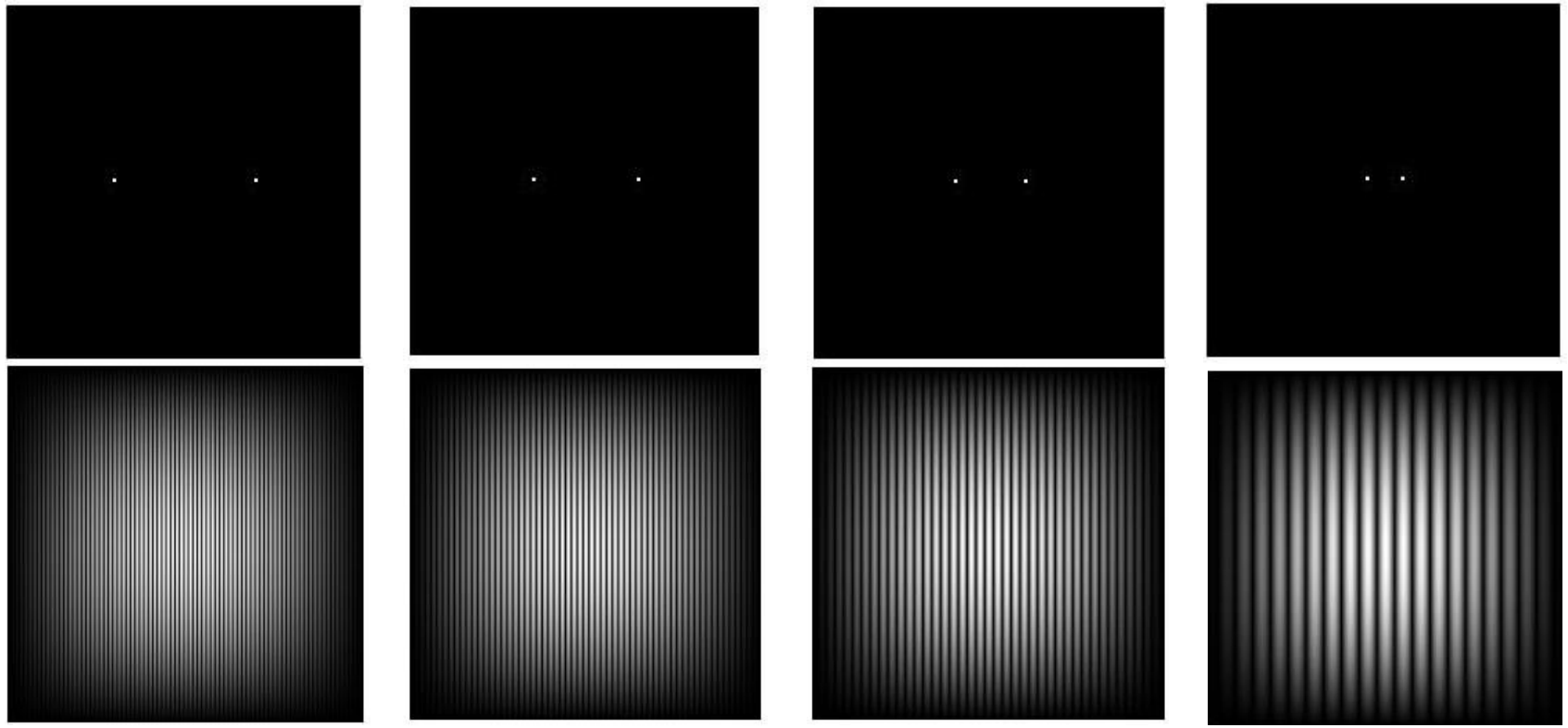


Figure 12. LEFT TO RIGHT: Two dots with separation distance 40, 30, 20, and 10, and their corresponding FT.

Analysis

Here, we demonstrated the anamorphic properties of the FT in different 2D patterns. We saw that for a tall rectangle, its FT is a horizontal slit with an airy pattern, while for the wide rectangle, we have a vertical slit with airy pattern. This is because of the ability of the FT to analyze the object in an inverse dimension such that the X dimensions become 1/X.

Meanwhile, for the two dots, we saw that their FT is a corrugated roof, reminiscent of the sinusoid function. This is expected since the FT of the sinusoid is two Dirac deltas, thus, if we get the FT of the two dots, we seem to be reversing the process and getting the original sinusoid. As we increase the separation distances between the dots, the frequency of the bands increases, which means that the wavelengths decrease.

Convolution Theorem Applications

Convolution is the process of *combining* two functions such that we create a function that looks like the two of them. This is useful mainly in filtering out unwanted areas in a photo, for example. It is given by the formula

$$h(x,y) = \iint f(x',y')g(x - x', y - y')dx'dy'$$

where f and g are two dimensional functions. Meanwhile, if we recast them by a linear transformation such as the Fourier transform, the convolution theorem tells us that

$$H = FG$$

where H, F, and G are the linear transforms of the previous functions.

Convolution Theorem Applications

For this activity, we will be exploring the applications of the convolution theorem by playing around with binary images of two dots, two circles, two squares, and an array of 200×200 zeros with different modifications. We will be taking their Fourier transforms and observe what happens when we vary different parameters such as distance and size.

Two dots along the x-axis symmetric about the center

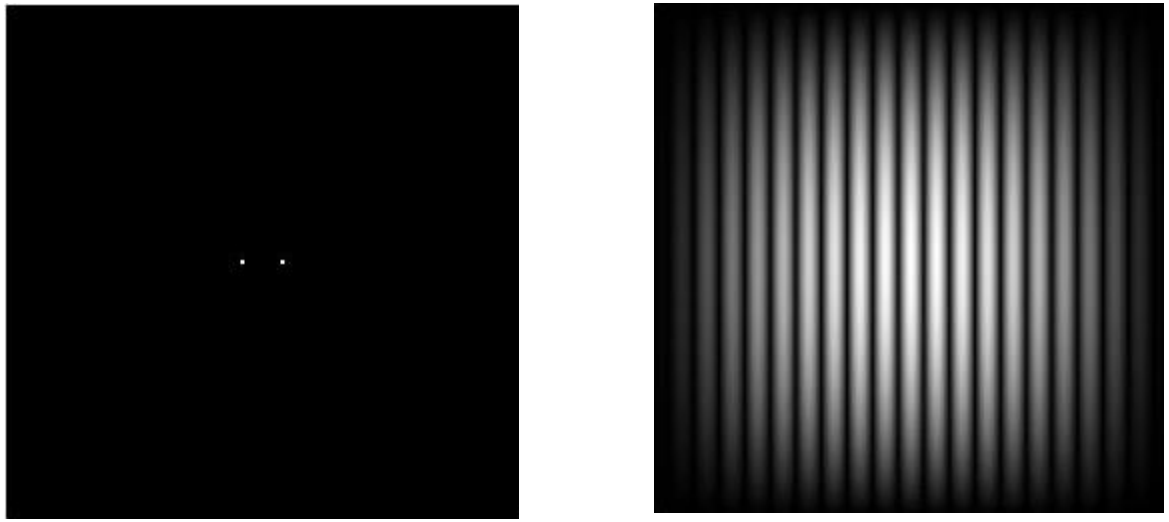


Figure 13. Two dots along an axis of symmetry (left) and its FT (left)

Here, we just recreate the two dots that we generated for the previous activity. Once again, we notice a corrugated roof pattern of the sinusoid for the Fourier transform. What would happen if we change the shape of these dots?

```
f=zeros(256,256);
% Two dots
f(128:129,118:119)=1;
f(128:129,138:139)=1;
imshow(f);
% Fourier transform
F=fft2(f, 256, 256);
F2=fftshift(F);
F2=abs(F2);
figure(1);imshow(F2,[]);
```

The code is just similar to the previous one. We create two white dots against a black background, one pixel each. We then take the Fourier transform using fft2( ) and shift it using fftshift ( ).

Two circles

```
% Black background
SizeX = 256;
SizeY = 256;
[X Y] = meshgrid(1:SizeX, 1:SizeY);

% Circle 1
cX1 = 78;
cY1 = 128;
r = 10;
circle1 = (Y-cY1).^2 + (X-cX1).^2 <= r.^2;

% Circle 2
cX2 = 178;
cY2 = 128;
r = 10;
circle2 = (Y-cY2).^2 + (X -cX2).^2 <= r.^2;

% combining both circles in one image
newImage = circle1 | circle2
imshow(newImage);
F = fft2(newImage)
F2=fftshift(abs(F));
figure(2);imshow(F2,[]);
```

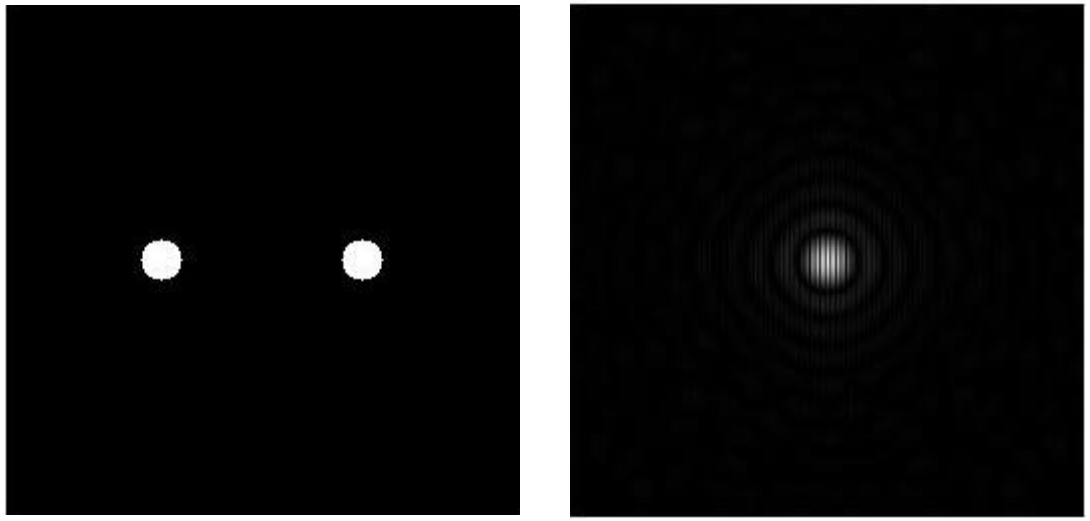


Figure 14 . Two circles with radius 10 and its corresponding Fourier transforms created via MATLAB.

