P166 - March 7 Homework

March 7, 2019

```
In [1]: #Author: Charles Jason C. Diaz
        #Notebook can be found in https://github.com/werhu/Physics

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

```
In [2]: #Constant size of images constructed
        size = 128
```

0.1 RGB to Grayscale conversion function

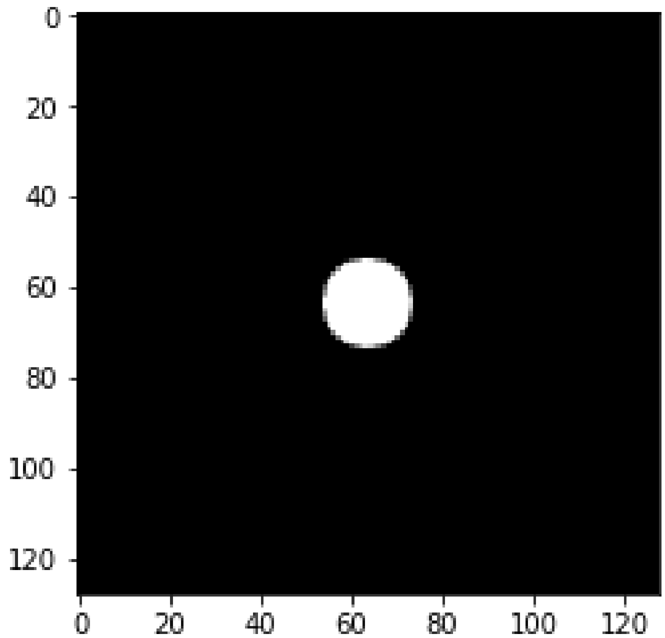
```
In [3]: #converts an rgb array into a grayscale array
        def rgb2gray(rgb):
            return np.dot(rgb[...,:3], [0.299, 0.587, 0.114])
```

0.2 Familiarization with discrete FFT

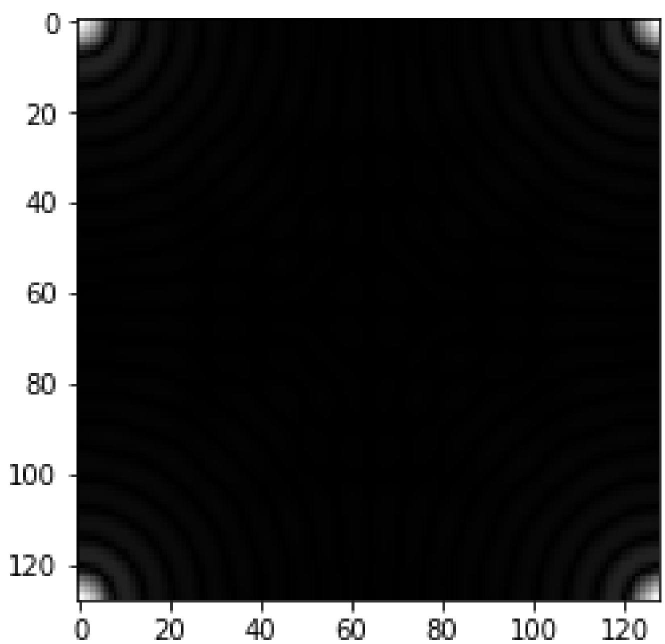
```
In [4]: def fourier(data):
            plt.imshow(data, cmap = 'gray') #Displays the initial image
            plt.show()
            datafft = abs(np.fft.fft2(data))
            plt.imshow(datafft, cmap = 'gray') #Displays the FT of the image
            plt.show()
            datashift = np.fft.fftshift(datafft)
            plt.imshow(datashift, cmap = 'gray') #Displays the FT modulus of the image
            plt.show()
            datafft2 = abs(np.fft.fft2(datafft))
            plt.imshow(datafft2, cmap = 'gray') #Displays the FT of the FT of the image
            plt.show()
```

0.2.1 Circle Fourier transforms

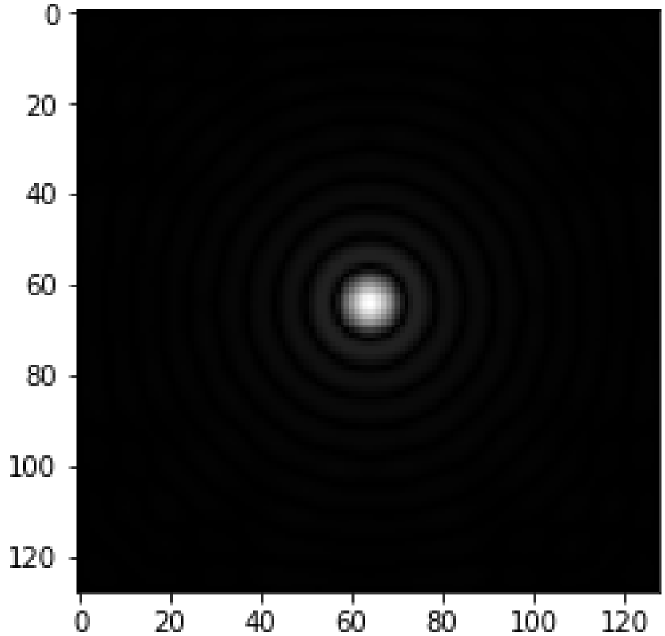
```
In [5]: circ = rgb2gray(plt.imread('circle.png'))
        fourier(circ)

        """As seen, the expected result for the circle can be seen: the shifted 2D FFT of the 
        The image of the circle is not perfect, meaning it has low resolution. This gives the 
        The unshifted 2D FFT has the quadrants rotated, which places the corners of the circle
        """
```

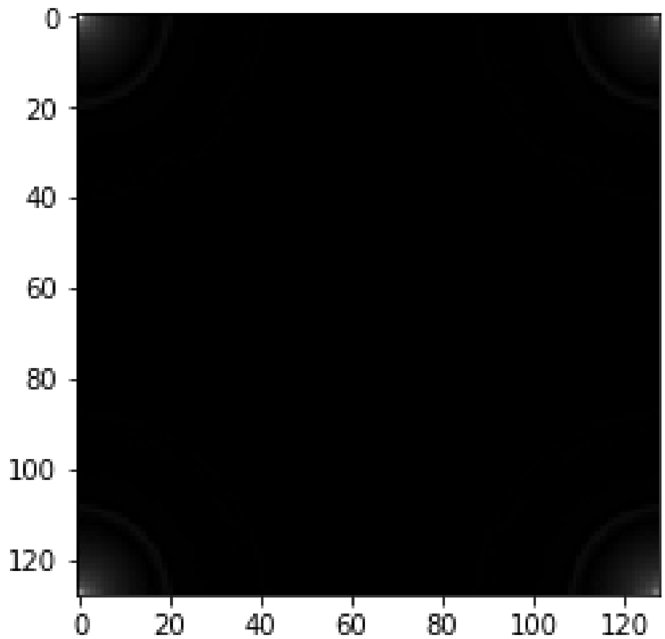


0
20
40
60
80
100
120
0
20
40
60
80
100
120



0
20
40
60
80
100
120
0
20
40
60
80
100
120

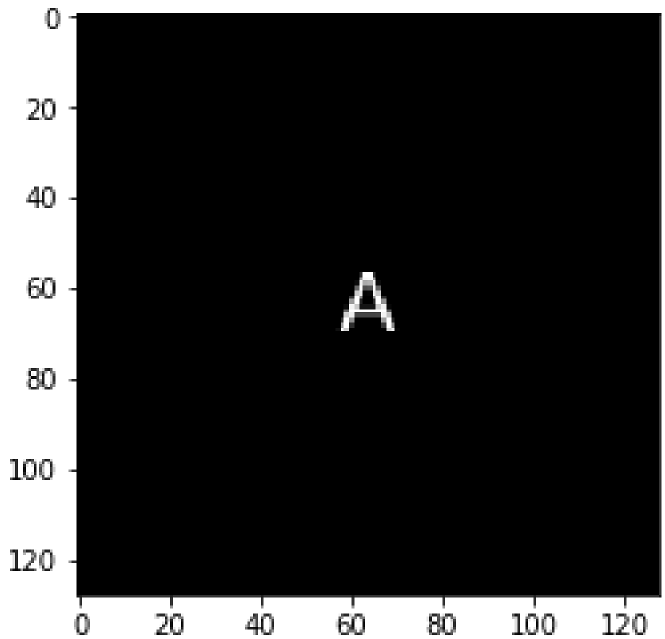


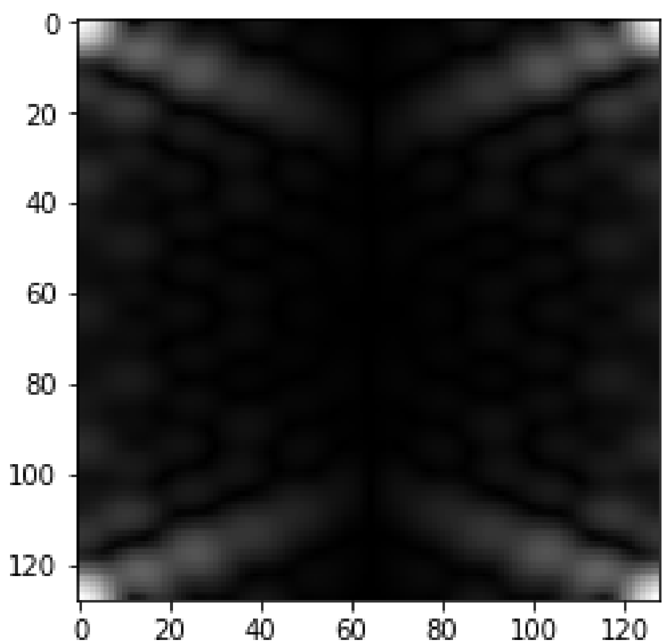


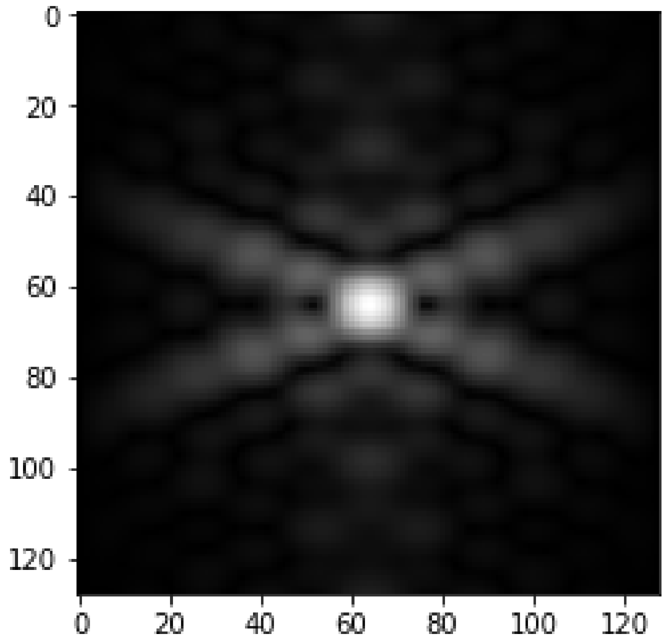
Out[5]: 'As seen, the expected result for the circle can be seen: the shifted 2D FFT of the ci

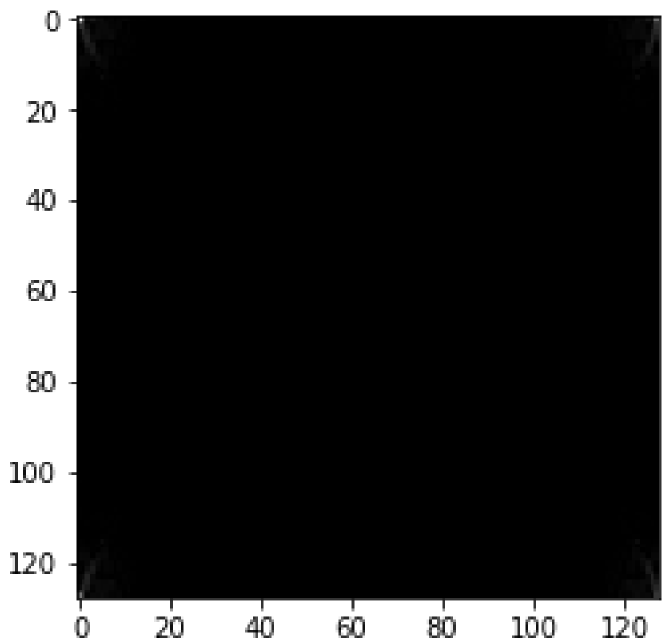
0.2.2 Letter A Fourier Transforms

In [6]:
```
A = rgb2gray(plt.imread('A.png'))
fourier(A)

