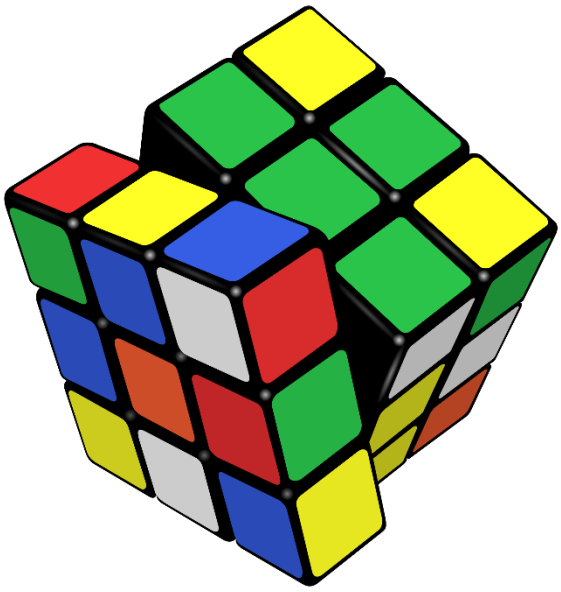
Assignment #3 – Transformations

Faculty of Engineering Science
Dept. of Electrical and Computer Engineering
Introduction to Digital Image Processing, 361-1-4751

Maor Assayag 318550746

Refahel Shetrit 204654891



1 2D-Fourier Transform

In this section we will exercise the 2D Fourier Transform on images and learn about some of its properties.

1.1 Writing your own functions

1.1.1 Write your own 2D-FFT function named *dip_fft2(I)* and an inverse-FFT function called *dip_ifft2(FFT)*.

Explanation: We wrote our own function based on following formulas. The algorithm is pretty straight-forward.
The equations for the FFT and iFFT for an image I of size $M \times N$:

$$F(u+1,v+1) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} I(m+1,n+1) \cdot e^{-2\pi i\left(\frac{pm}{M}+\frac{pn}{N}\right)}$$

$$I(m+1,n+1) = \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u+1,v+1) \cdot e^{2\pi i\left(\frac{pm}{M}+\frac{pn}{N}\right)}$$

Figure 1.1.1.1 – from the course Moodle website

1.1.2 Write your own shift zero-frequency component to center of spectrum function named *dip_fftshift(FFT)*.

Explanation: In the Moodle Q&A segment we got clearance to use *circshift* function. The aim is to shift the zero-frequency component (DC) to thecenter of spectrum. The function helped us rotate the Quarters of the FFT matrix as follow :

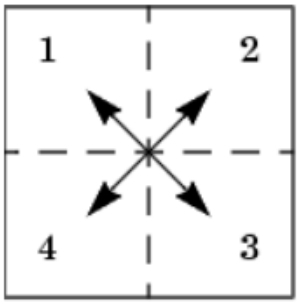


Figure 1.1.2.1 - from the course Moodle website

1.1.3 Read the beatles.png image accompanied to this exercise and convert it to grayscale. Make sure you also convert it into type double (if needed) and normalize it to [0; 1] values before proceeding. Note that the image is not a built-in MATLAB file, so please download it from the course website.