Changing the dots to circles, we just create two circles of radius 10 using MATLAB and combine them. We take the FT of the two and we notice a one circle with an airy pattern. Within it, it looks like there is a corrugated roof pattern.
It can be inferred that the FT of the two circles is a convolution of the FTs of a circle and the two dots from the previous part.

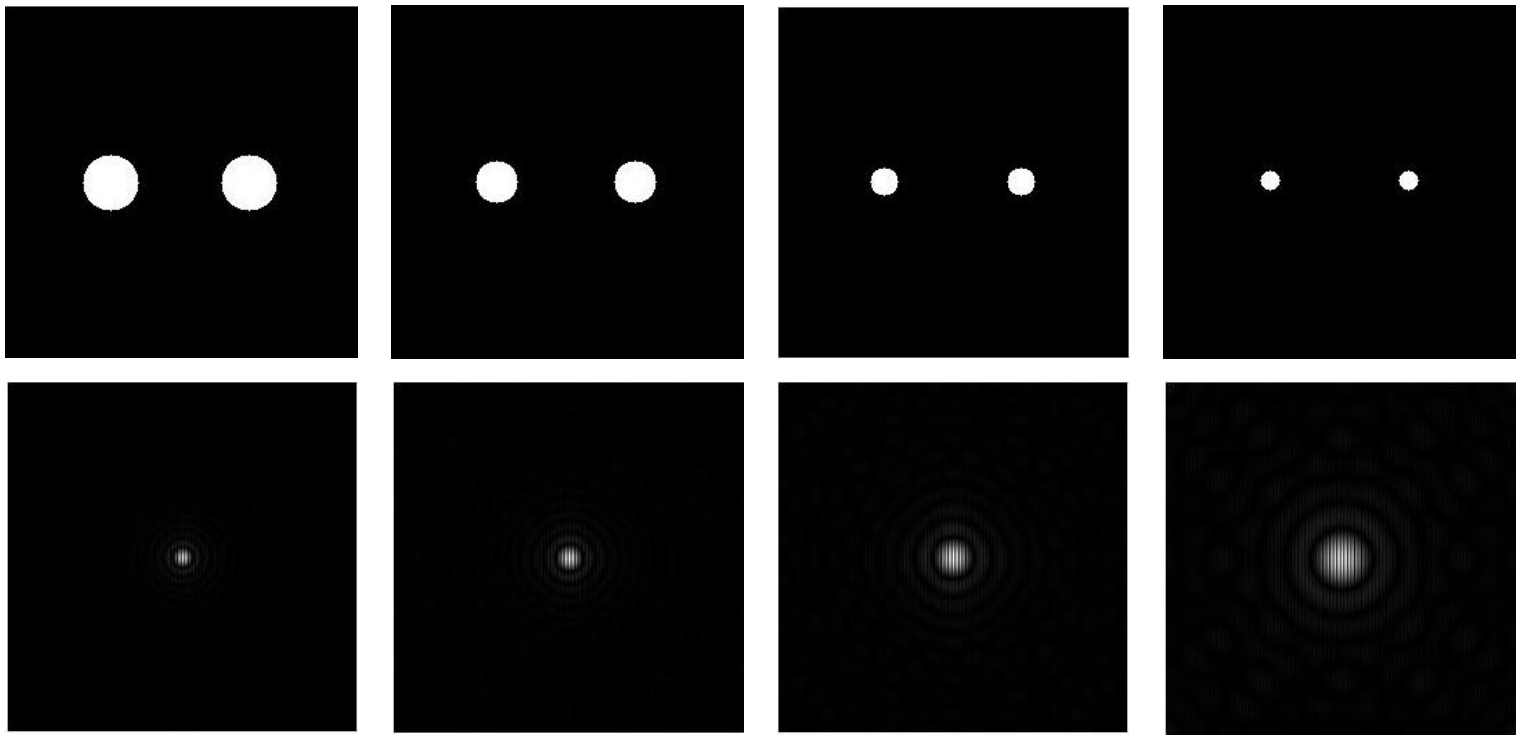


Figure 15. Two circles with radius 15, 10, 7.5, and 5 (left to right), and their corresponding Fourier transforms created via MATLAB.

As we vary the radius of the circle, we notice that the radius of the FT gets smaller as our two circles get bigger. This is expected since the FT is displays the inverse of the dimensions of the original image, hence, if the circle gets bigger in the X and Y dimensions, then it will change in the 1/X and 1/Y dimensions as well. Thus, There is no difference in the corrugated roof pattern because as we have found out previously, the pattern changes as the distance between the dots, or in this case, circles change.

Two squares

```
% Squares
S = zeros(256,256);
S(120:136, 74:90)=1;
S(120:136, 166:182)=1;
imshow(S);
% Fourier Transform
FS = fft2(S);
Fsmod = fftshift(abs(FS));
colormap("gray");
%mesh(Fqmod);
imagesc(Fsmod);
axis equal;
axis tight;
```

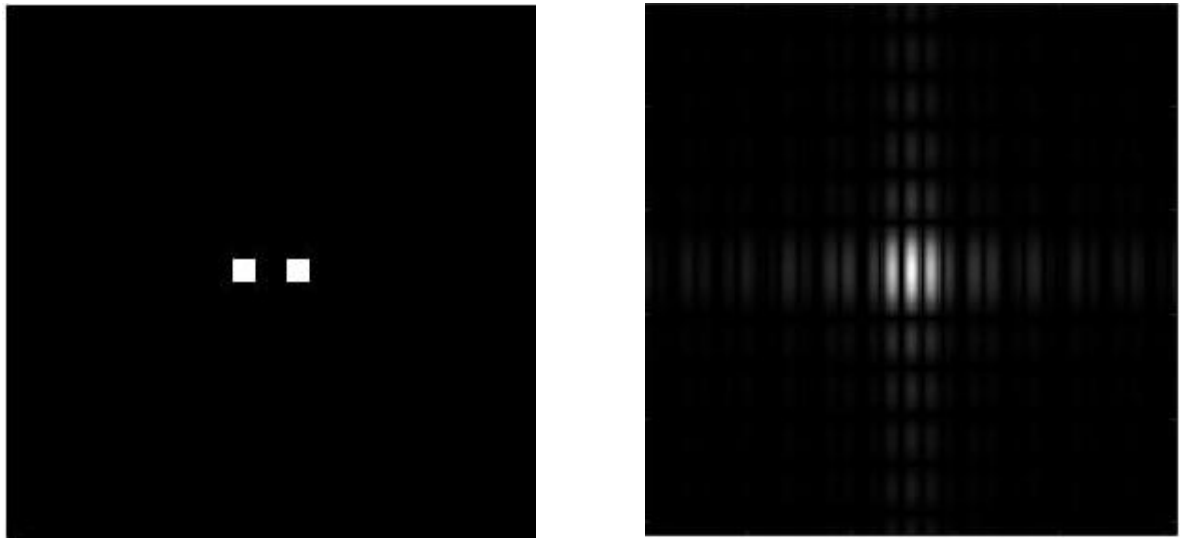


Figure 16. Two squares with width 10 and its corresponding Fourier transforms created via MATLAB.

Changing the circles to squares, we created two squares of width 10 and place them along the axis, symmetric about the center. Notice that the FT of the image looks like the FT of a square but with a corrugated roof pattern.
It can be inferred that the FT of the two circles is a convolution of the FTs of a circle and the two dots from the previous part.

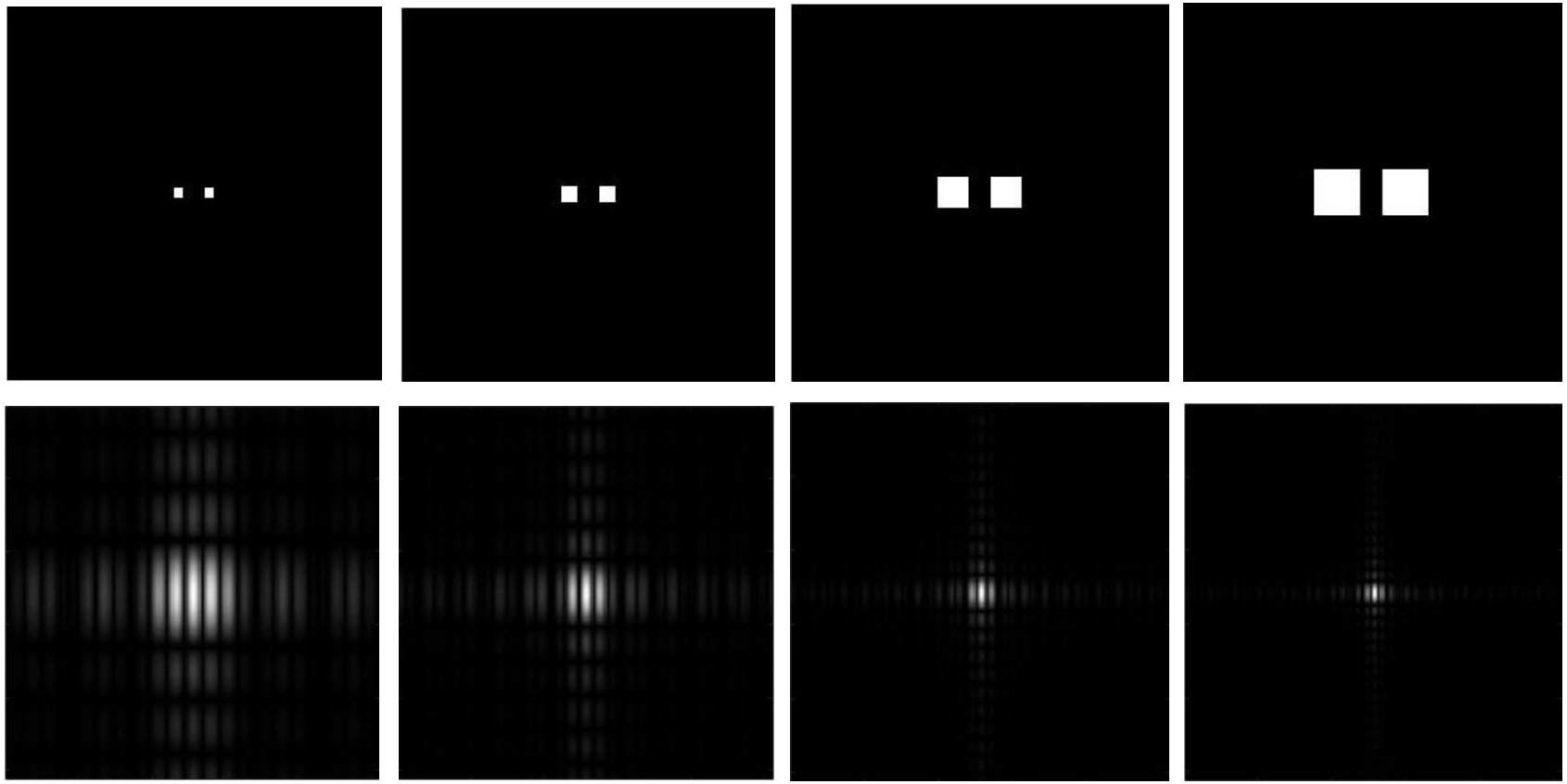


Figure 17. Two squares with width 5, 10, 15, and 20 (left to right), and their corresponding Fourier transforms created via MATLAB.