""" The image of A used is not perfectly sharp. This gives rise to various frequencies
```





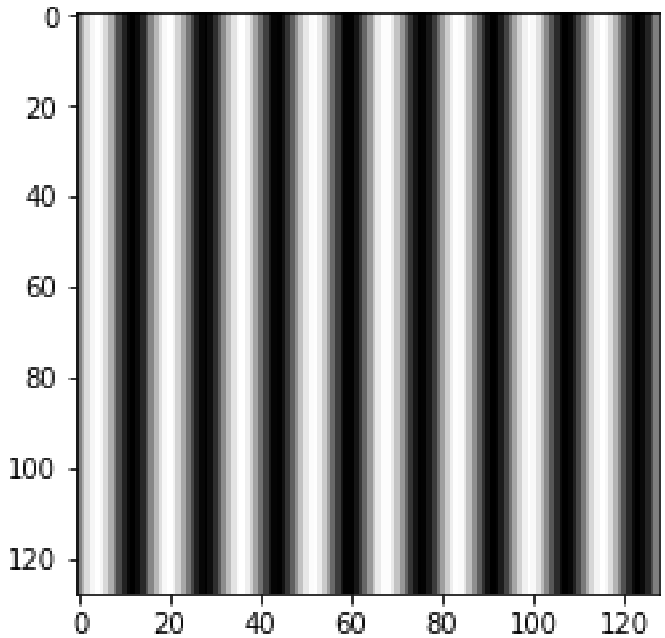


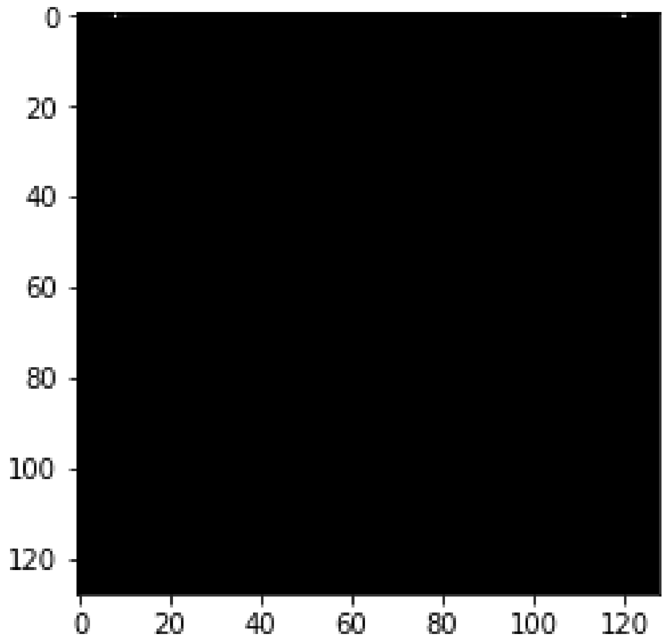


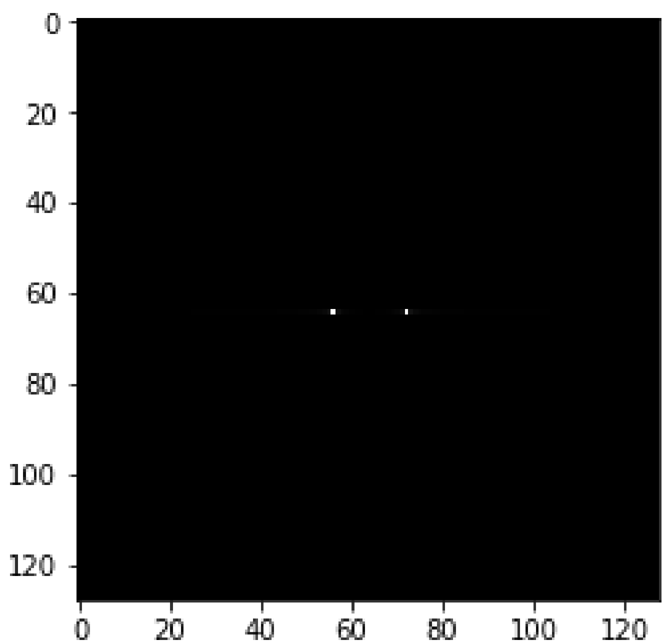

```
Out[6]: ' The image of A used is not perfectly sharp. This gives rise to various frequencies in
```

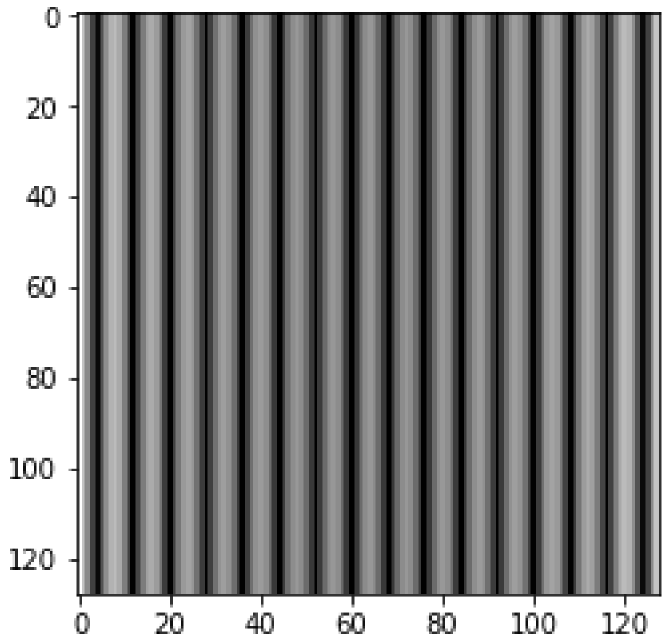
0.2.3 Corrugated Roof Fourier Transforms

```
In [7]: x = np.linspace(-1, 1, size)
        y = np.linspace(-1, 1, size)
        X,Y = np.meshgrid(x,y)
        f = 4 #frequency
        z = np.sin(2*np.pi*f*X)
        fourier(z)

        """Seen here is the FTs of a sinusoid in x. There are two dots in the FT modulus of th
```




0
20
40
60
80
100
120
0
20
40
60
80
100
120

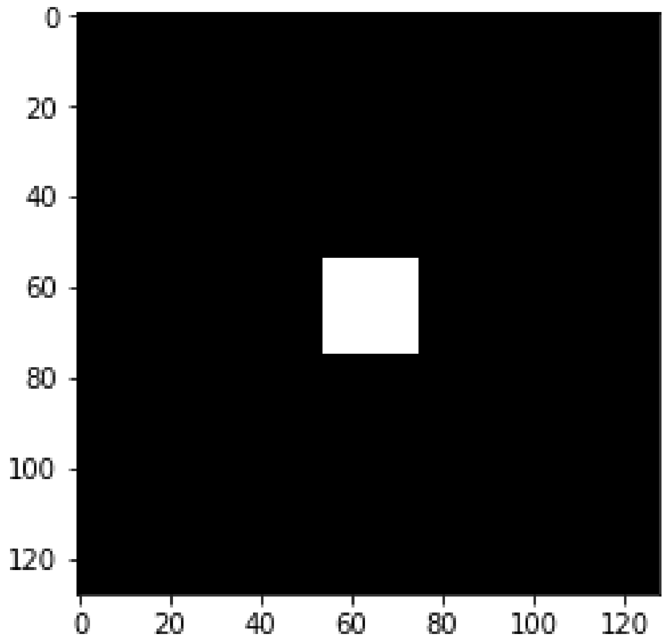

0
20
40
60
80
100
120
0
20
40
60
80
100
120

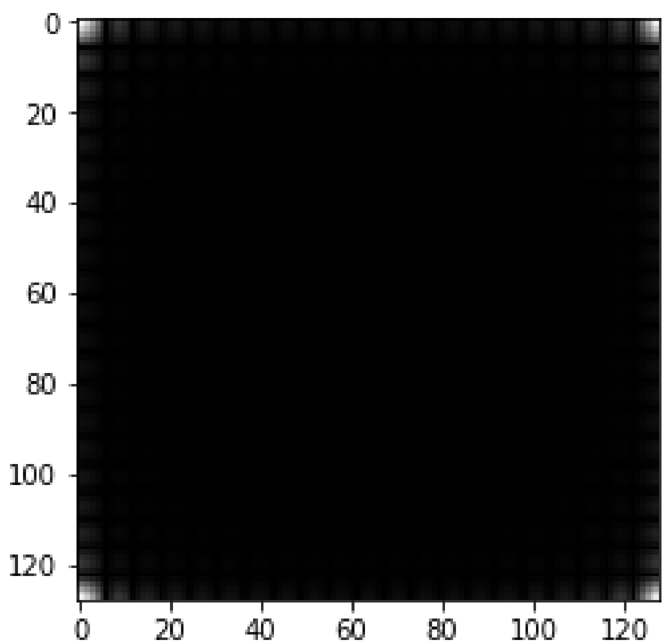


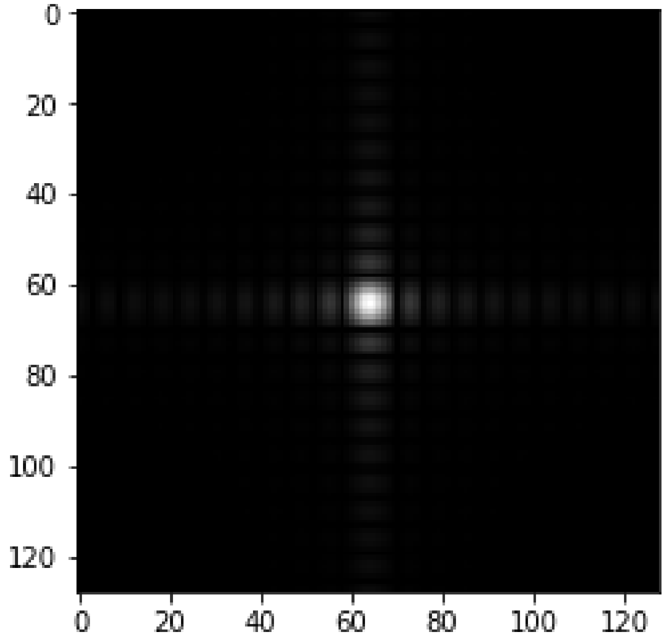

```
Out[7]: 'Seen here is the FTs of a sinusoid in x. There are two dots in the FT modulus of the i
```

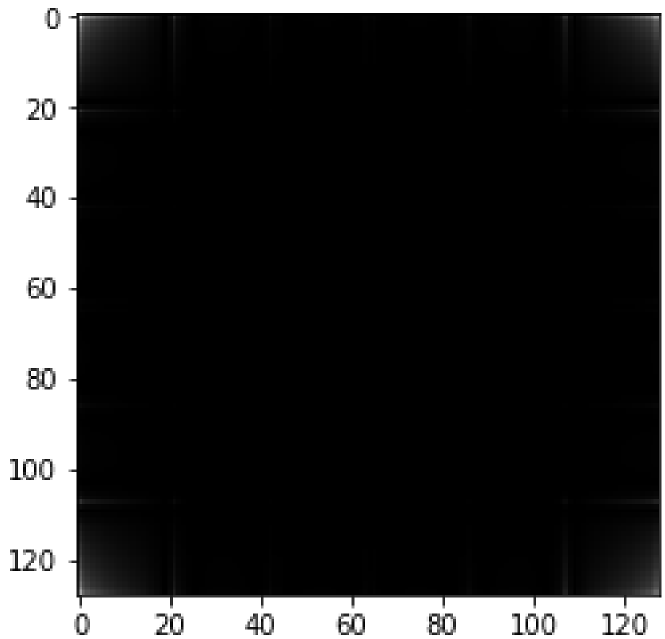
0.2.4 Square

```
In [8]: side = 20 #size of the square
        square = np.zeros([size,size])
        for i in range(0,size):
            for j in range(0,size):
                if ((i >= (size/2)-side//2 and i <= (size/2)+side//2) and (j >= (size/2)-side/
                    square[i,j] = 1
        fourier(square)

        """The square has frequency in both axes, which leads to the FT modulus seen here: band
```


0
20
40
60
80
100
120
0
20
40
60
80
100
120


0
20
40
60
80
100
120
0
20
40
60
80
100
120






```
Out[8]: 'The square has frequency in both axes, which leads to the FT modulus seen here: bands
```

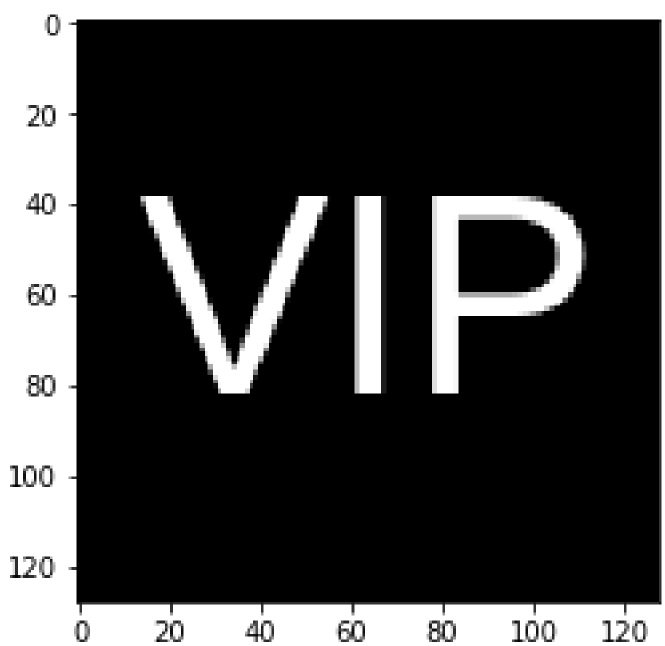
0.3 Simulation of an imaging device

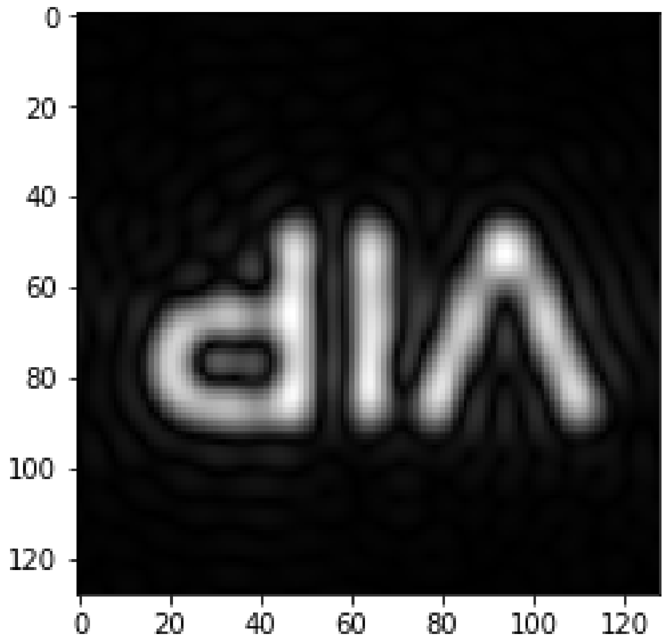
0.3.1 User defined Convolution function