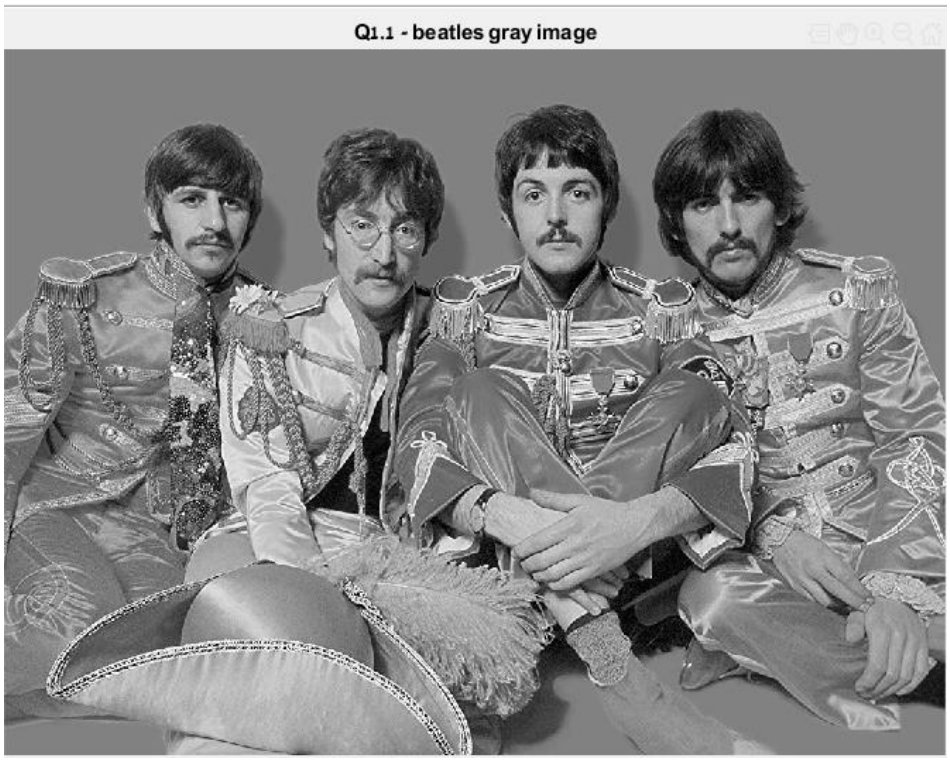


Figure 1.1.3.1

Explanation : We read the image, converted it to grayscale and normalized it as requested.

1.1.4 Compute the 2D-FFT of the image using your function, and displayed the resulting image. Make sure you shift the output image before displaying it using *dip_fftshift()*. Use *imshow()* and *colorbar* functions to display the results.