As we vary the width of the square, we see that the width of the FT increases with decreasing square size. This is just similar to what we have observed for the two circles. Meanwhile, we also observe that the corrugated roof pattern did not change since we did not move the squares away from each other. Let's see what happens when we do this.

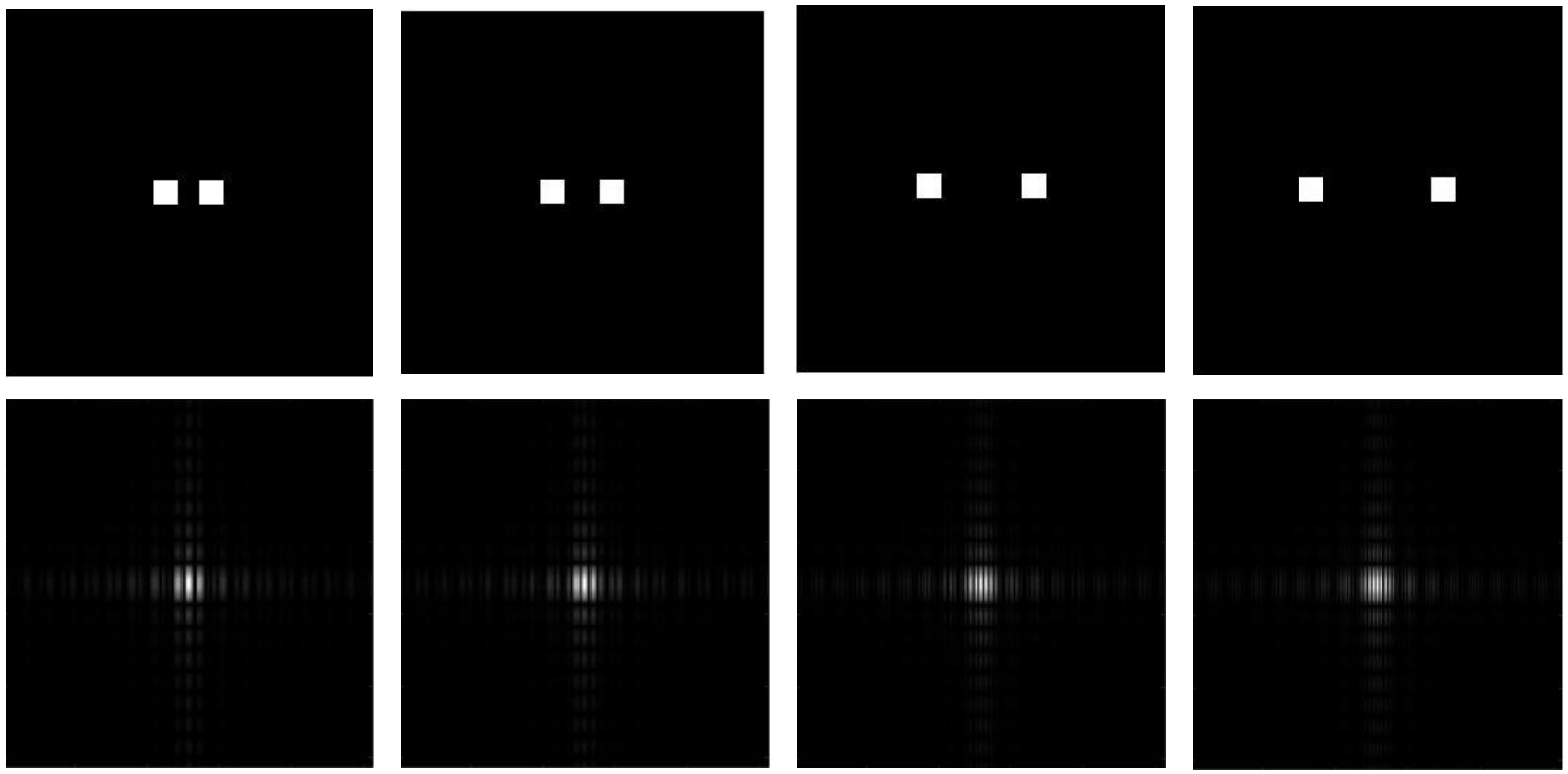


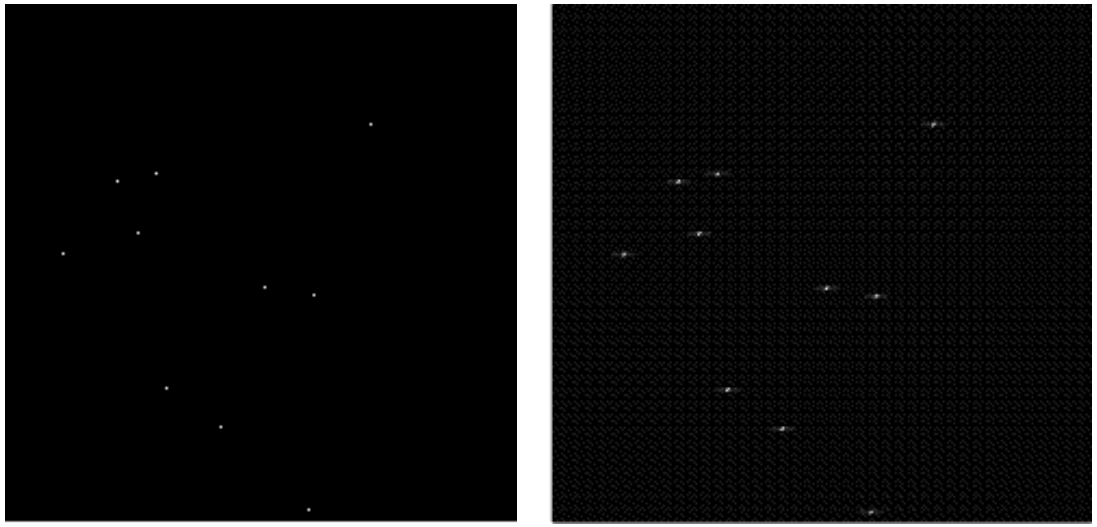
Figure 18 . Two squares with width 10, with separation distances 16, 26, 56, and 76 mm (left to right), and their corresponding Fourier transforms created via MATLAB.

Much like our observation in the separation distances of the two dots, we notice that as the separation distance gets farther, the frequency of the corrugated roof gets larger. This follows the behavior of the FT of the sinusoid that we have found out earlier, except, here, the pattern is a convolution of the square FT and the two dots FT.

Array of 200 x 200 zeros with 10 random 1's

```
A = zeros(200,200);
d= ([[1 1 1], [1 8 1], [1 1 1]]);
A(randperm(numel(A), 10)) =1;
figure(1); imshow(A);
J = convn(A,d);
mesh(J); colormap("gray"); view (0, -90);
axis equal; axis tight;
```

We create a 200 x 200 array containing zeroes and put 10 random 1s in it. We do this using the randperm( ) function. We expect to obtain a 200 x 200 image with a black background and 10 white dots plotted randomly.



Observe that after the convolution, we have made the point brighter. In doing this, I convolved the array with a spot matrix d.

Figure 21. Array of 200 x 200 zeros 10 random 1s (left), and its convolution with a spot matrix (right)

Array of 200 x 200 zeros with 1's in the x and y axes

```
% 200x200 array of zeros with equally spaced 1s along x and y axis
Z = zeros(200,200);
Z(100,[1,10,20,30,40,50,60,70,80,90,100,110,120,130,140,150,160,170,180,190,200])=1;
Z([1,10,20,30,40,50,60,70,80,90,100,110,120,130,140,150,160,170,180,190,200], 100)=1;
%% Fourier Transform
FZ = fft2(Z);
FZmod = fftshift(abs(FZ));
figure(2); imshow(Z);
figure (3); mesh(FZmod);
colormap("gray"); view(0,90);
axis equal; axis tight;
```

For the code, we generated the dots manually by indicating their positions on the graph. We then take their FT and display it.

Here, we see that we have created an image of an array of zeros with 1s on the x and y axes. Its FT shows a grid pattern. We can think of the dots that we have placed to be the Dirac deltas that we played with before, thus the FT is a combination of the sinusoid of the corrugated roof.