```
In [9]: def convolution(data, aperture):
            plt.imshow(data, cmap = "gray")
            plt.show()
            datafft = np.fft.fft2(data)
            lensshift = np.fft.fftshift(aperture)
            FRA = lensshift * datafft
            IRA = np.fft.fft2(FRA)
            Image = abs(IRA)
            plt.imshow(Image, cmap = "gray")
            plt.show()
```

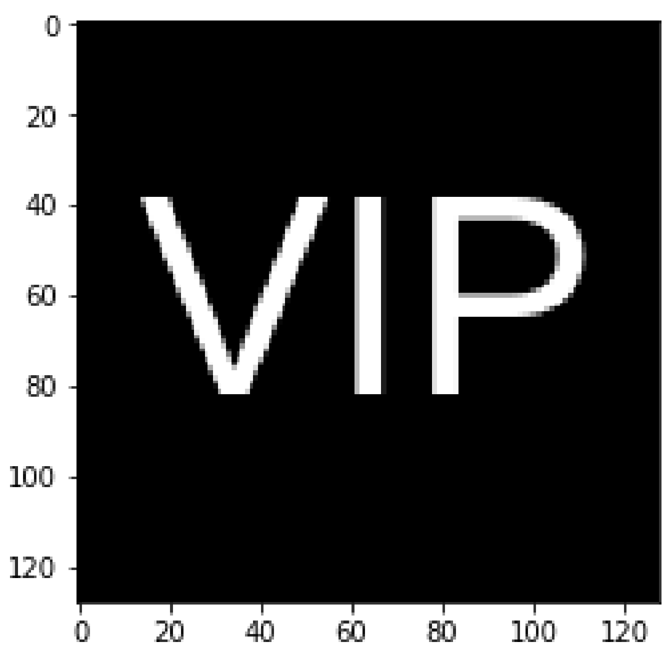
```
In [10]: vip = rgb2gray(plt.imread("VIP.png"))
         lens = rgb2gray(plt.imread('circle.png'))
         lensbig = rgb2gray(plt.imread('circlebig.png'))
```

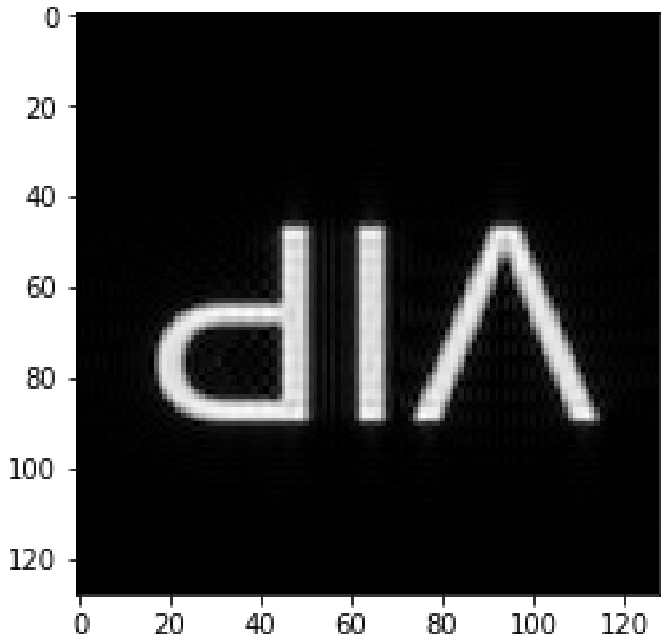
```
In [11]: convolution(vip, lens)
```





In [12]: convolution(vip, lensbig)





Convolving an image with an aperture results in the image flipping along the horizontal. Furthermore, a smaller aperture leads to a more blurry image as compared to a large aperture.

0.4 Template matching using Correlation

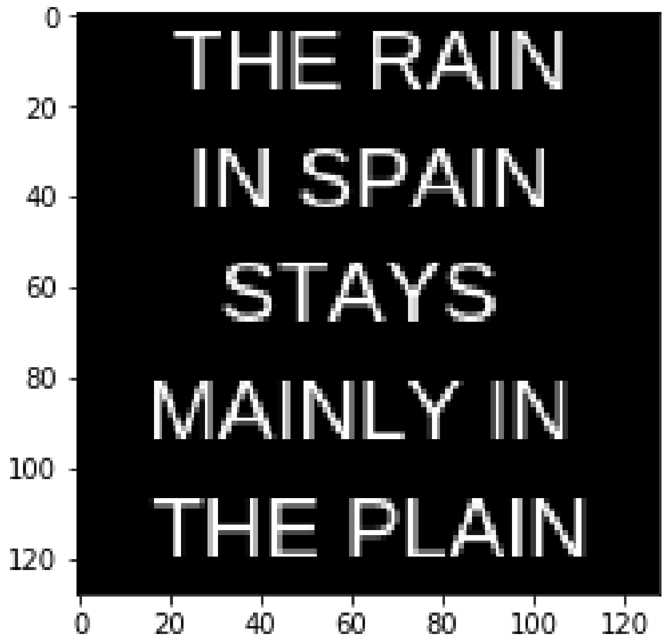
In [13]:
```
sentence = rgb2gray(plt.imread("sentence.png"))
A = rgb2gray(plt.imread('A.png'))
```

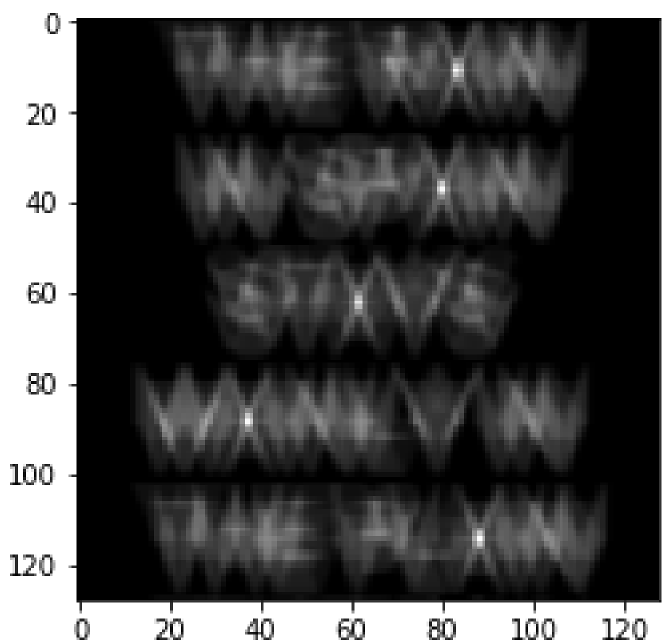
In [14]:
```
sentencefft = np.fft.fft2(sentence)
Afft = np.fft.fft2(A)
sentenceconj = np.conj(sentencefft) #conjugate of the FT of the image
```

In [15]:
```
Image = np.zeros([128,128])
Image = np.fft.fft2(Afft * sentenceconj)
```

In [16]:
```
Imageinv = abs(np.fft.fftshift(Image))
```

In [17]:
```
plt.imshow(sentence, cmap = "gray")
plt.show()
plt.imshow(Imageinv, cmap = "gray")
plt.show()
```



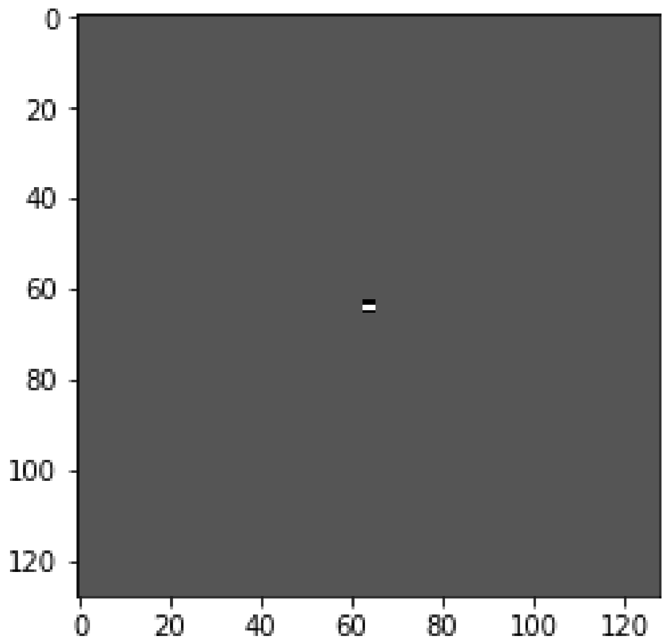


The output seen above shows that there are peaks at letter A, the same pattern used in the correlation.

0.5 Edge Detection using the convolution integral

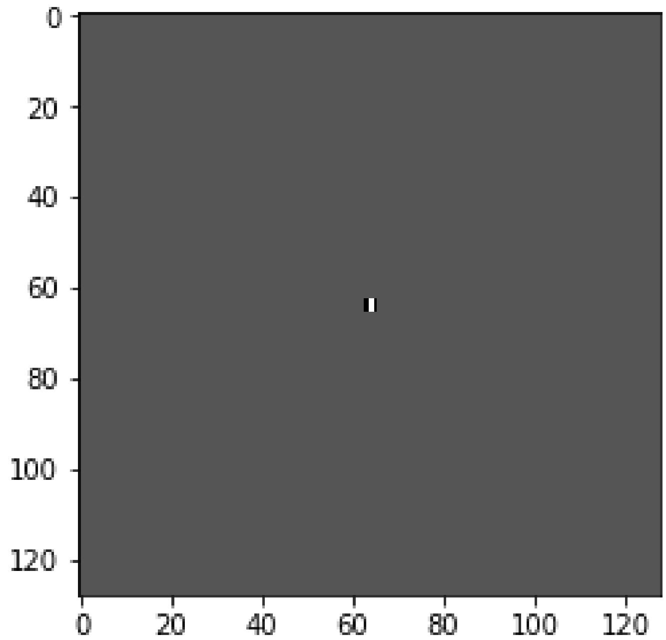
In [18]:
```
patternhori = np.zeros([size,size])
patternhori[size//2-1:size//2+2,size//2-1:size//2+2] = -1
patternhori[size//2,size//2-1:size//2+2] = 2
plt.imshow(patternhori, cmap="gray")
```

Out[18]: <matplotlib.image.AxesImage at 0x7f787afa8c88>



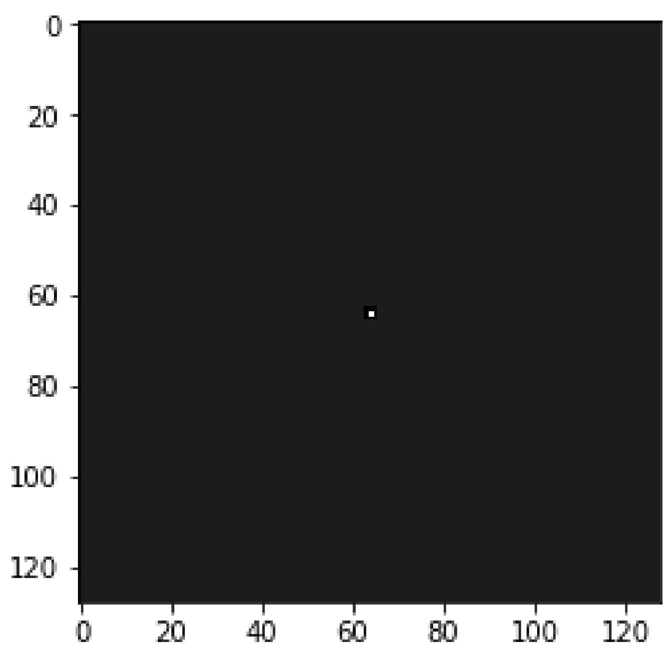
In [19]:
```
patternvert = np.zeros([size,size])
patternvert[size//2-1:size//2+2,size//2-1:size//2+2] = -1
patternvert[size//2-1:size//2+2,size//2] = 2
plt.imshow(patternvert, cmap="gray")
```

Out[19]: <matplotlib.image.AxesImage at 0x7f787aef4ef0>

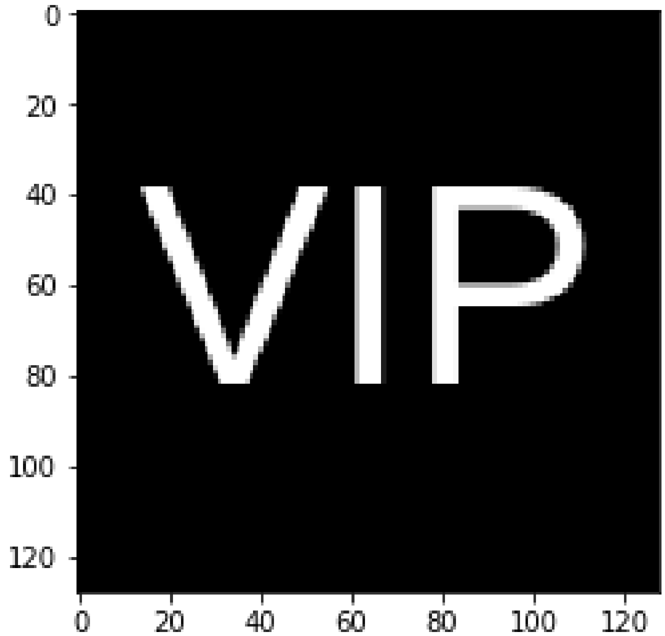


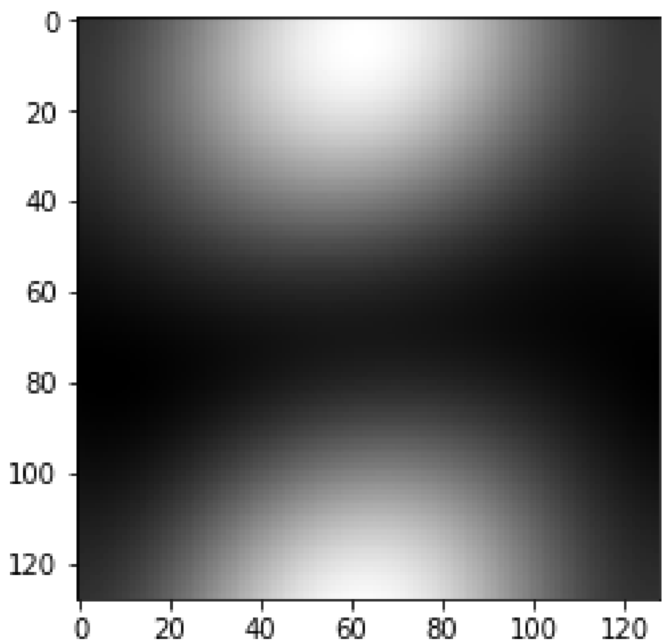
```
In [20]: patternspot = np.zeros([size,size])
         patternspot[size//2-1:size//2+2,size//2-1:size//2+2] = -1
         patternspot[size//2,size//2] = 8
         plt.imshow(patternspot, cmap="gray")
```