Figure 1.1.4.1

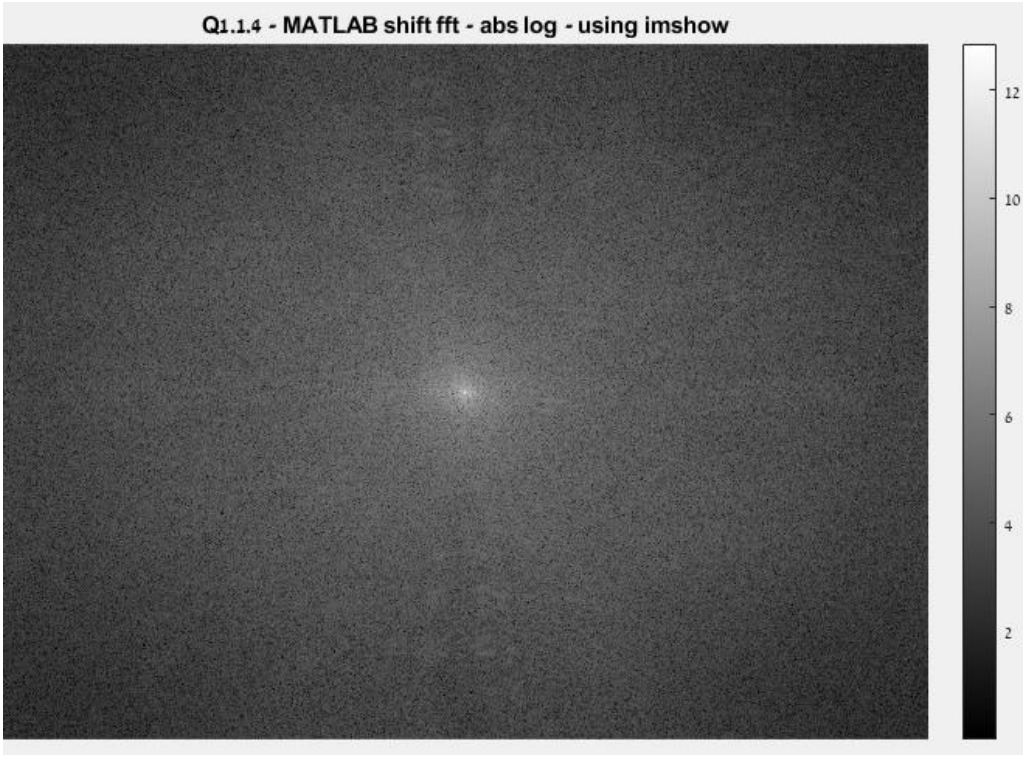


Figure 1.1.4.2

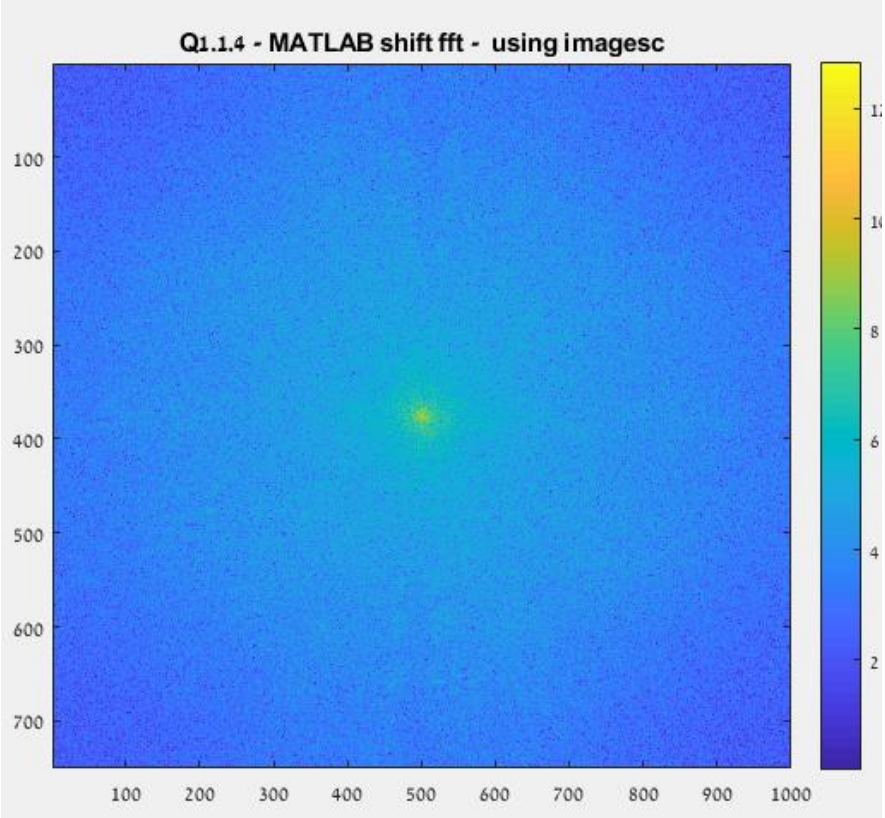


Figure 1.1.4.3

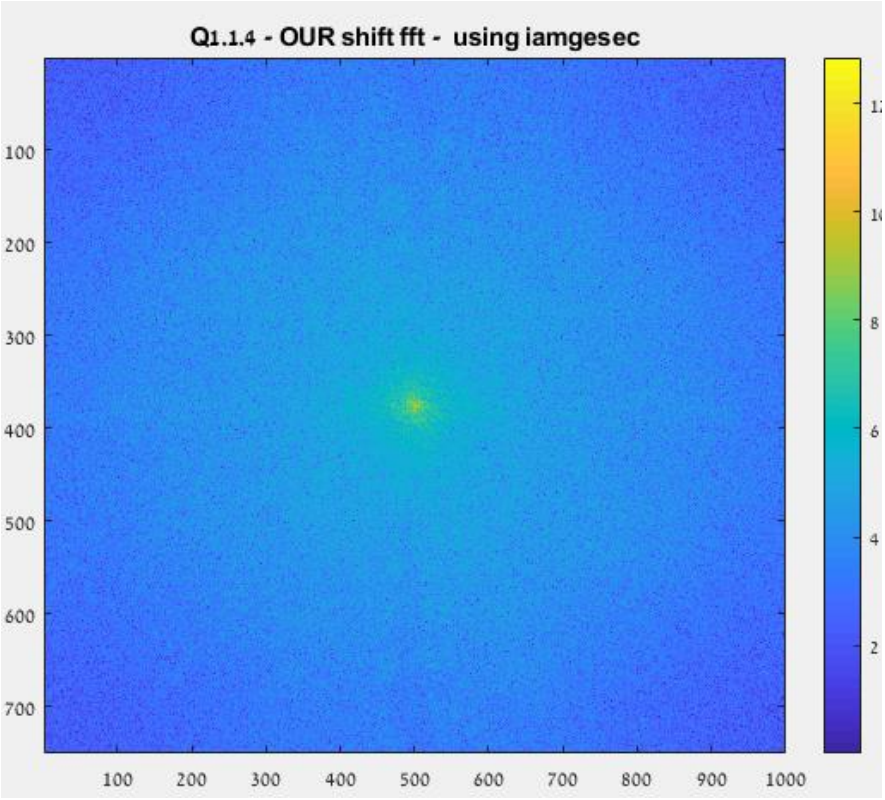


Figure 1.1.4.4

Explanation: Firstly, we displayed the shifted FFT with *imshow()* (log abs scale) in order to emphasize the difference between frequencies. We applied our *dip_fftshift()* function,

resulting the low frequencies to be at the center of the image. We also showed the FFT using the function *'imagesc()'*, which often used to show the difference in concentration. Along the process, we kept track of the results from the built-in MATLAB function as displayed above.