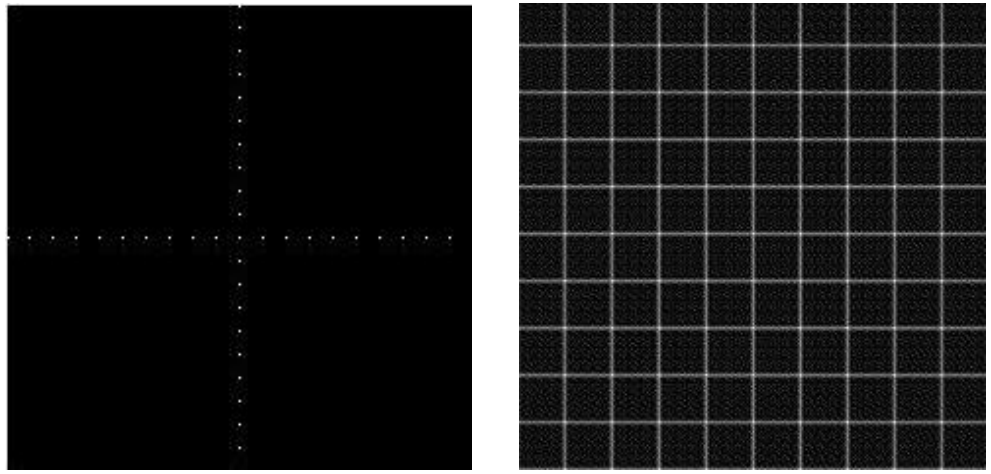
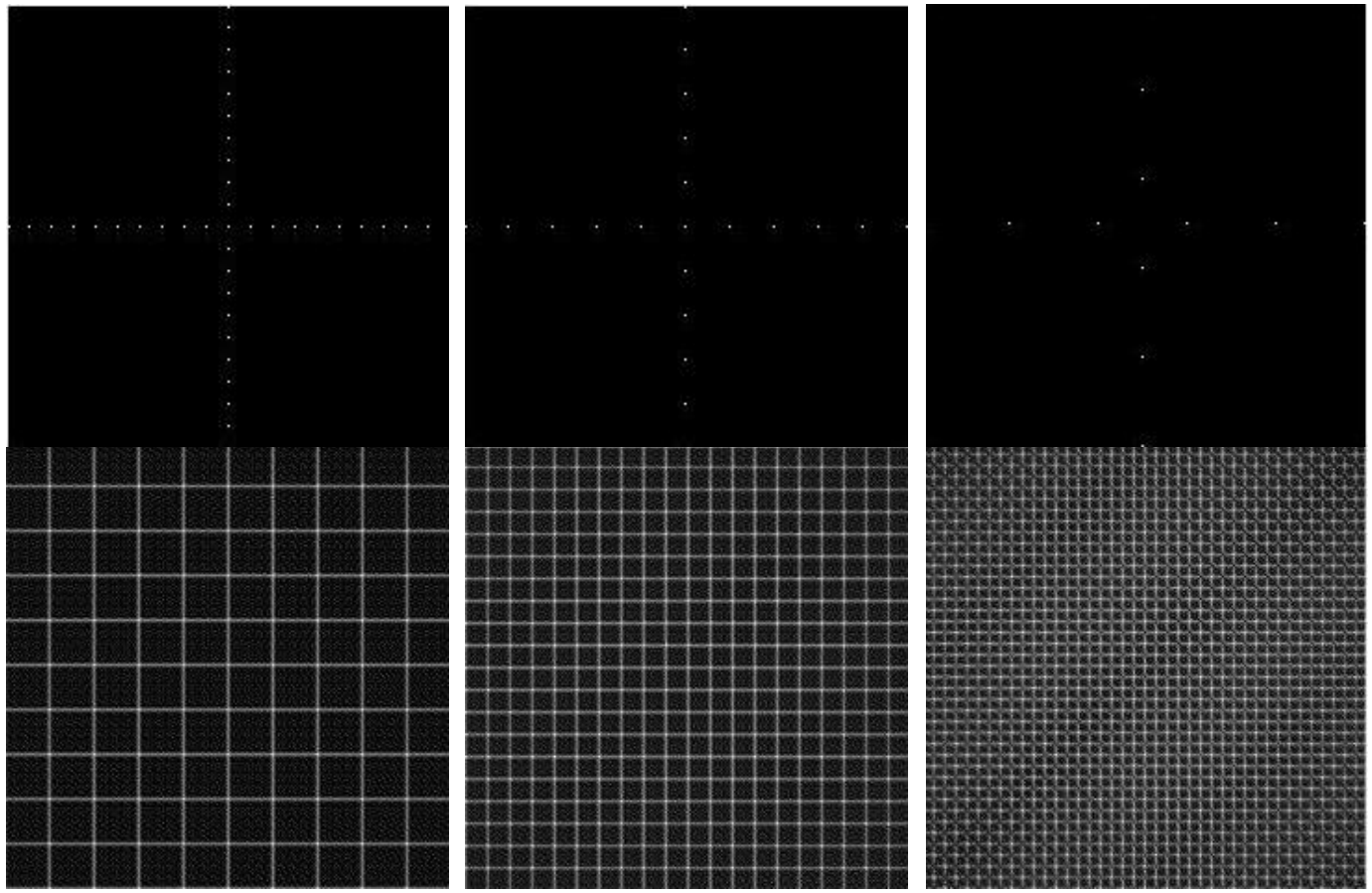


Figure .20 Array of 200 x 200 zeros with 1's in the x and y axes with separation distance 10 and its corresponding Fourier transform

Array of 200 x 200 zeros with 1's in the x and y axes



As the spacing between the white dots increases, the frequency of the grids decrease. This is consistent with our finding earlier that as the spatial distance between the dots increase, the frequency of the sinusoid increases as well.

Figure 21 . Array of 200 x 200 zeros with 1's in the x and y axes with separation distances 10, 20, and 40 mm (left to right), and their corresponding Fourier transforms

Convolution in Pattern Filtering and Removal

Images, especially physically printed ones, are easy subjects for deformation or any changes. In the first activity of this module, we tried to restore a faded image back to its original color. Now, what if it is the texture of the image that is changed and not the color anymore? We can also use convolution in solving this problem!

For the next parts, we will be playing around with convolution applications in order to remove unwanted patterns or textures in images. Here, the main idea is to add a filter to the photo based on its Fourier transforms to mask the unwanted things out and recover or obtain the photo that we want.

Lunar Landing Scanned Pictures: Line removal

For the first part, we use this photo of two lunar craters taken by the unmanned Lunar Orbiter V spacecraft in 1967. Our goal is to remove the vertical lines in the image, which was the result of the combined automatic development of the photo on the spacecraft and its digitizing to be transmitted on Earth.

The plan here is to remove the vertical lines by filtering in the Fourier domain.

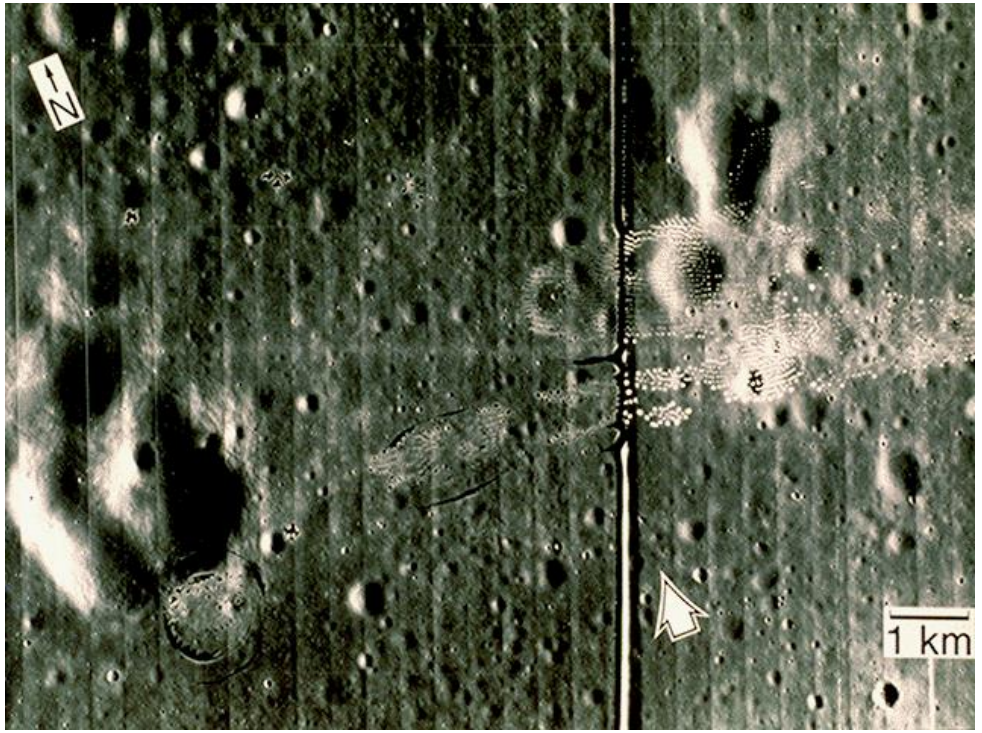


Figure 22. Photo of two lunar craters with vertical lines [1]

Lunar Landing Scanned Pictures: Line removal

```
I = imread('moon.png');
figure(1);imshow(I)

% set the imnage to grayscale
Igray=rgb2gray(I);
figure(2);imshow(Igray)

% get FT modulus of image
F=fft2(Igray);
sF = fftshift(F);

% displays the FT of the image
sFmag = log(abs(sF));
figure(3); imshow(sFmag);
colormap("gray")
```

I started out by importing the image and converting it to grayscale. Now, note that the original file of the moon photo is a GIF. I changed it to PNG so that it can be rendered as is. I took the Fourier transform of the image since this will be the basis of the filter that we will be making later on.

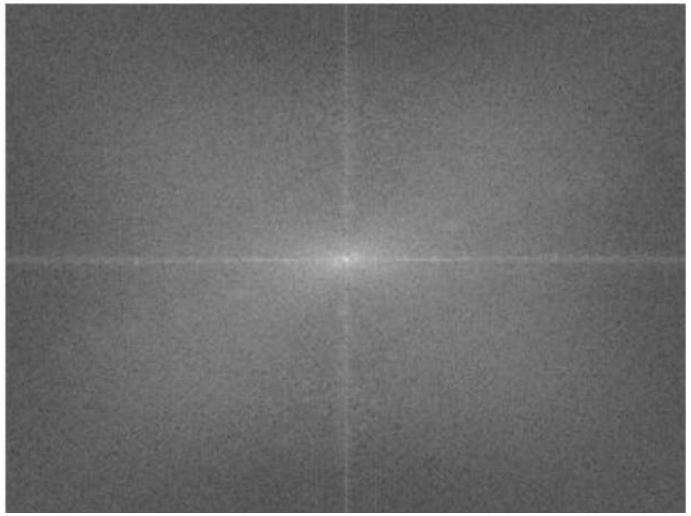


Figure 23. Fourier transform of the moon image in JPG.

First, I got the FT of the grayscale image and shifted it so that the brightest areas of the FT would be focused on the center. This is the FT that I got. It can be seen in figure () that the filter needs to be placed as a horizontal line.