```
Out[20]: <matplotlib.image.AxesImage at 0x7f7878d46c50>
```

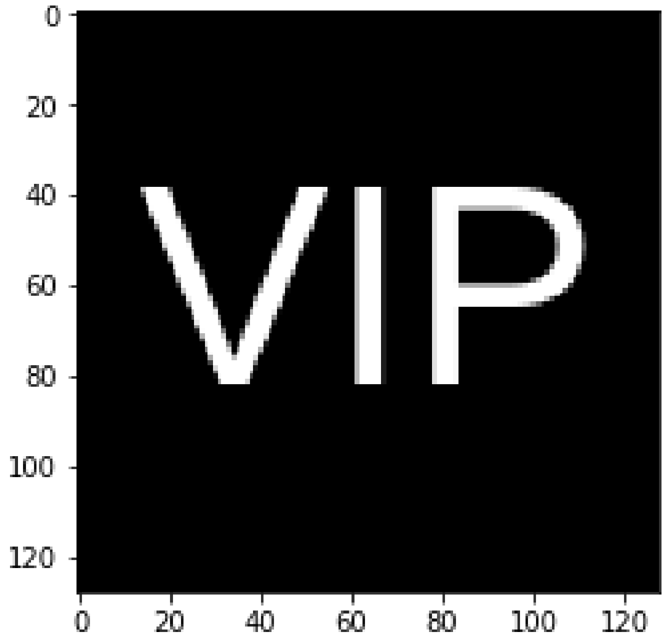


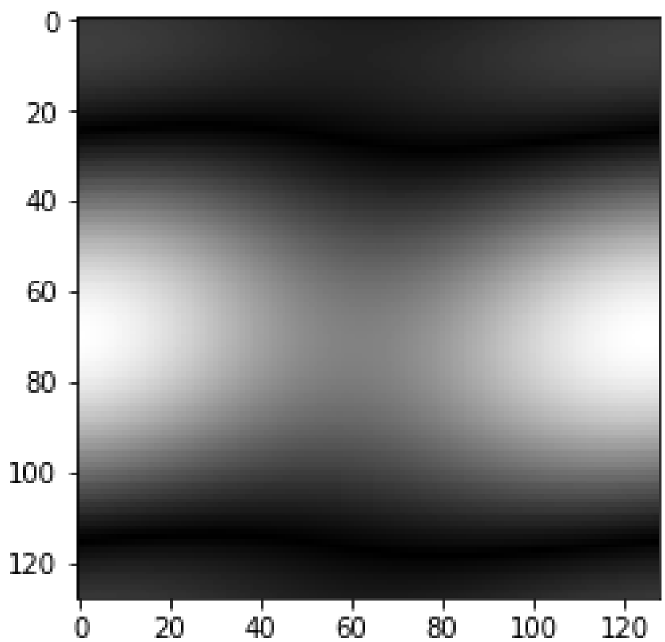
```
In [21]: convolution(vip, patternhori)
```



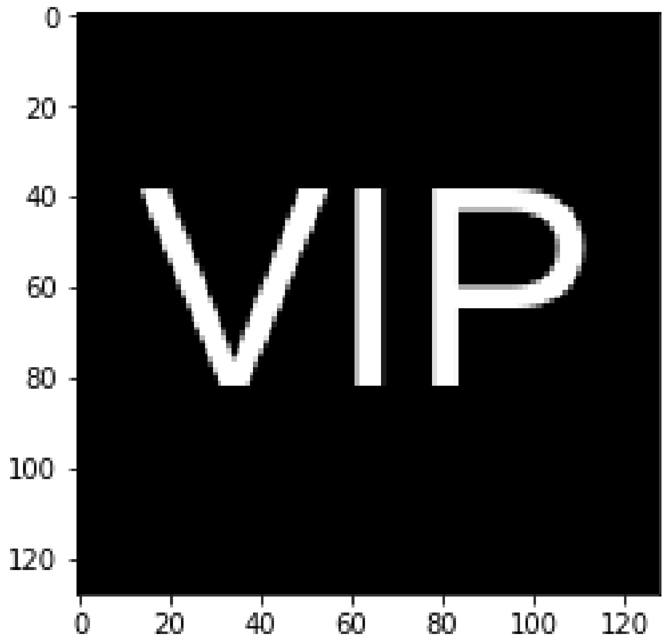


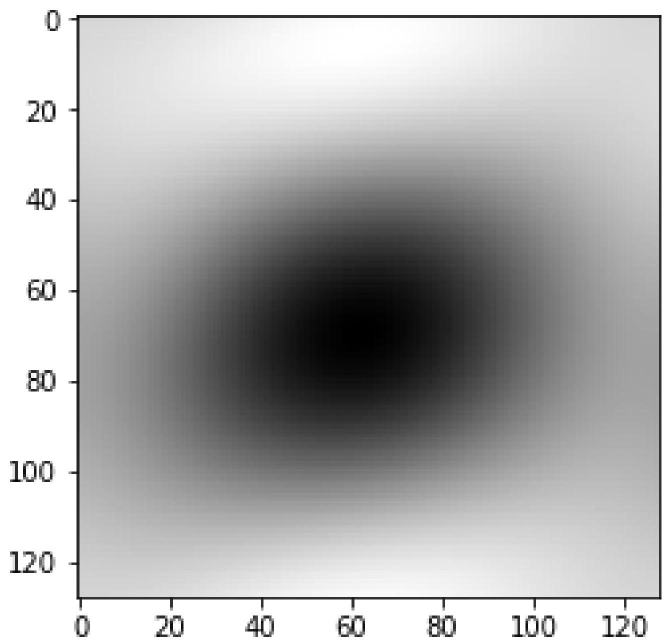
```
In [22]: convolution(vip, patternvert)
```






```
In [23]: convolution(vip, patternspot)
```





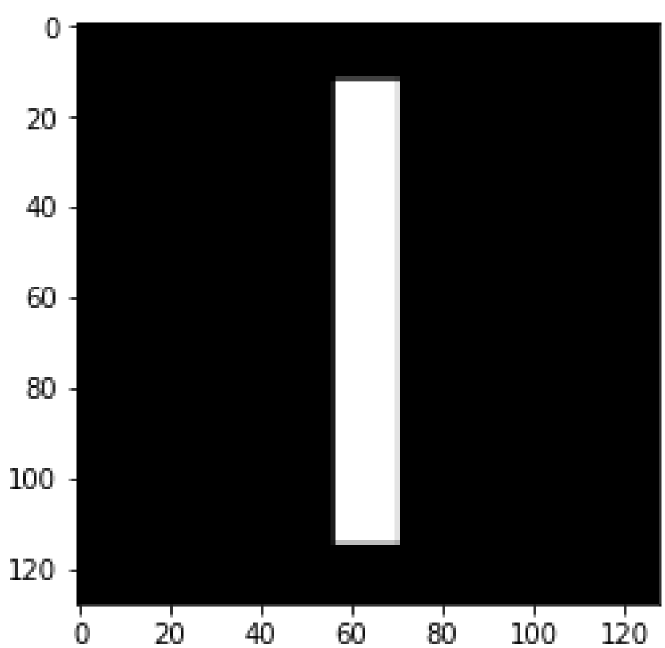
I was not able to successfully complete this section because the created patterns were not found in the VIP image. This could be attributed to the rudimentary method used in creating the patterns. They might not be in the exact center.

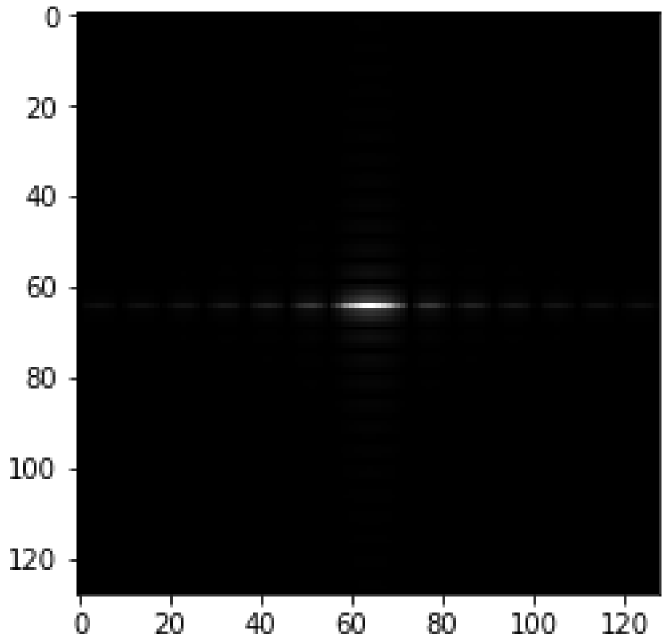
0.6 Anamorphic Property of FT of different 2D patterns

In [24]:
```
#User defined anamorphism function
def anamorphism(data):
    plt.imshow(data, cmap = 'gray')
    plt.show()
    datafft = np.fft.fft2(data)
    datashift = abs(np.fft.fftshift(datafft))
    plt.imshow(datashift, cmap = 'gray')
    plt.show()
```

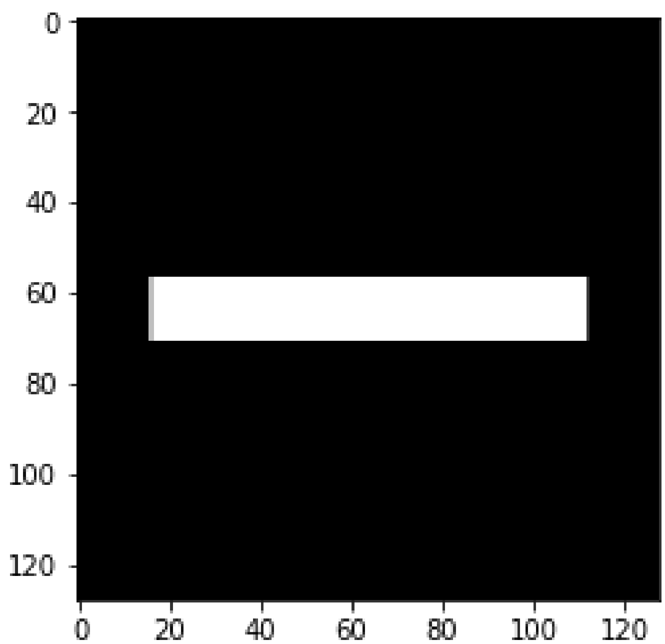
In [25]:
```
tall = rgb2gray(plt.imread('Tall Rectangle.png'))
wide = rgb2gray(plt.imread('Wide Rectangle.png'))
dotsclose = np.zeros([size,size])
dotsfar = np.zeros([size,size])

#Dots image constructor
for i in range(0,size):
    for j in range(0,size):
        if i == size/2:
            if j == (size/2-5) or j == (size/2+5):
                dotsclose[i,j] = 1
            elif j == (size/2-10) or j == (size/2+10):
                dotsfar[i,j] = 1
```

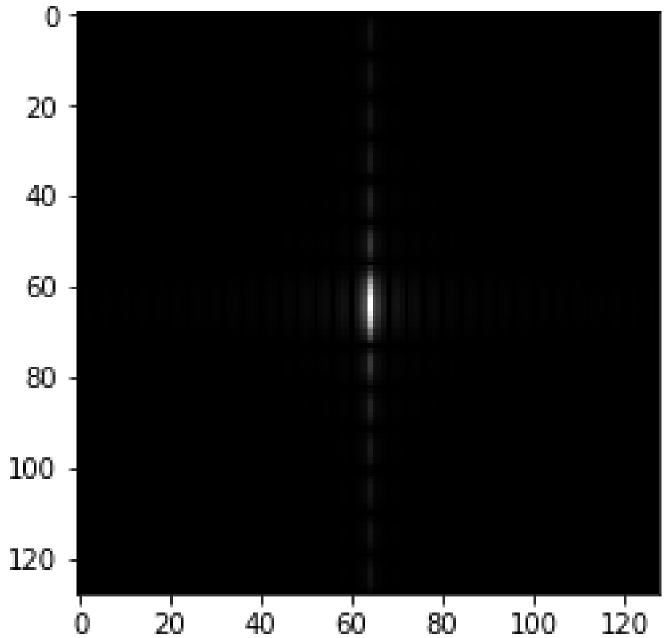
In [26]:
```
anamorphism(tall)

#The FT of this rectangle is wide at the horizontal axis, which is where this rectang
```



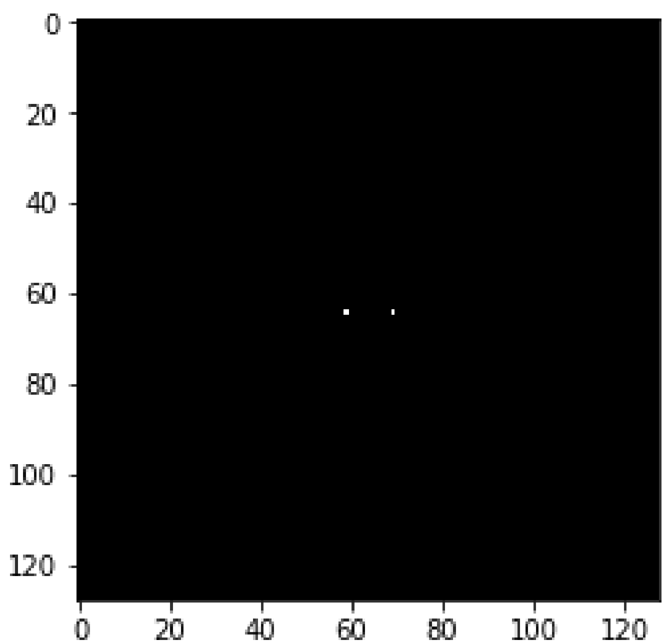


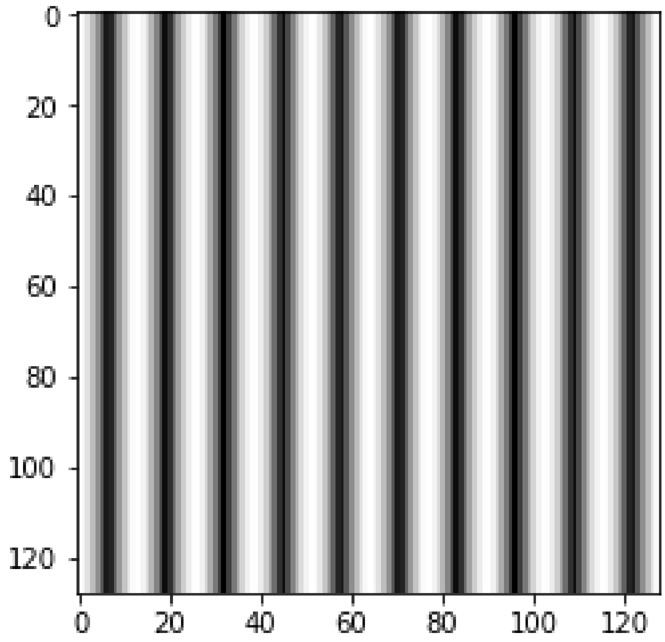