Lunar Landing Scanned Pictures: Line removal

```
%Filter
n=size(Igray);
x = linspace(-1,1,n(1));
y = linspace(-1,1,n(2));
[X,Y] = ndgrid(x,y);
R = sqrt(X.^2+Y.^2);
F=ones(n(1),n(2));
F(find(R>0.02 & abs(X)<0.01))=0;

% displaying FT with the mask
FTcanv2 = log(abs(F));
figure(5);
imagesc(FTcanv2);
colormap("gray")

%convolution
filtF = F.*sF
X= ifftshift(filtF);
filtImg = real(ifft2(X));
imshow(filtImg, [])
```

Now, we create the filter for our Fourier domain. To do this, I made two horizontal strips cut out at the center, which I did by adding the equation for the circle and assigning ones and zeros to the pixels.

The convolution is given by the product of the inverse FT of the image's shifted FT and the filter.

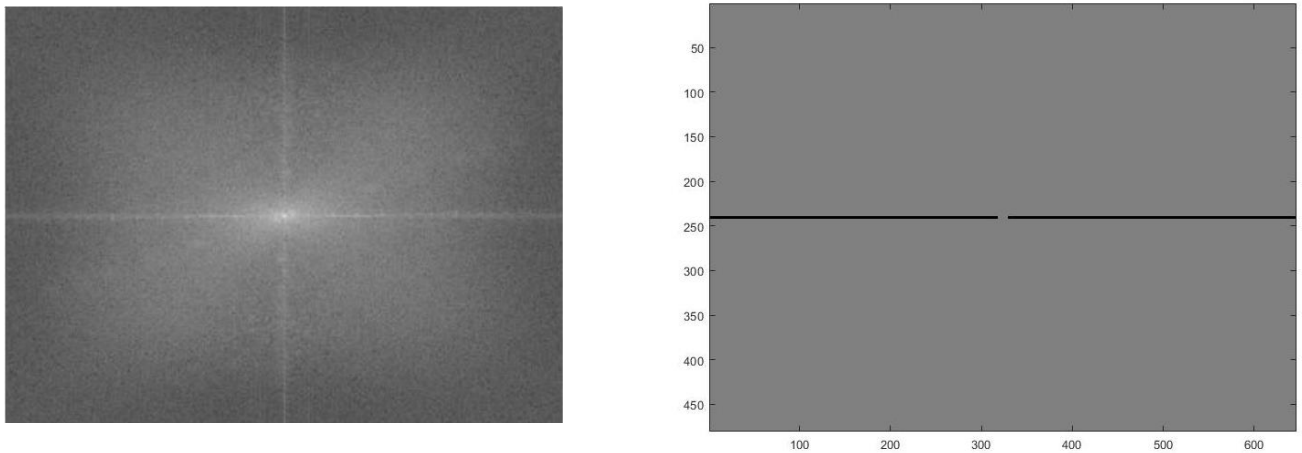


Figure 24. Fourier transform of the moon image and the filter created based on it.

Lunar Landing Scanned Pictures: Line removal

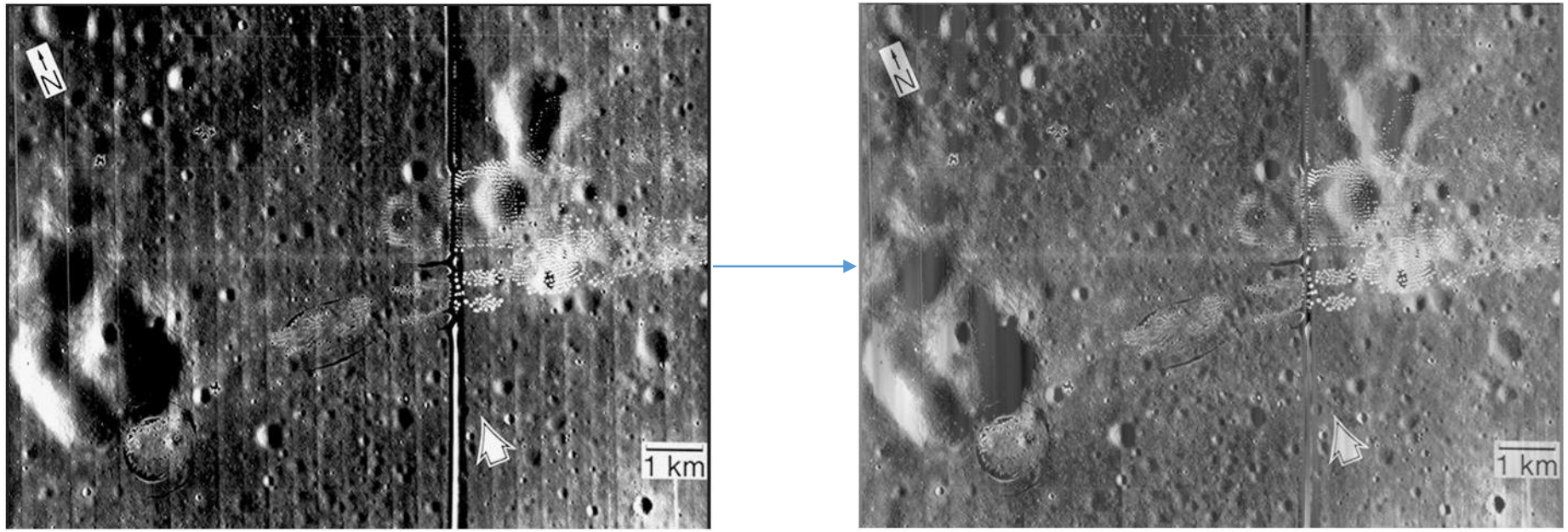


Figure 25. Grayscale of the moon image (left) and the convolved image without vertical lines (right).

This is the side-by-side comparison of the grayscale of the moon image and the convolved image. We can see that the vertical lines formed from the developing and digitizing are now gone and the image looks more representative of the actual scenario as it is free from the unwanted pattern.

Canvas Weave Modeling and Removal

Now, we use convolution to model and remove a canvas weave pattern from an image taken of an oil painting from the UP Vargas Museum.

Here, we notice that it looks like there is a texture in the photo, which most painting are prone to have after having been exposed in museums for a long time. Our goal is to remove the pattern and improve the brush strokes of the painting.

Figure 26 . Detail of an oil painting from the UP Vargas Museum Collection

Canvas Weave Modeling and Removal

```
% turning the image from RGB to grayscaled
canvas = imread("canvasweave.JPG");
Igray = rgb2gray(canvas);
imshow(Igray);

%taking the Fourier Transform
F = fft2(Igray);
sF = fftshift(F);

% displays the FT of the image
sFmag = log(abs(sF));
figure(3);
imagesc(sFmag);
colormap("gray")
```

The process here is very similar to what we did for the lunar photo. First, we convert the image to grayscale and get its Fourier transforms. Based on its shifted FT, we craft a mask that would remove the canvas weave pattern.

Here is the FT of the image. We can see that there are several spots near the center of the image, as well as a vertical and horizontal line in the middle. We will try to patch this to obtain our desired image.

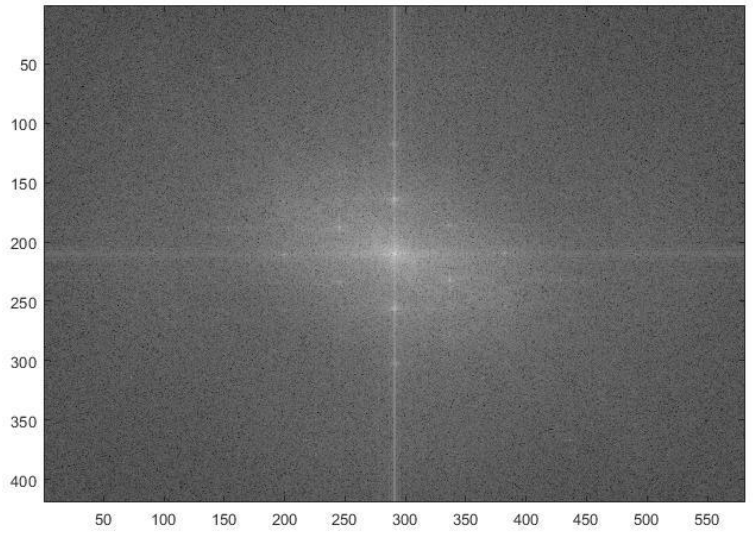


Figure 27. Fourier transform of the oil painting image.

Canvas Weave Modeling and Removal

```
%to create the mask
i=0;
sF(1:185, 288:292)=i;
sF(235:419, 288:292)=i;
sF(208:210, 1:265)=i;
sF(208:210, 316:581)=i;
sF(233:238, 240:247)=i;
sF(184:190, 240:247)=i;
sF(233:238, 334:340)=i;
sF(184:190, 334:340)=i;

% displaying FT with the mask
sFmag = log(abs(sF));
figure(3);
imagesc(sFmag);
colormap("gray")

% displaying the Filtered image
X= ifftshift(sF);
filtImg = real(ifft2(X));
imshow(filtImg, [])
```

Because of the Fourier transform, the challenging part here is creating the mask. The plan is to cover the vertical and horizontal line, as well as the four spots between them, without covering the center. I used the imtools() function to pinpoint the locations of the masks that I want to use. I also assigned zeros to my filter mask.

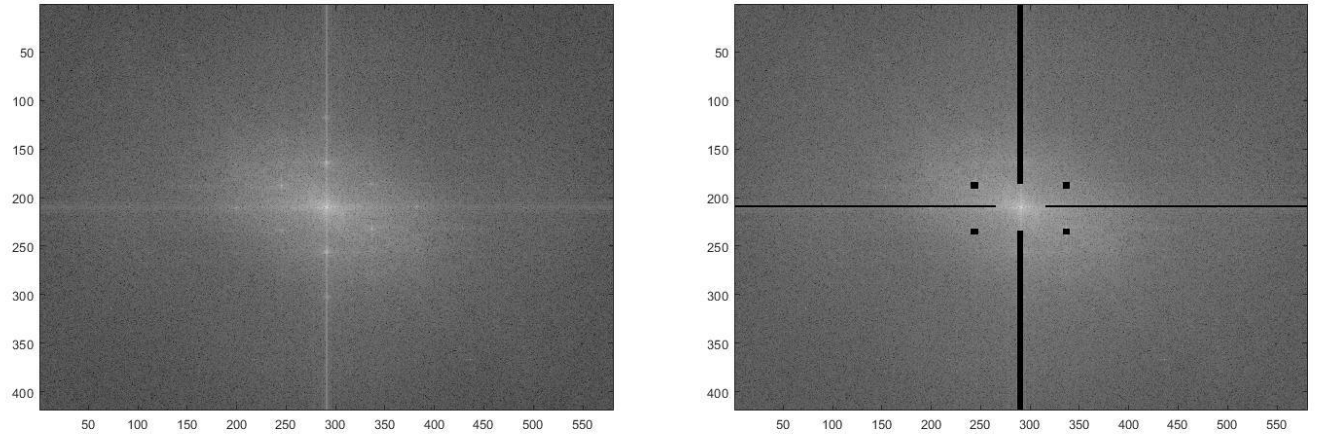


Figure 28. Fourier transform of the painting image and the filter created based on it.