```
In [27]: anamorphism(wide)
         #The FT of this triangle is wide at the vertical axis, which is where this rectangle
```





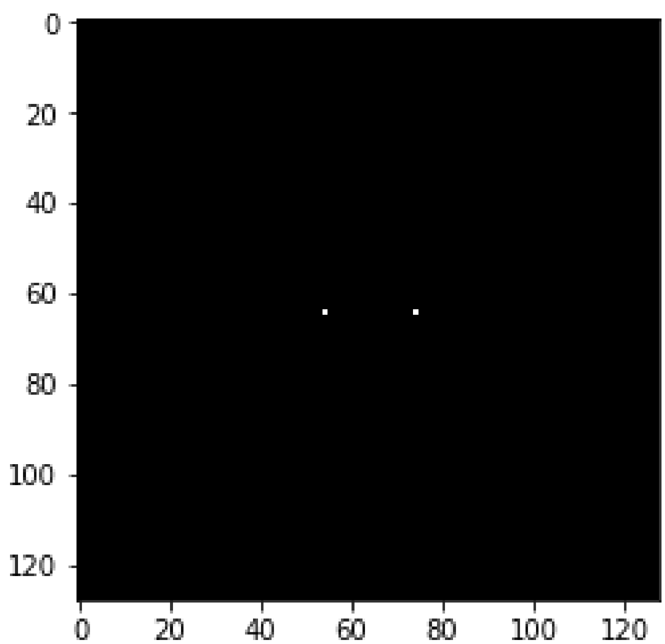
```
In [28]: anamorphism(dotsclose)

         #As seen in a previous section, the FT of a sinusoid is two dots. Thus when the FT of
```

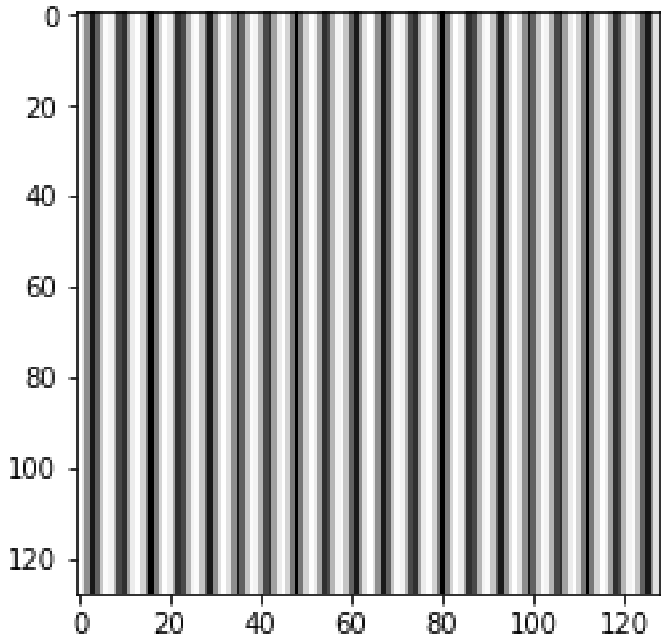




In [29]: anamorphism(dotsfar)

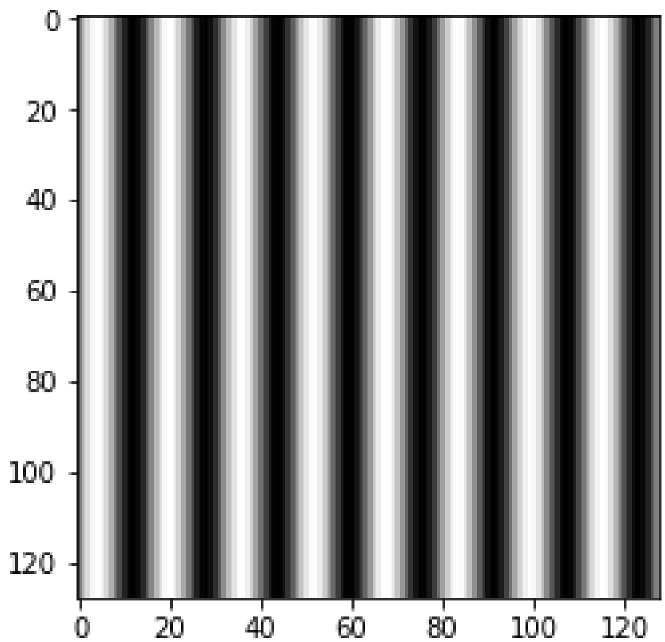
```
#The FT sinusoid pattern of two dots that are farther from one another has a higher f
```

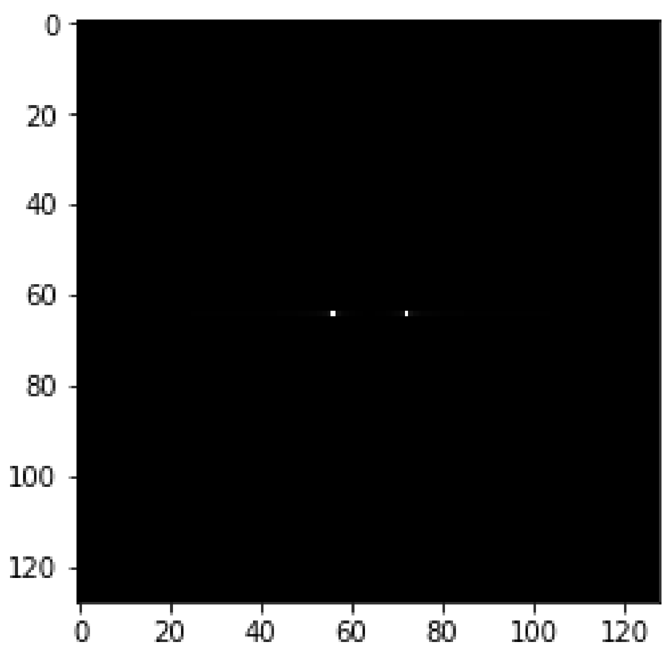




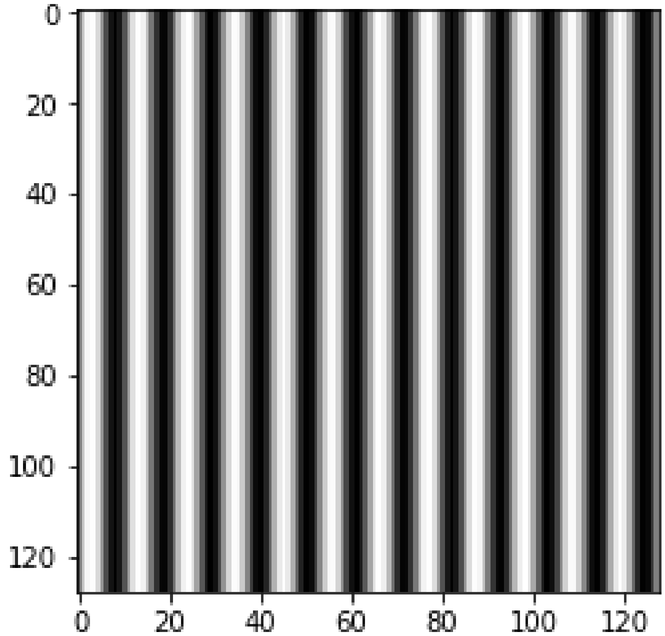
0.7 Rotation Property of the FT

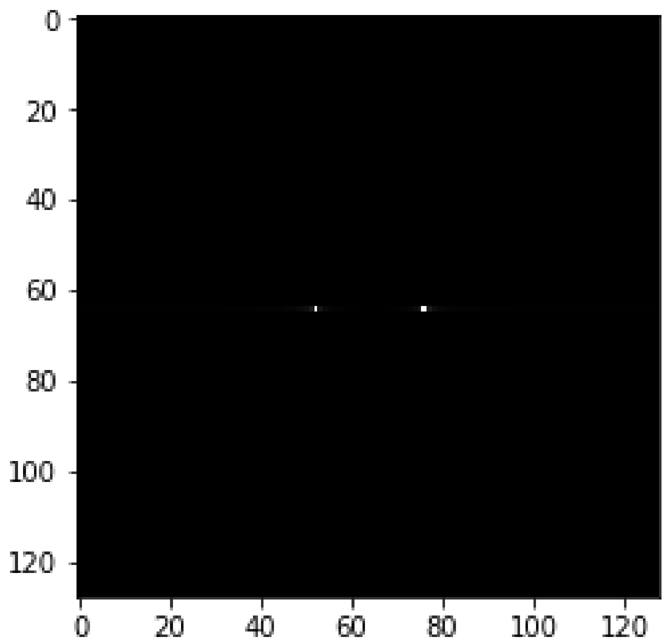
```
In [30]: x = np.linspace(-1, 1, size)
         y = np.linspace(-1, 1, size)
         X,Y = np.meshgrid(x,y)

In [31]: f = 4
         z = np.sin(2*np.pi*f*X)
         anamorphism(z)
         #Once again, the FT of a sinusoid is two dots.
```



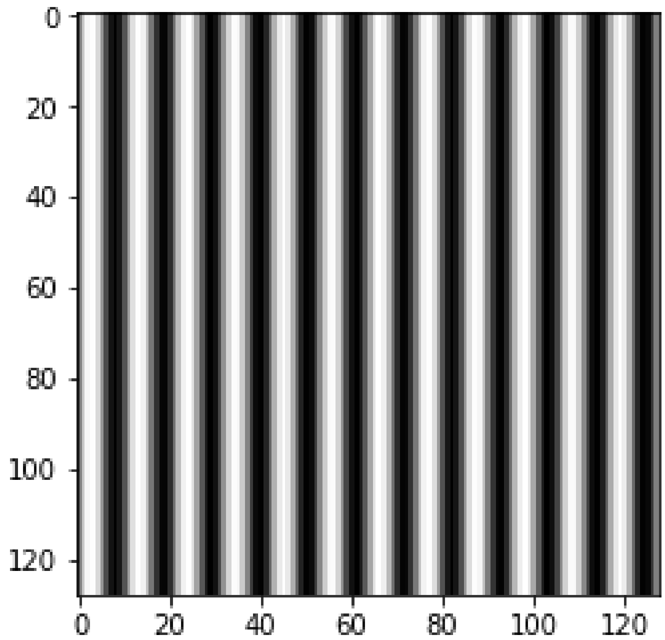


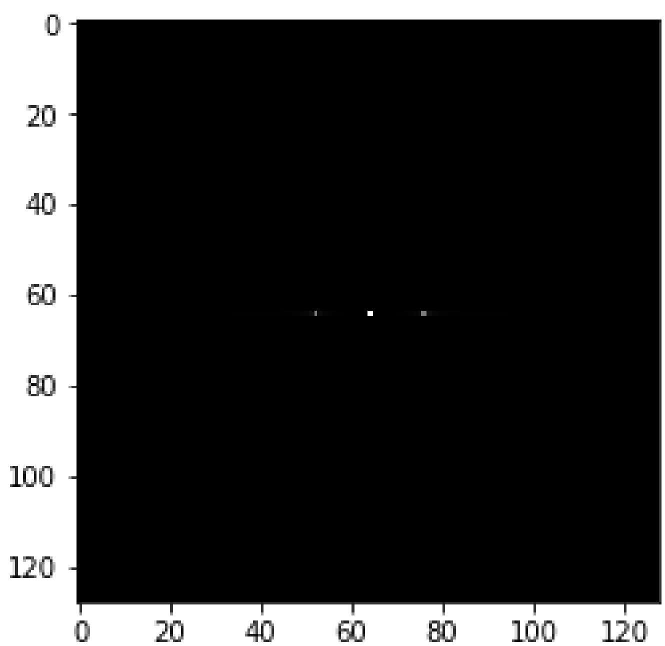
```
In [32]: f = 6
         z = np.sin(2*np.pi*f*X)
         anamorphism(z)
         #Increasing the frequency of the sinusoid increased the distance between the two dots
```



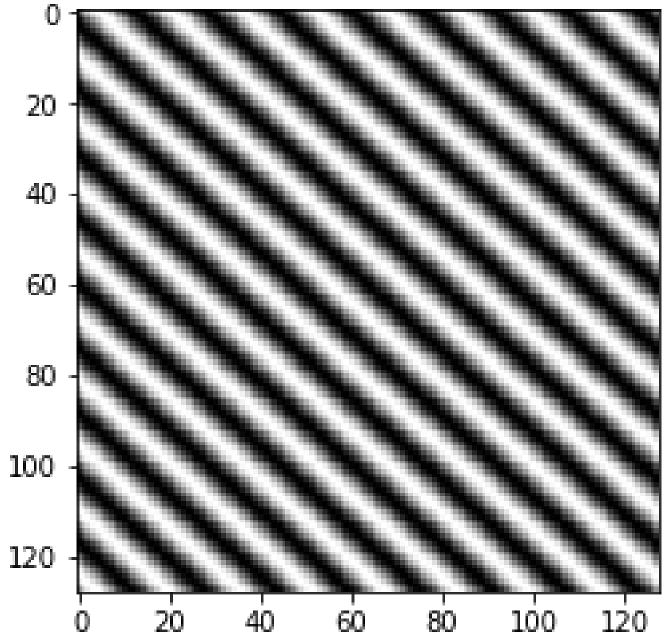


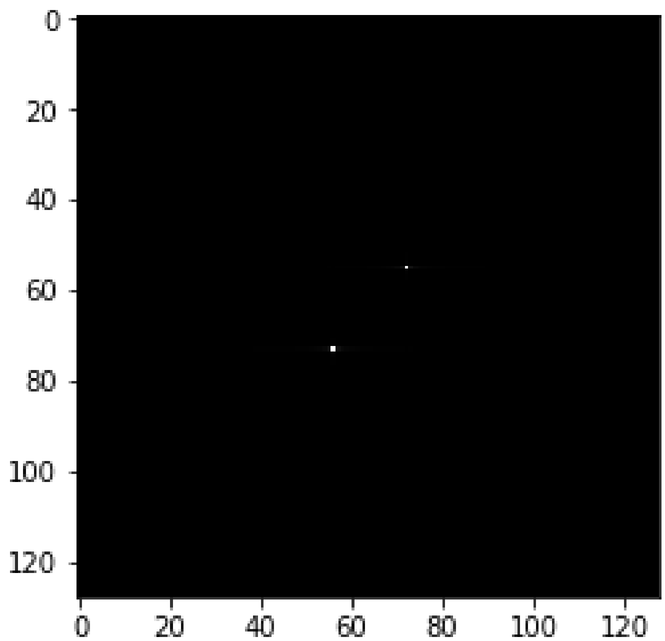
```
In [33]: z = np.sin(2*np.pi*f*X)+1
         anamorphism(z)
         #Adding a constant bias to the sinusoid produced another dot in the FT, in the middle
```



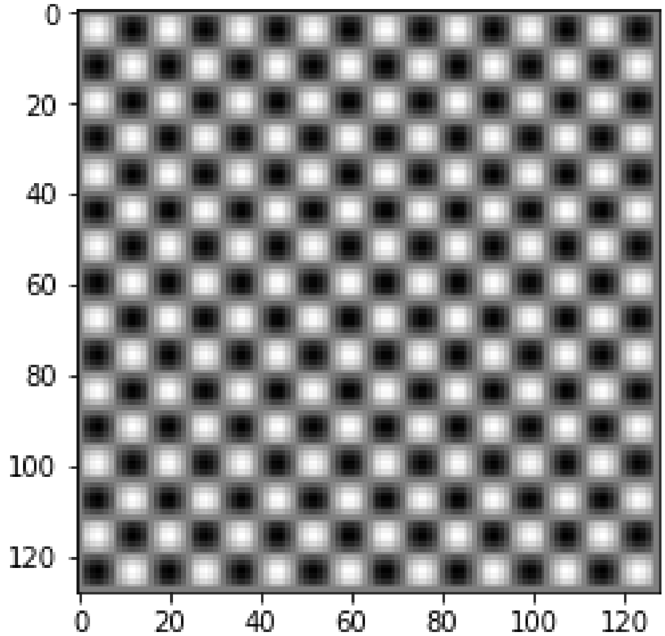


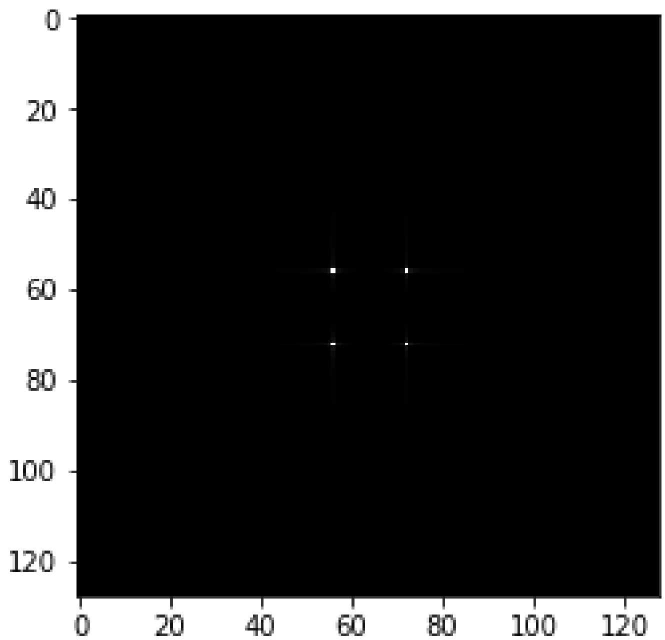