Canvas Weave Modeling and Removal

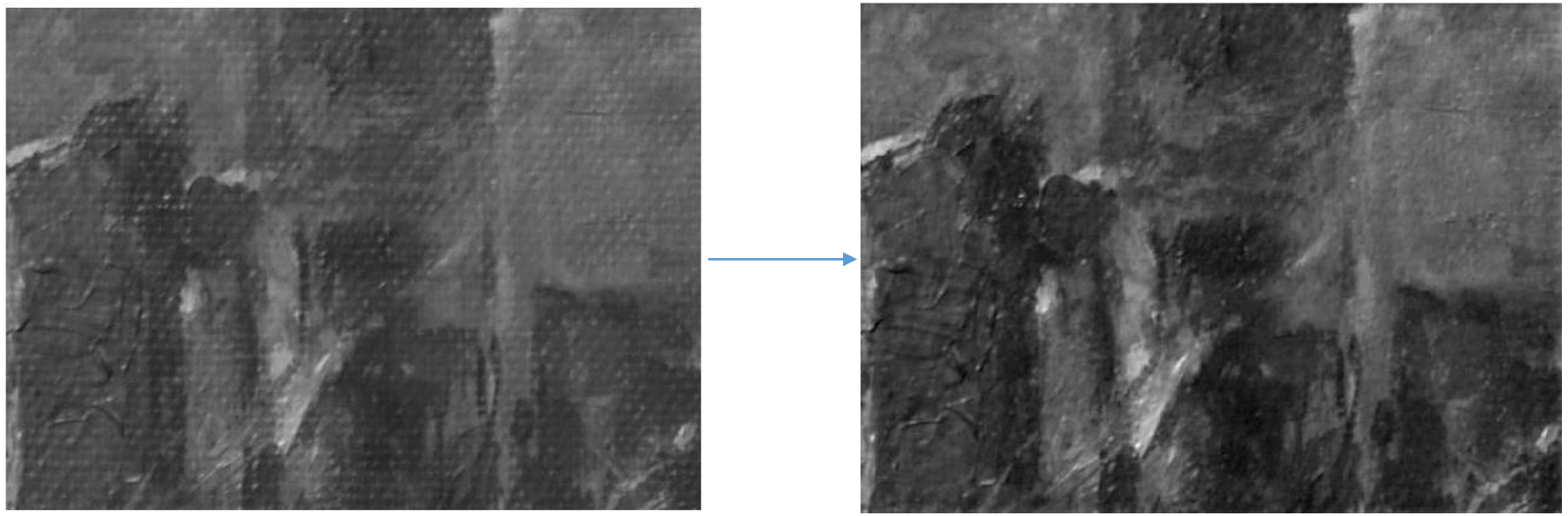


Figure 29. Grayscale of the painting image (left) and the convolved image without canvas weaving (right).

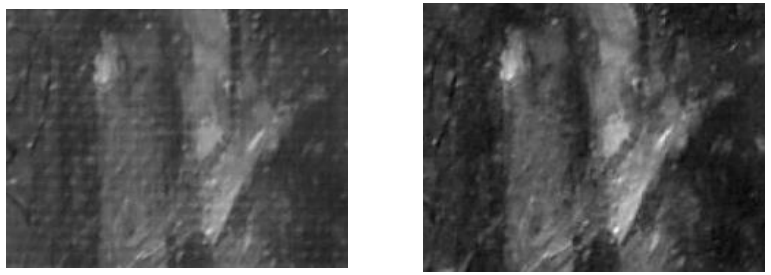


Figure 30. Clearer brush strokes

This is the side-by-side comparison of the grayscale of the painting image and the convolved image. We can see that the canvas weave pattern is not that visible anymore and the brush strokes are clearer in the convolved image.

Analysis

We made use of convolution to filter the images and remove our unwanted patterns. When we shift the image in the Fourier dimension, we see which areas need to be covered, but we make sure to keep the middle area so as not to decrease the quality of our images.

For the moon picture, we created a horizontal filter to remove the vertical lines. This is consistent with our observations earlier where when we take the Fourier transform of an object, we see an inverse of the image since we now take the 1/X and 1/Y dimensions.

Meanwhile, for the oil painting, we modelled a canvas weave pattern so that we could remove the original canvas weave pattern in the photo, which diminished its quality. This technique is really useful in restoring and preserving old artworks and artifacts in museums that got distorted or changed through the years.

Simulation of an Imaging Device

In imaging devices, such as a camera, apertures play a role in the sharpness of an image. The aperture dictates how much light can pass through, the depth of field, and amount of aberrations that may affect an image. In the physical world (outside MATLAB), the smaller the aperture, the sharper the image taken will be.

For this activity, we aim to simulate the ability of an imaging device to sharpen or blur out images by manipulating the aperture size. We make use of convolution using a circular aperture as our mask.
We first create a 128 x 128 pixel photo of the word NIP written in the font Arial using Paint. We convolve this with apertures of varying sizes and see how they affect the NIP Image.

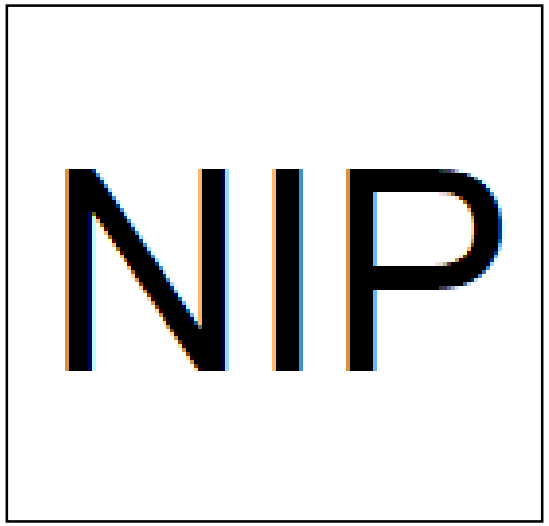


Figure 31. Photo of the word NIP created in Paint.

For the circular lens aperture, we create a 128×128 image of a white circle (centered) against a black background.

For the aperture, I wrote a simple code to generate the desired sizes of the aperture. Notice that in the equation for the circle, I added a coefficient to vary the sizes. The smaller the coefficient, the bigger the size is of the aperture.
I also just saved the images using the imwrite function for convenience.

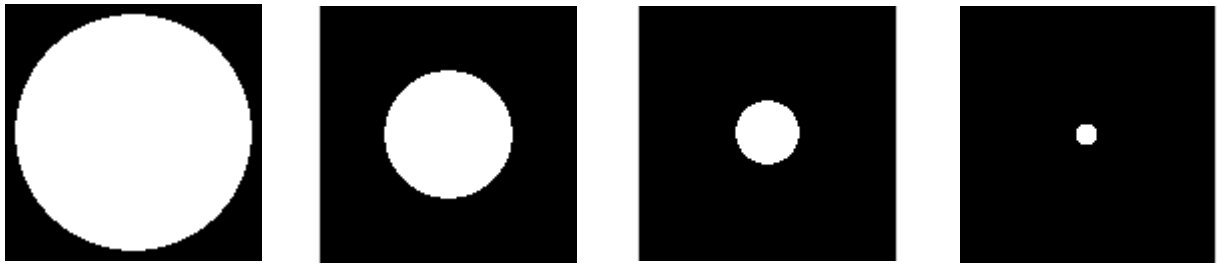


Figure 32. Varying aperture sizes made using MATLAB

```
% We create circles of different sizes
x=linspace(-2,2,128);
y=x;
[X,Y]=meshgrid(x,y);
% Colormap is customized to render a
% black background and a white cirle
R1 = sqrt(X.^2 + Y.^2);
figure(1); imshow(R1);
colormap([1 1 1; 0 0 0]);

% Coefficients are added to change
% the size of the circle
R2 = 0.27*sqrt(X.^2 + Y.^2);
figure(2); imshow(R2);
colormap([1 1 1; 0 0 0]);

R3 = 0.5*sqrt(X.^2 + Y.^2);
figure(3); imshow(R3);
colormap([1 1 1; 0 0 0]);

R4 = 3*sqrt(X.^2 + Y.^2);
figure(5); imshow(R4);
colormap([1 1 1; 0 0 0]);

%imwrite(R1, 'aper1.jpg');
%imwrite(R2, 'aper0.27.jpg');
%imwrite(R3, 'aper0.5.jpg');
%imwrite(R4, 'aper3.jpg');
```

Get the product of their FFT and get the inverse to get the convolved image

Here, we import the NIP and the aperture images and turn them into grayscale. We also made sure that they are of the same sizes so that we can convolve them easily later.
We take the Fourier transform of the NIP picture, but not of the aperture anymore as it is already its own transform. Instead, we use the shifted FT of the aperture and convolve it with the transform of the NIP picture, and then we take their inverse.

```
a = imread("NIP.png");
figure(1); imshow(a);
r = imread("3aper.png");
R = imresize(r,[128 128]);
% Grayscale of the images
rgray = im2gray(R);
agray = im2gray(a);

Fr = fftshift(double(rgray));
%aperture is already in the Fourier Plane
% and need not be FFT'ed
Fa = fft2(agray);
FRA = Fr.*(Fa);
IRA = fft2(FRA); %inverse FFT
FImage = abs(IRA);

imagesc(FImage);
colormap(gray(256));
```

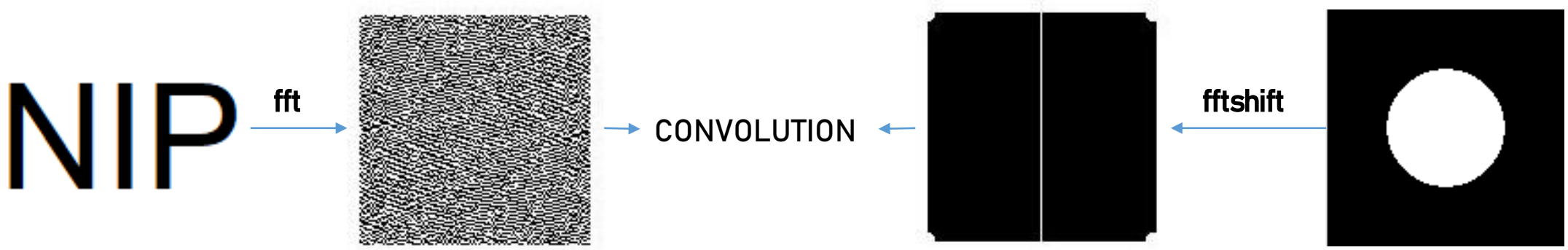


Figure 33. Diagram of the convolution process. Note that the fft of the NIP image displays only the real part of a complex input.