```
In [34]: theta = 40
         z = np.sin(2*np.pi*f*(Y*np.sin(theta) + X*np.cos(theta)))
         anamorphism(z)
         #Rotating the sinusoid also rotated the two dots at the same angle.
```



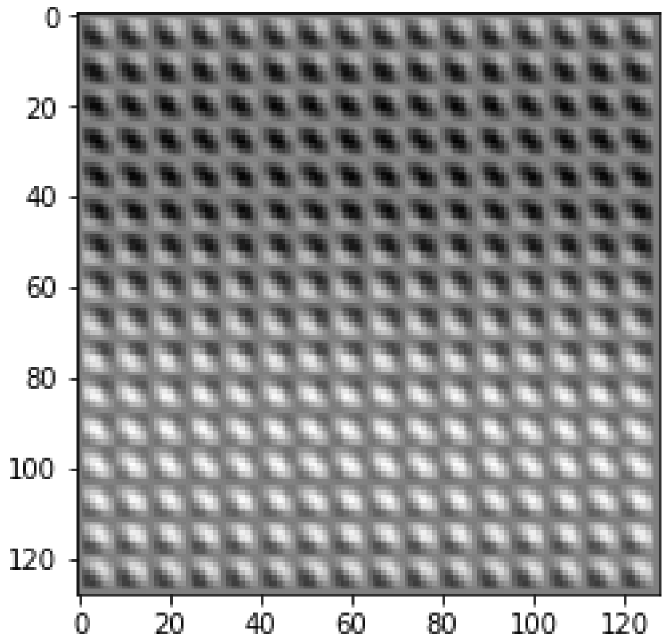


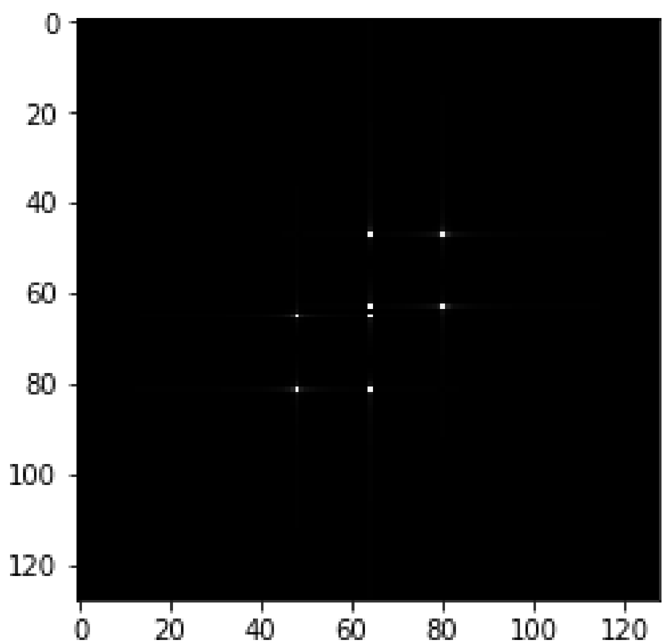
```
In [35]: z = np.sin(2*np.pi*4*X)*np.sin(2*np.pi*4*Y)
         anamorphism(z)
         #Adding two sinusoids together produced two more dots in the FT of the sinusoid produ
```





In [36]:
```
z = np.sin(2*np.pi*4*X)*np.sin(2*np.pi*4*Y)*np.sin(2*np.pi*f*(Y*np.sin(theta) + X*np.c
anamorphism(z)
#Taking into account the previous results, I predicted that adding a rotated sinusoid
#The result of the above process actually looks like the FT of the initial two sinuso
```


0
20
40
60
80
100
120
0
20
40
60
80
100
120

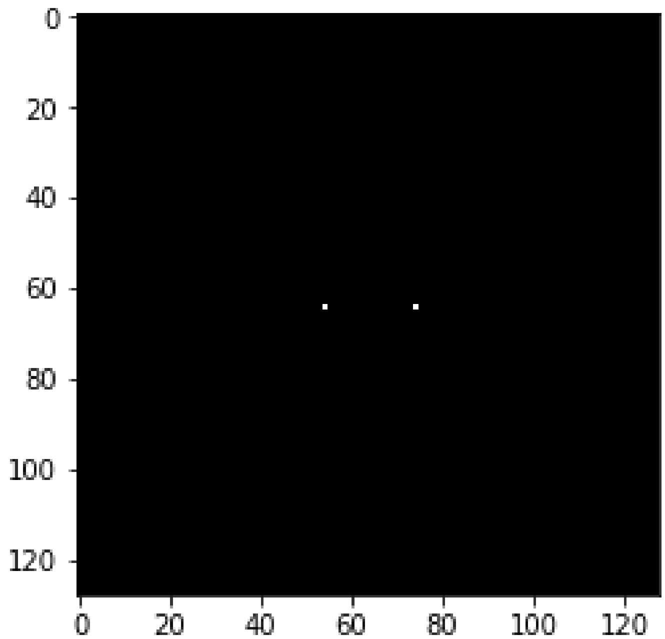

0
20
40
60
80
100
120
0
20
40
60
80
100
120

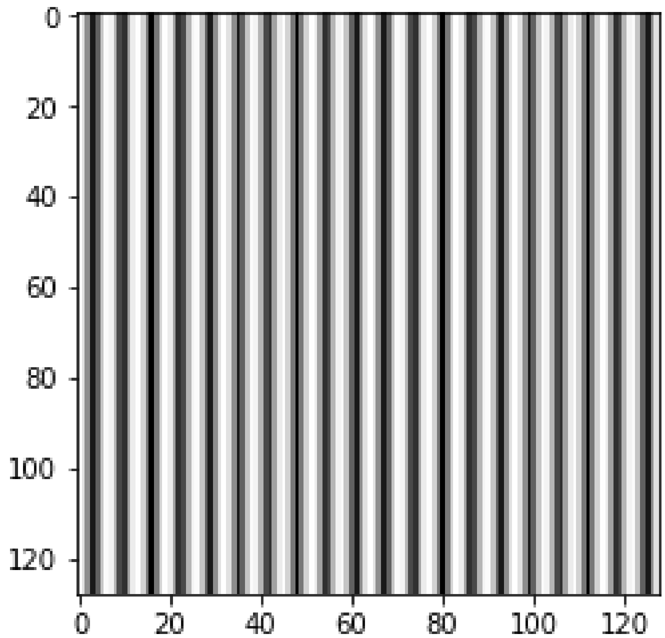
0.8 Convolution Theorem Redux

0.8.1 2 Dots

```
In [37]: Image = np.zeros([size,size])
         for i in range(0,size):
             for j in range(0,size):
                 if i == size/2:
                     if j == (size/2-10) or j == (size/2+10):
                         Image[i,j] = 1
```

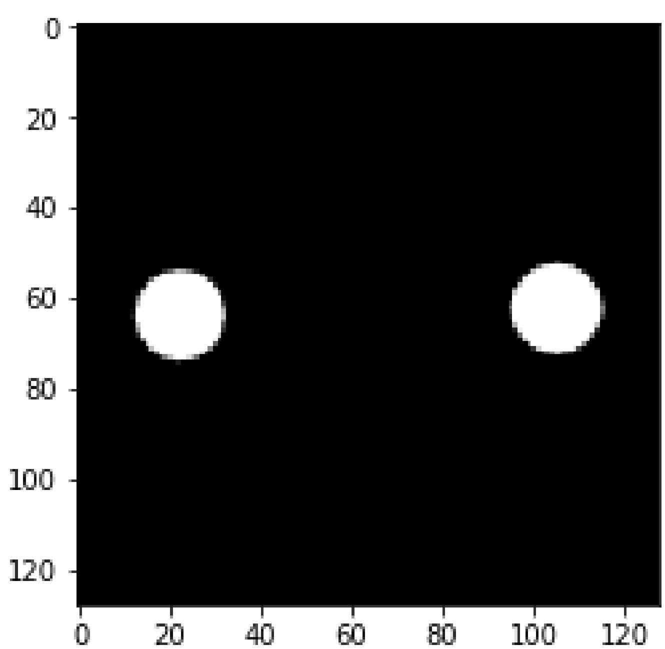
```
In [38]: anamorphism(Image) #FT modulus of two dots displayed: a sinusoid.
```

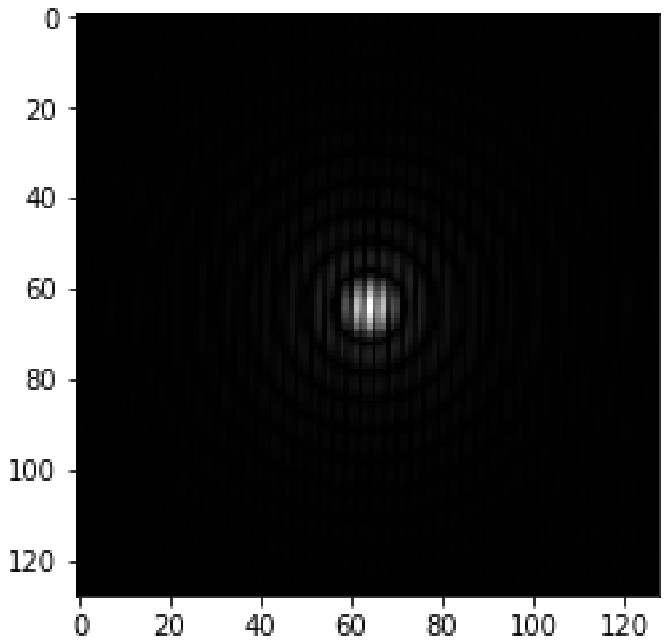




0.8.2 2 Circles

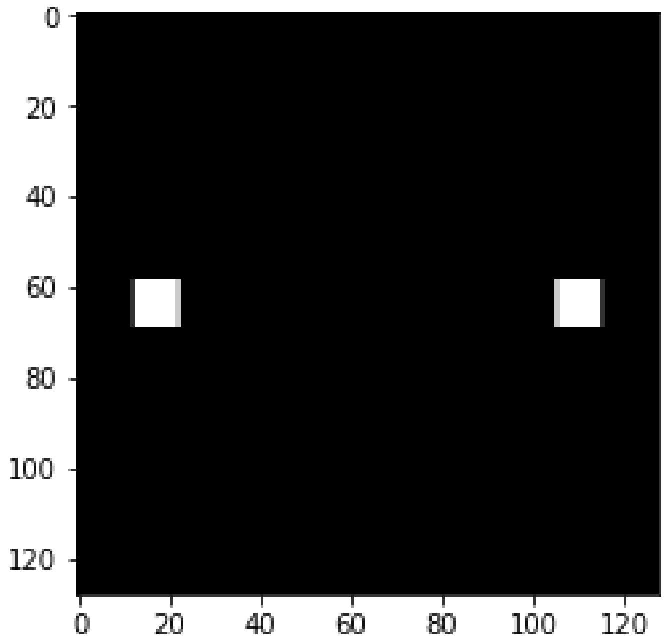
```
In [39]: circles = rgb2gray(plt.imread('Two dots.png'))
         anamorphism(circles)

         #FT modulus of circles displayed: an airy pattern cut by vertical lines
```

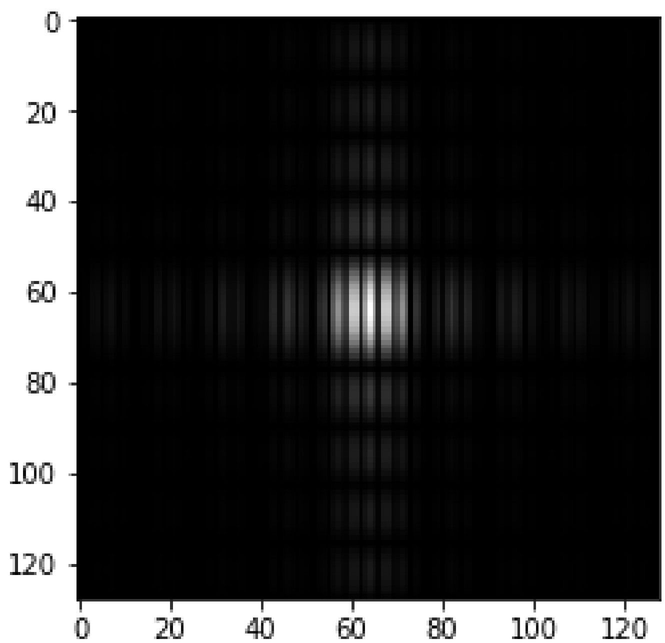




0.8.3 2 Squares