Convolution Results

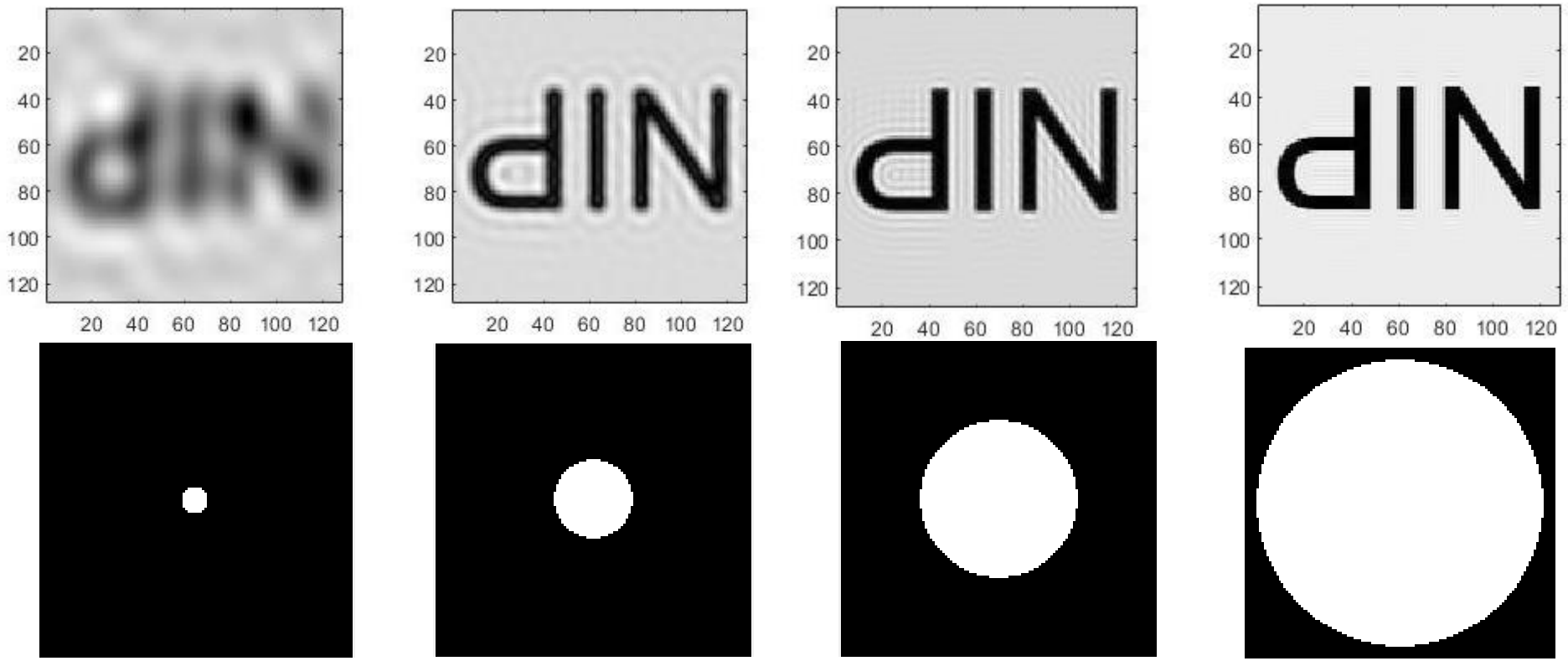


Figure 34 . Results of the convolution with their corresponding apertures below.

Analysis

Notice that as the aperture decreases in size, the object becomes more blurred. This is contrary to what we have learned in optics where the image gets sharper as the aperture is decreased. This may be because we applied the Fourier transform to the objects thus, the dimensions will be inverted. We also notice that the NIP word got inverted as well, which is consistent with what we observe in optics where the image gets inverted as the light rays pass an aperture.

Meanwhile, we also see that for the smaller radius, we notice an airy pattern on the resulting NIP image. This is consistent with our observations earlier with the FT of a circle, giving off an airy pattern.

Template Matching Using Correlation

Correlation is the measurement of the similarities between two functions. This is given by

$$p(x,y) = \iint f(x',y')g(x + x', y + y')dx'dy'$$

We also have a theorem that relates the linear transforms of f and g: $P = F*G$, where P, F, and G are the FT of the functions above, and * stands for complex conjugation.

In imaging, we can use this to match objects with a template, to see if they exist in that said template. This is useful for pattern and image recognition applications.

Template Matching Using Correlation

Here, we make use of the 128 x 128 pixel template saying "THE RAIN IN SPAIN STAYS MAINLY IN THE PLAIN", which was created using Paint. We also create an object containing the letter "A" to match with our template. The font is Arial.

We aim to correlate the two images and see their points of intersection using the element by element Fourier transform of A and the conjugate of the template.

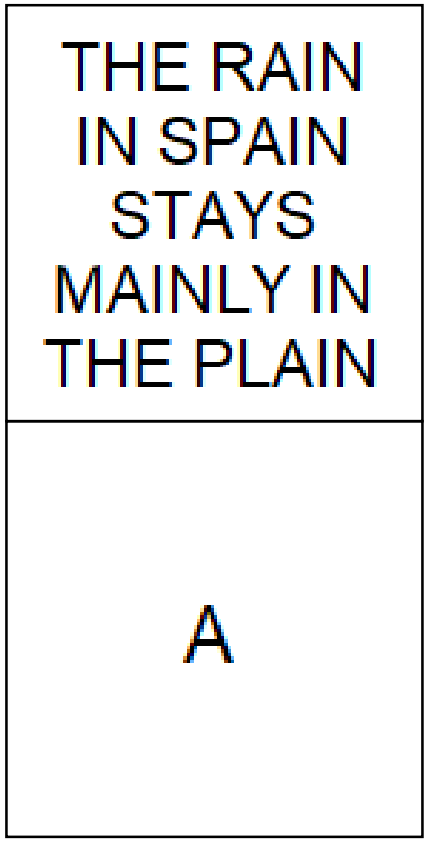


Figure 35 . Photo of the word NIP created in Paint.

Template Matching Using Correlation

We first import the template and our object to match in grayscale. We take the Fourier transform of "A" and multiply it to the conjugate of the transform of the template, following the correlation theorem. We then get the inverse transform of the correlated image and see where the points of signal similarities are.

Here, I added a few extra lines of codes to get my desired images, such as customizing the colormap to be gray, and making my axes equal and tight to produce a 128 by 128 image.

```
plain = imread("rain.png");
spain = imresize(plain, [128 128]);
pic = imread("a.png");
a = imresize(pic, [128 128]);
axis on;
sgray = im2gray(spain);
agray = im2gray(a);
Fr = fft2(sgray);
%aperture is already in the Fourier Plane
% and need not be FFT'ed
Fa = fft2(agray);
% We use the correlation theorem
FRA = Fa.*conj(Fr);
IRA = fft2(FRA); %inverse FFT
FImage = abs(IRA);
imagesc(FImage);
colormap(gray(256));
axis equal;
axis tight;
```

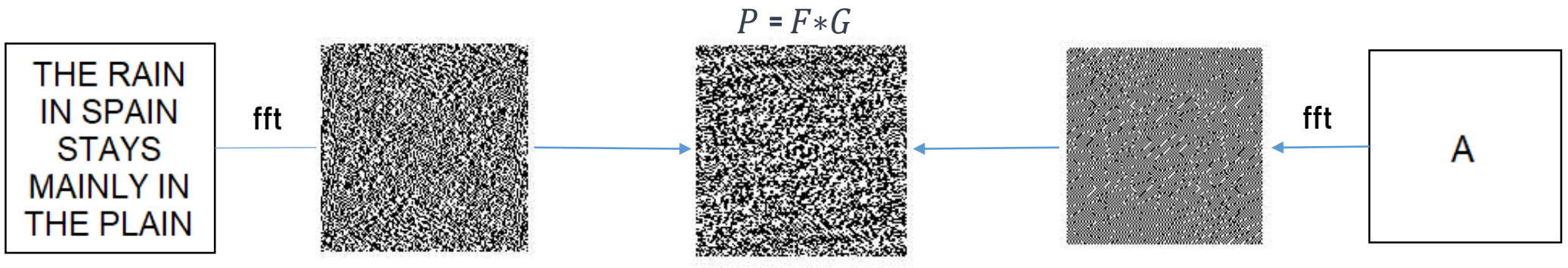


Figure 36 . Diagram of the correlation process. Note that the fft of the images displays only the real part of a complex input.

Template Matching Using Correlation

After correlating the template and the image of "A", we obtain the template matching result, which can be seen in the image on the right. Here, we see that there are four marks wherein the similarities are the highest. These indicate the areas where the template and our object matched.

Template matching is very useful in image detection as it allows us to pinpoint intersections of similarities, which may be used in identifying an image. This is very related to what we did for the second activity, which is feature extraction, since we also had to identify the particles there.

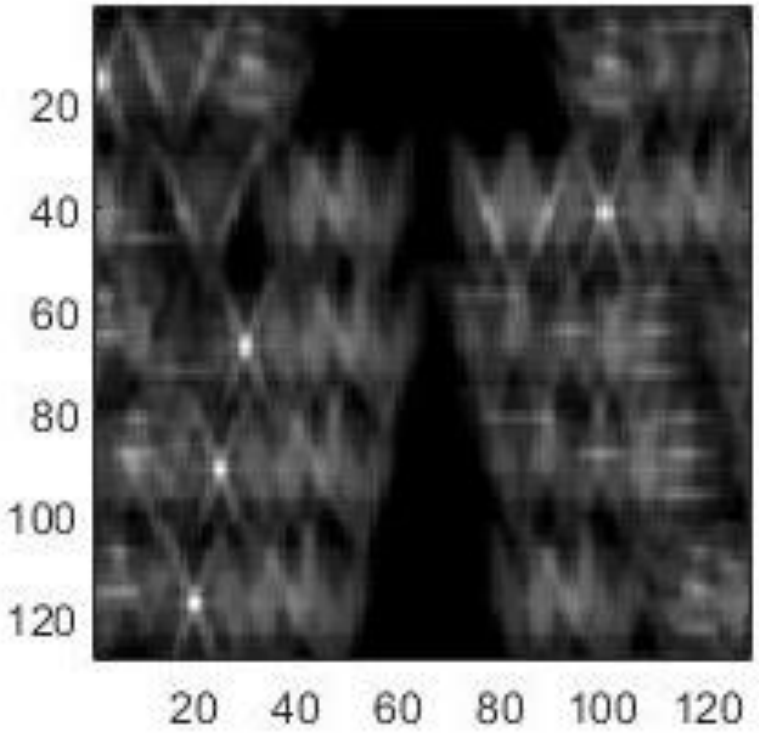


Figure 37. Result of the template matching via correlation with "A" as the object.

Template Matching Using Correlation

I also tried to perform template matching using the other most frequently appearing letters in the template, I and N. The left side of the figure above shows the template matching with I while the right side is with N. We see that there are more points of intersections with I since we can almost make up the letters in the correlation when we look at it. Meanwhile, for N, there are more number of clear intersection marks (8) than with A (4).

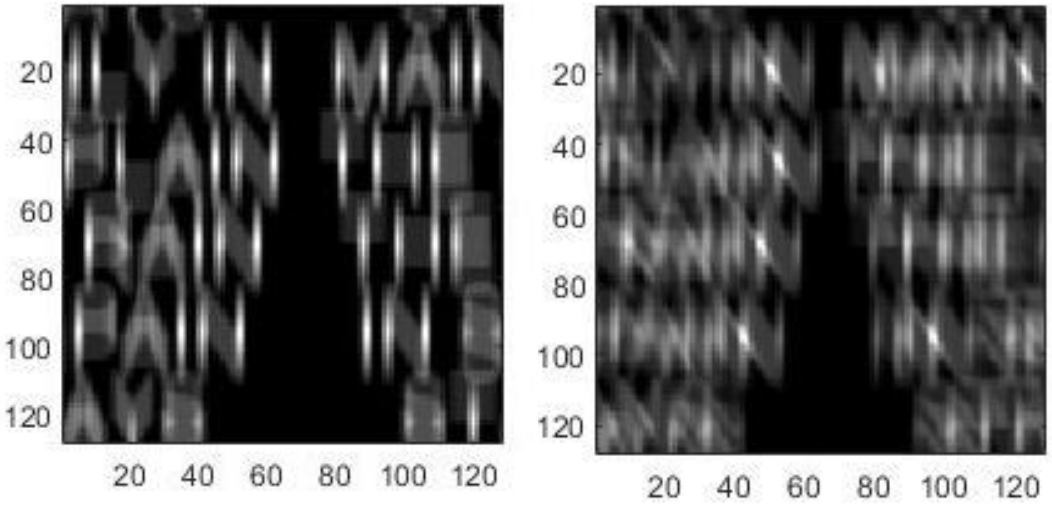


Figure 38. Result of the template matching via correlation with "I" as the object (left), and "N" (right).

The frequency of appearance of A, I, and N in the template are 5, 6, and 6 respectively. However, we see that there is a large difference in their template correlation. This may be because I is much less complicated than the letters A and N, since it is just one vertical line, which may have made the algorithm think that it matched with most of the letters with a vertical line as well.

Edge Detection Using the Convolution Integral

In the second activity, we tried feature extraction and was able to identify the outlines of each particle in the process. In this activity, we do a similar and related process of edge detection using the convolution integral. Here, we perform something similar to the template matching, except we use a 3x3 matrix to convolve with the NIP image we created earlier.

The goal of this portion is to find the matrix that will give us the best edge detection result.

Edge Detection Using the Convolution Integral

```
NIP = imread('NIP.png');
NIP = im2bw(NIP);
% This is the 3x3 matrix
pattern = [2 -1 -1; -1 2 -1; -1 -1 2];
convolve = conv2(NIP, pattern);
imshow(convolve);
```

The code here is just short thanks to the conv2() function of MATLAB. What we do here is to convert the NIP image to binary using im2bw() so that we can apply the convolution. We then input the matrix that we want to use and convolve it with the binary NIP image. For this activity, we use the following matrices:

$$\begin{matrix} -1 & -1 & -1 \\ 2 & 2 & 2 \\ -1 & -1 & -1 \end{matrix}$$
Horizontal

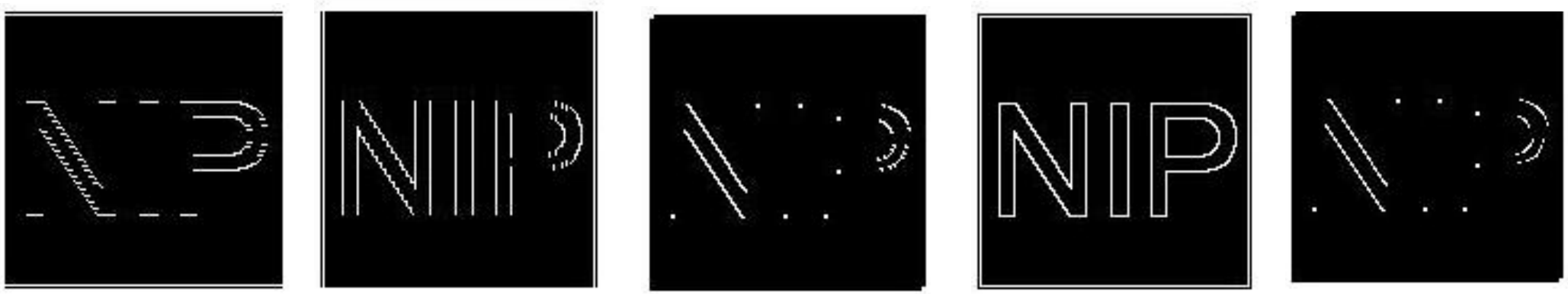
$$\begin{matrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{matrix}$$
Vertical

$$\begin{matrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{matrix}$$
Diagonal

$$\begin{matrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{matrix}$$
Spot

$$\begin{matrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{matrix}$$
Edge

Edge Detection Using the Convolution Integral



Horizontal	Vertical	Diagonal	Spot	Edge
$\begin{matrix} -1 & -1 & -1 \\ 2 & 2 & 2 \\ -1 & -1 & -1 \end{matrix}$	$\begin{matrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{matrix}$	$\begin{matrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{matrix}$	$\begin{matrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{matrix}$	$\begin{matrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{matrix}$

Figure 39. Results of the convolution of the NIP image with different matrices.

Analysis

For the edge detection, we observe that the spot matrix produced the best result. Meanwhile, the patterns used in the convolution can be seen very clearly in the results. For example, with the horizontal matrix, we see that the brightest edges detected are those along the horizontal, thus the vertical lines of the N, I, and P are erased. Meanwhile, the opposite is true for the vertical matrix since we see that the vertical lines appear the brightest.

Meanwhile, for the diagonal, we see that only the diagonal of the letter N can be seen the clearest, since it follows the pattern. For the edge matrix, we observe a similar result with the diagonal matrix, except the edges of the letters appear the brightest.

Reflection

Technical

The canvas weave pattern and line removal took me the longest to figure out. For the moon surface, I was so frustrated that my desired image did not show up. It turns out, you have to change the file into JPG so that the software could read it properly. When I imported it as a GIF, I got this image attached.

Meanwhile, I found it difficult to display the FT of the objects I created because it would not display my expected objects. Instead, I tried to mesh the FT and display it in view(0, 90) and this is where I got the two dot FTs of the sinusoids, and the FTs of the other objects.
Lastly, I did not know how to plot the 200x200 array of zeros with random 1s. I was able to figure it out after my friend helped me how to generate random numbers using MATLAB, since I was used to doing this in Python.

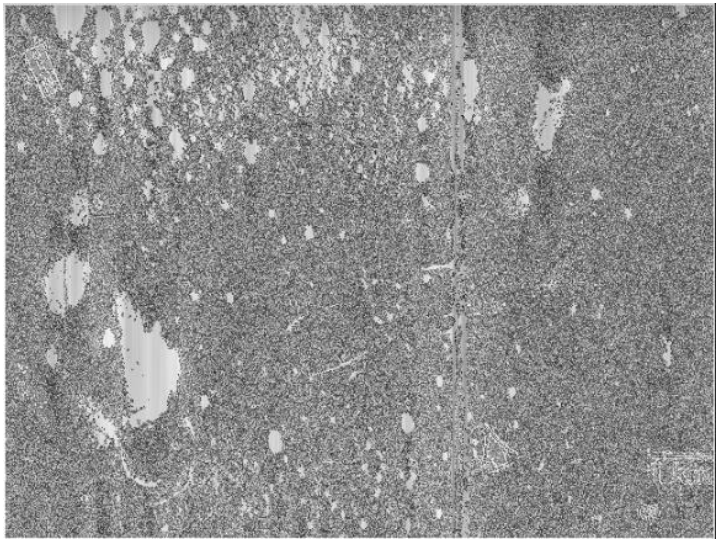


Figure 40. Moon surface image imported as GiF.

References

[1] NASA Lunar Orbiter. Obtained from
http://www.lpi.usra.edu/lunar/missions/apollo/apollo_11/images/hi_res_vertical_lg.gif

[2] Richards, P (2004). "On Fourier Transforms and Delta Functions," Obtained from
https://www.ldeo.columbia.edu/~richards/webpage_rev_Jan06/Ch3_FourTrans%26DeltaFns.pdf

[3] Soriano, M. (2022), "FOURIER TRANSFORM PROPERTIES AND APPLICATIONS"