```
In [40]: squares = rgb2gray(plt.imread('2squares.png'))
         anamorphism(squares)

         #FT modulus of squares displayed: the FT of a square cut by vertical lines
```





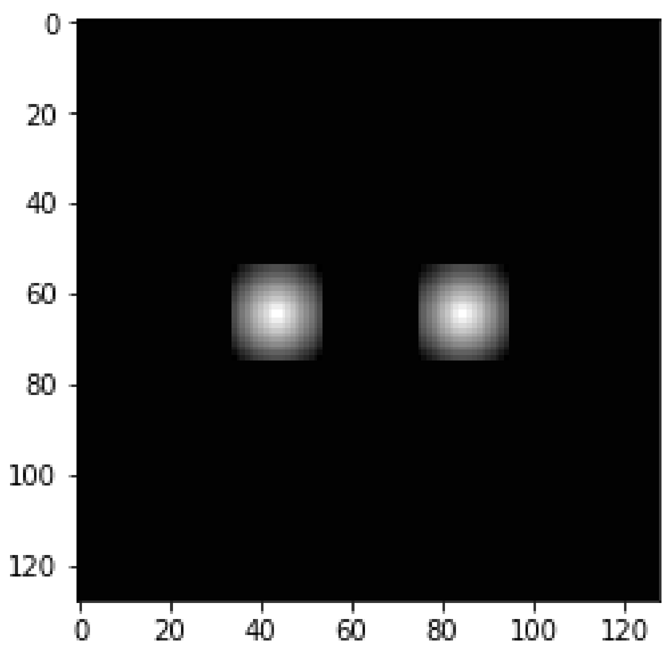
0.8.4 Gaussian

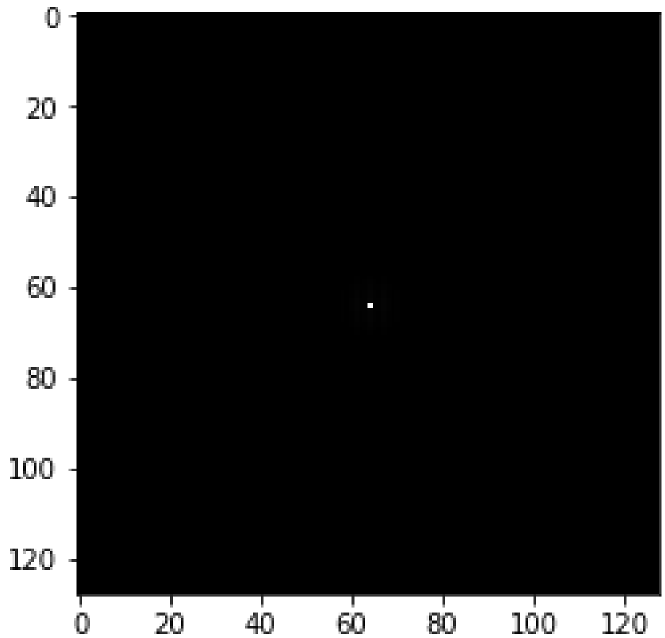
In [41]:
```
sigma = 2
mu = 0
```

```
amp = 1 / (sigma*np.sqrt(2*np.pi))
rad = 20
xg = np.linspace(-5,5,rad)
yg = np.linspace(-5,5,rad)
XG,YG = np.meshgrid(xg,yg)
r = np.sqrt(XG**2 + YG**2)
r = r*-1
```

In [42]:
```
gaus = np.ones([size,size])
gaus = gaus*-7
row = -1
column = 0

#adds equidistant gaussian circles to the image
for i in range(0,size):
    for j in range(0,size):
        if ((i >= size/2 - rad/2) and (i <= size/2 + rad/2)):
            if (j >= size/2 - 30 and j < size/2 - 10) or (j > size/2 + 10 and j <= si
                gaus[i,j] = r[row,column]
                column += 1
            else:
                column = 0
    if (i >= size/2 - 10 and i <= size/2 + 10):
        row += 1
```

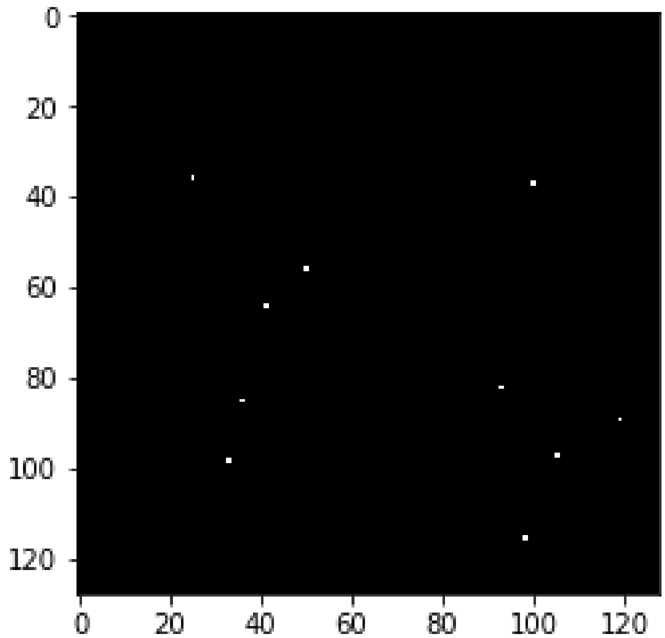
In [43]:
```
anamorphism(gaus) #Result: a single dot. I attribute this to the error of my method o
```

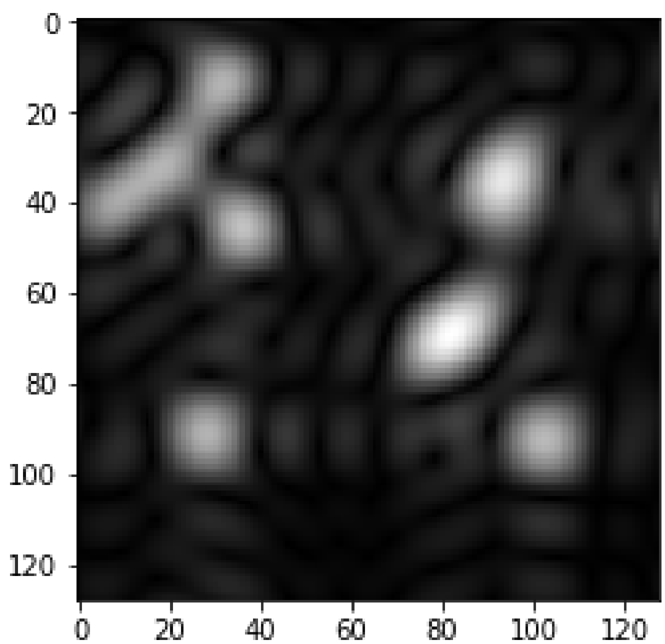




0.8.5 Dirac Delta

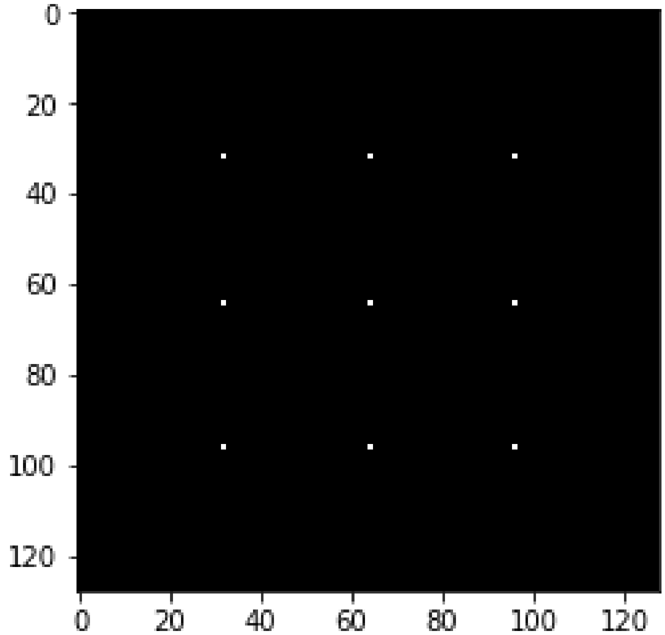
```
In [44]: A = np.zeros([size,size])

         #Sets up an image with 10 random dots
         for i in range(0,10):
             x = np.random.randint(0,size)
             y = np.random.randint(0,size)
             while(True):
                 if A[x,y] == 0:
                     A[x,y] = 1
                     break
                 else:
                     continue

         d = np.zeros([size,size])
         d[size//2-5:size//2+5,size//2-5:size//2+5] = 1

In [45]: convolution(A, d)
         #The image A is convolved with a vertical line, where the convolution result is shown
```

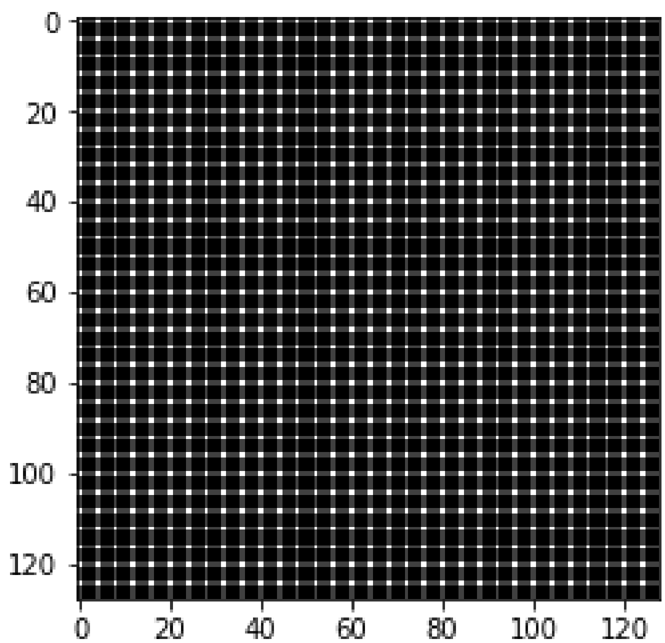




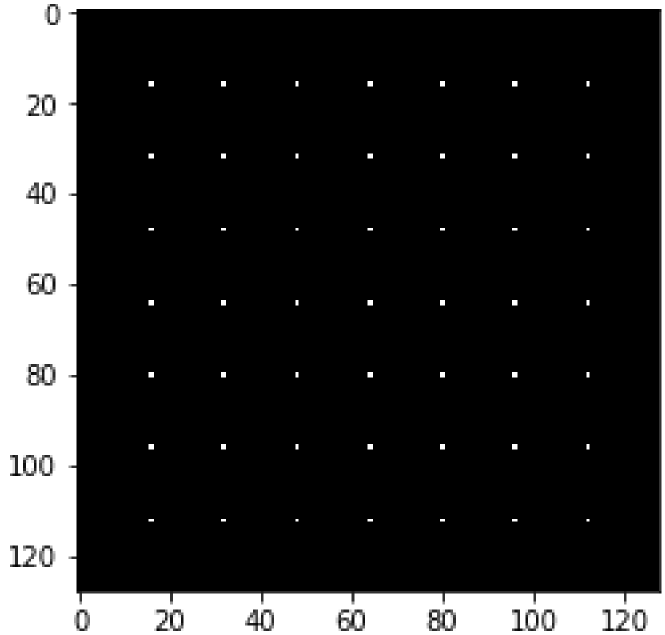

```
In [46]: d = np.zeros([size,size])
         e = size//4
         for i in range(1,4):
```

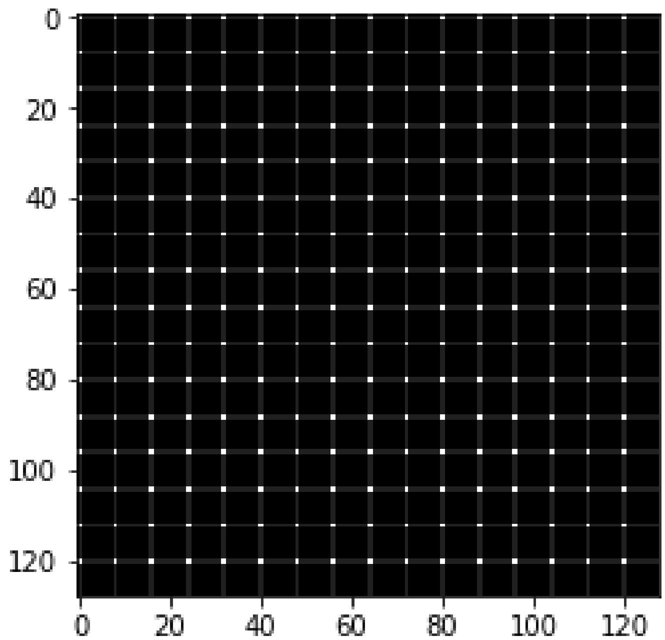
```
    d[e*i,[e*1,e*2,e*3]] = 1
anamorphism(d)

#Dots equally spaced with each other, less spacing of the dots in the FT
```





In [47]:
```
d = np.zeros([size,size])
e = size//8
for i in range(1,8):
    d[e*i,[e*1,e*2,e*3,e*4,e*5,e*6,e*7]] = 1
anamorphism(d)

#Dots equally spaced with each other with less spacing, more spacing of the dots in th
```

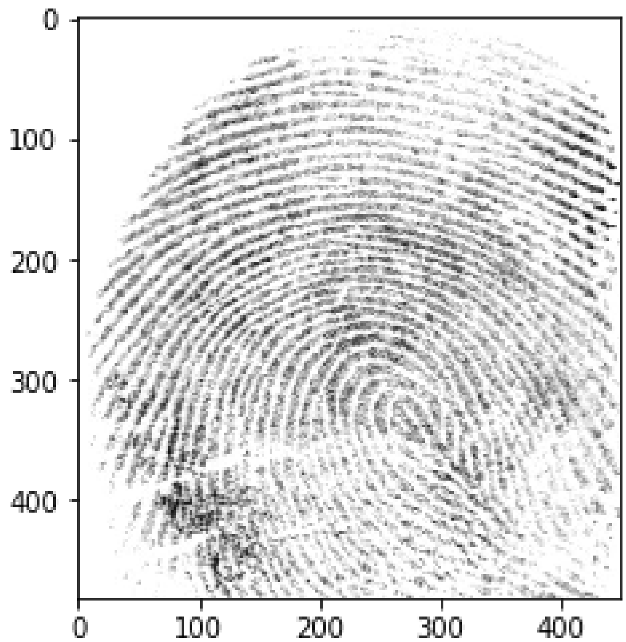




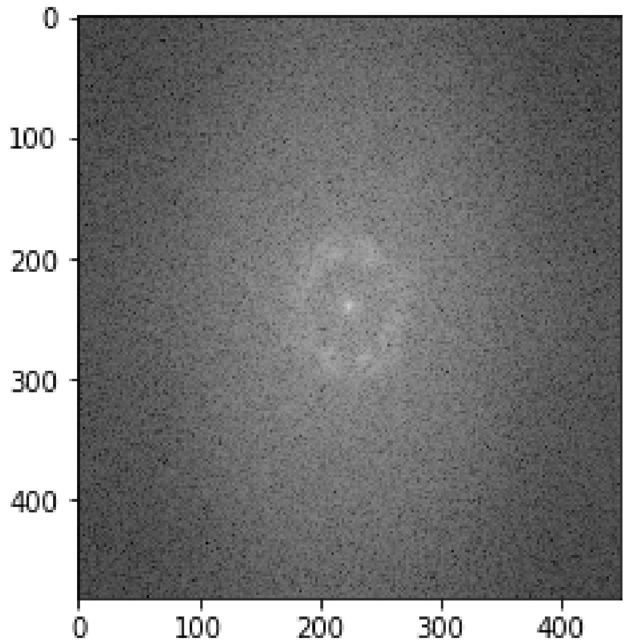
0.9 Fingerprints : Ridge Enhancement

In [48]:
```
#Fingerprint image lifted from: http://hristoalexiev.blogspot.com/p/computer-vision.h
finger = rgb2gray(plt.imread('Raw Fingerprint.jpg'))
plt.imshow(finger, cmap="gray")
```

Out[48]:
```
<matplotlib.image.AxesImage at 0x7f78a6a1a668>
```

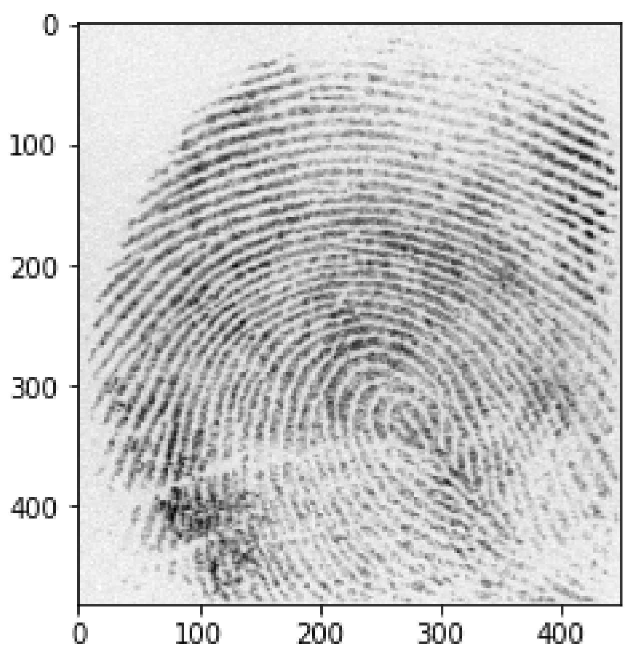


In [49]:
```
fingerfft = np.fft.fft2(finger)
fingershift = np.log(abs(np.fft.fftshift(fingerfft)))
plt.imshow(fingershift, cmap = "gray") #Frequencies in the fingerprint image. Fingerp
filterf = fingershift
filterf[np.where(fingershift<9)] = 0 #Masks the elements in fingershift less than 9

final = np.flip(np.abs(np.fft.fft2(np.fft.fft2(finger)*np.fft.fftshift(filterf)))) #M
```




```
In [50]: plt.imshow(final, cmap = "gray")
```

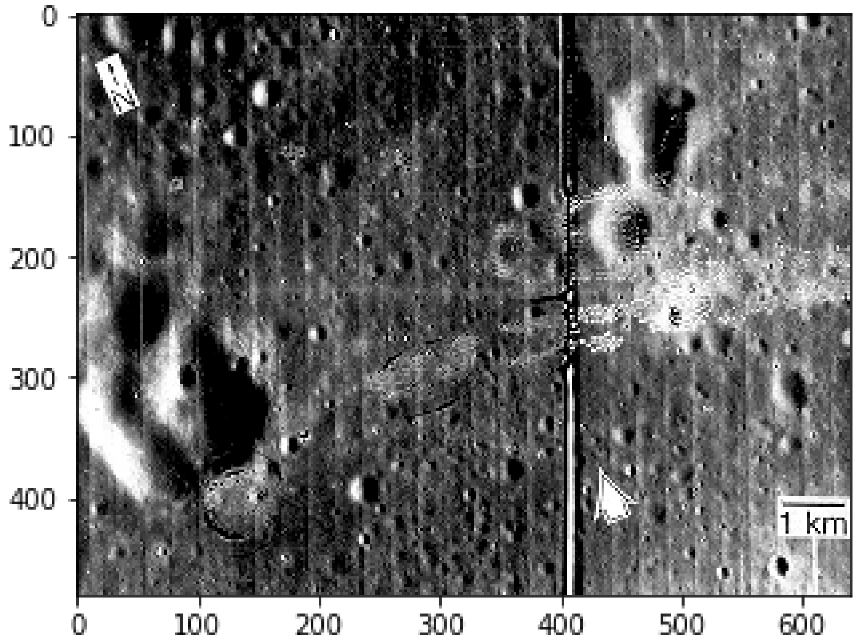
```
Out[50]: <matplotlib.image.AxesImage at 0x7f7878d26da0>
```



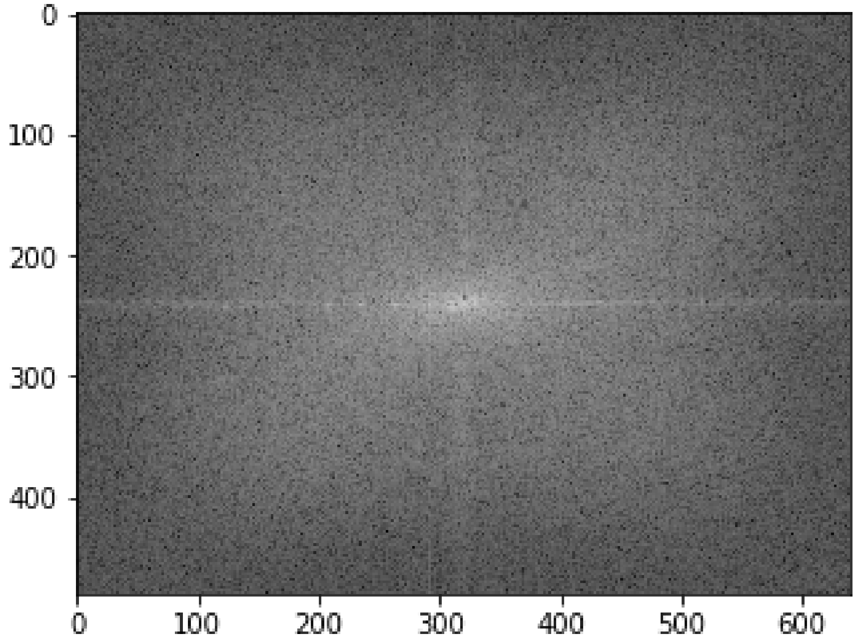
0.10 Lunar Landing Scanned Pictures : Line removal

In [51]:
```
lunar = rgb2gray(plt.imread('lunar.gif'))
plt.imshow(lunar, cmap="gray")
```

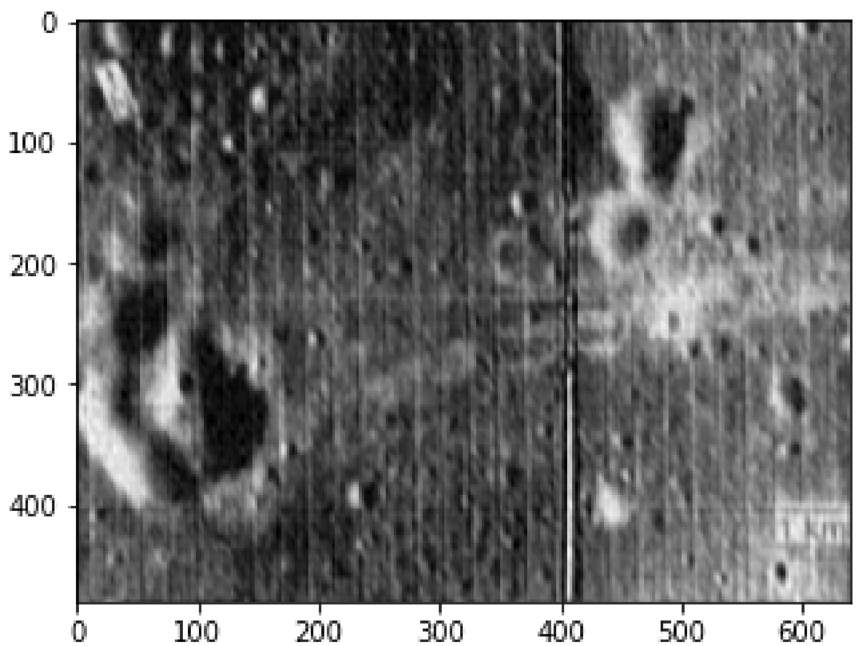
Out[51]: <matplotlib.image.AxesImage at 0x7f7878ccbf60>



In [52]:
```
lunarfft = np.fft.fft2(lunar)
lunarshift = np.log(abs(np.fft.fftshift(lunarfft)))
plt.imshow(lunarshift, cmap = "gray") #Frequencies in the lunar image. Vertical and H
filterl = lunarshift
filterl[np.where(lunarshift<11)] = 0

final = np.flip(np.abs(np.fft.fft2(np.fft.fft2(lunar)*np.fft.fftshift(filterl))))
```



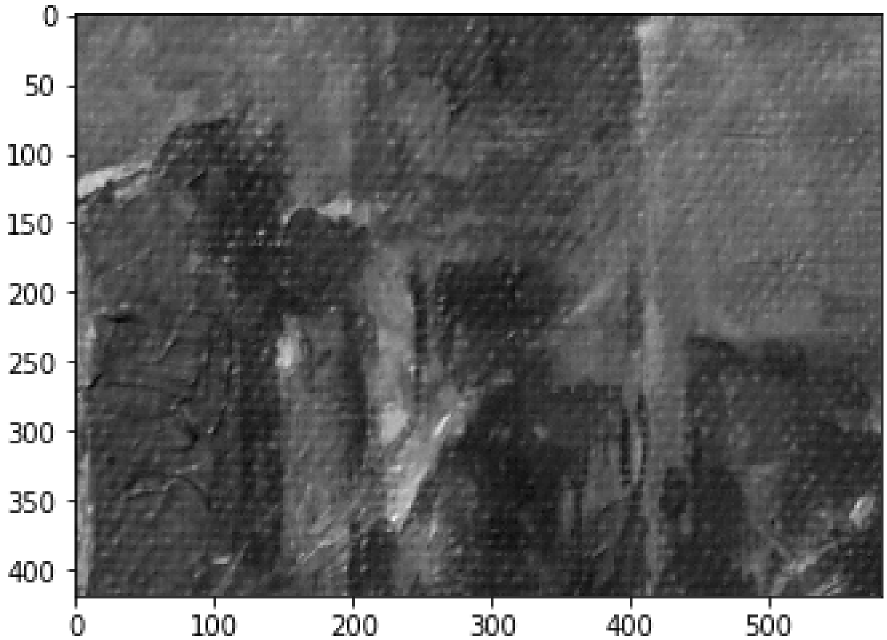
```
In [53]: plt.imshow(final, cmap = "gray")
         #Masking was not successful because the condition was not enough.

Out[53]: <matplotlib.image.AxesImage at 0x7f787af17be0>
```



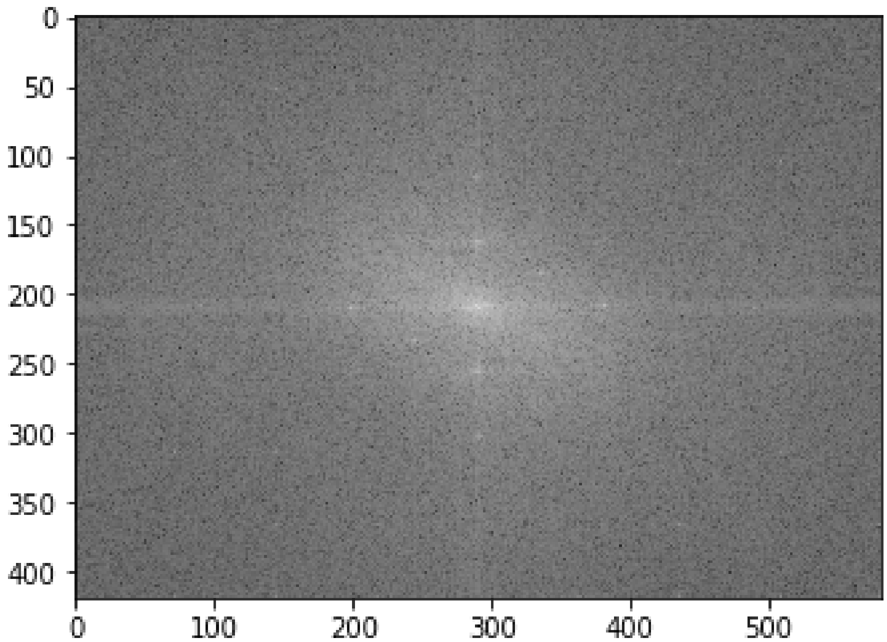
0.11 Canvas Weave Modeling and Removal

In [54]:
```
canvas = rgb2gray(plt.imread('canvas.JPG'))
plt.imshow(canvas, cmap="gray")
```

Out[54]: <matplotlib.image.AxesImage at 0x7f787b0ec978>



In [55]:
```
canvasfft = np.fft.fft2(canvas)
canvasshift = np.log(abs(np.fft.fftshift(canvasfft)))
plt.imshow(canvasshift, cmap = "gray") #Frequencies in the canvas image. Vertical and
filterc = canvasshift
filterc[np.where(canvasshift<9)] = 0

final = np.flip(np.abs(np.fft.fft2(np.fft.fft2(canvas)*np.fft.fftshift(filterc))))
```



In [56]:
```
plt.imshow(final, cmap = "gray")
#Masking semi-successful, lowerleft still has noticeable brushstrokes.
```

Out[56]: <matplotlib.image.AxesImage at 0x7f7878ddc358>