1.1.5 Reconstruct the original image by using your inverse-FFT function. Is it identical to the original image? Note that the output of the iFFT are complex numbers - you should display only the real part of the image using *real()*.

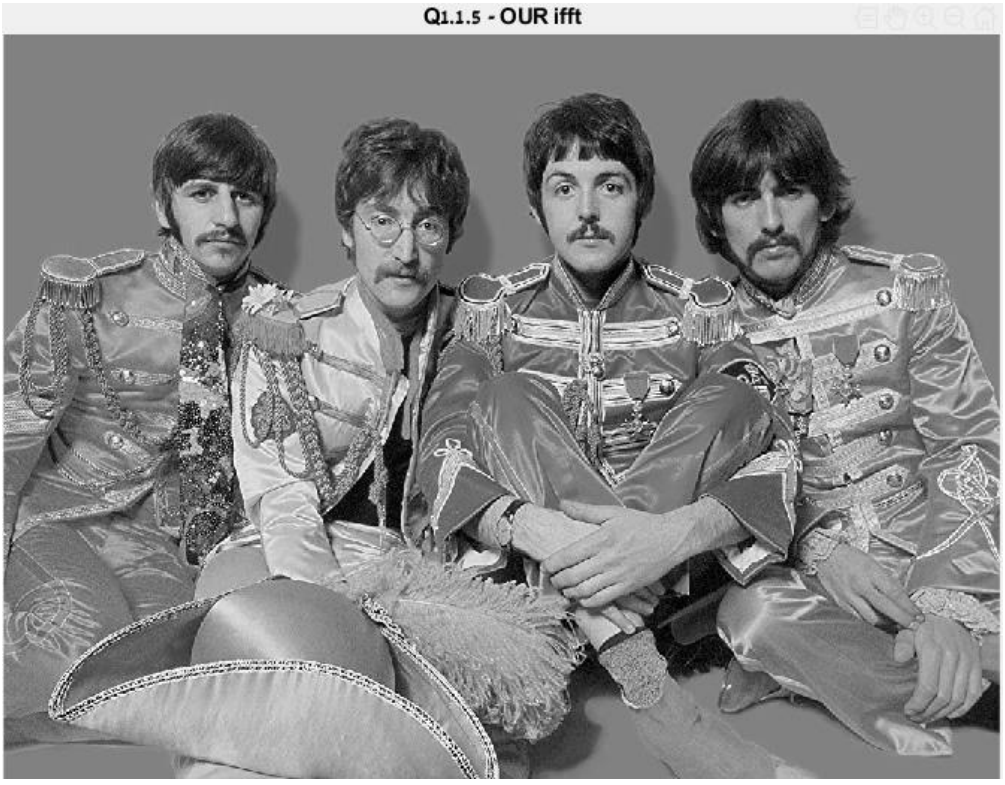


Figure 1.1.5.1

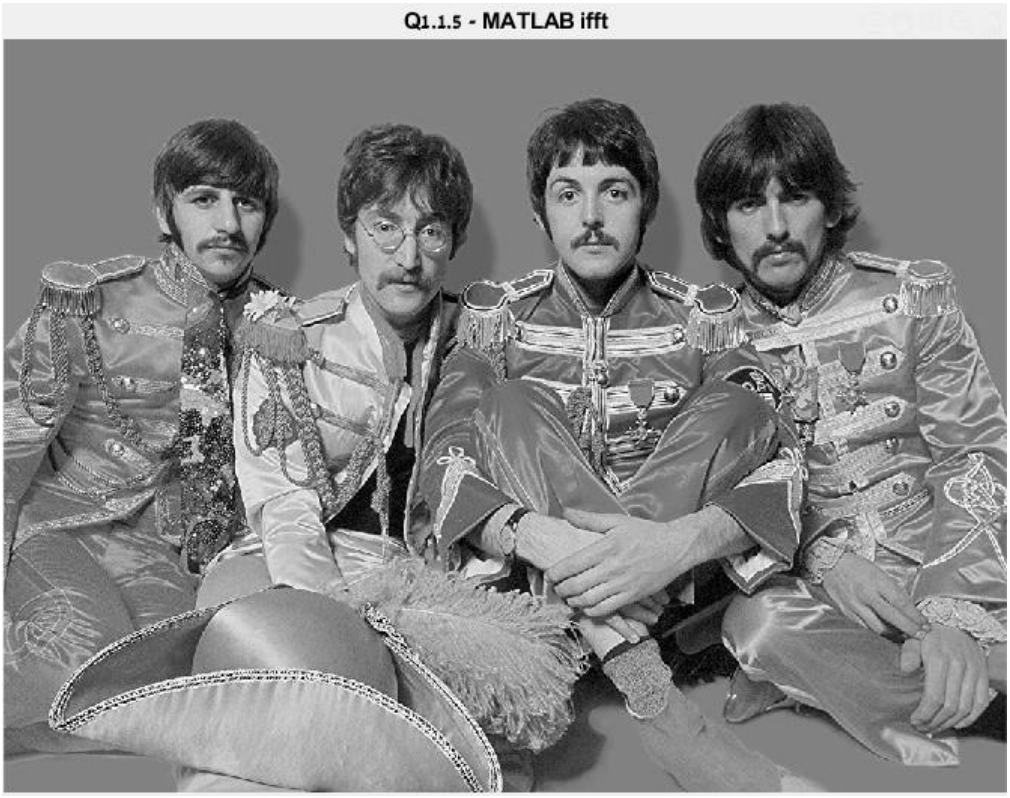


Figure 1.1.5.2

Figure 1.1.5.3

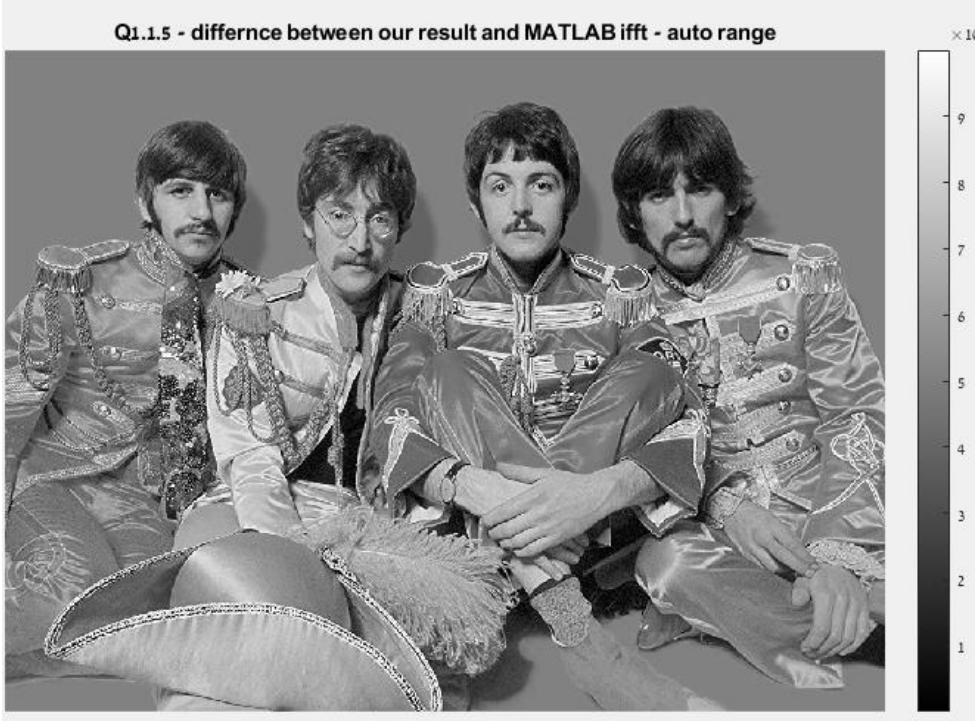


Figure 1.1.5.4

Explanation: After using *dip_ifft()* function, we reconstruct the original FFT image. We displayed the results using *real()* and *imshow()*. We also checked the difference between our results and MATLAB, we found out the we have an error in a scale of max ~0.007 due to rounding operations. If we displayed the difference In the range *of [0,1]* – we see a black image (all values close to 0). In the other hand, we found out that the error changing equally from pixel to pixels, resulting the same picture if we choose auto range in *imshow()* !

1.2 Transformation properties

In this section you MAY use the built-in MATLAB *fft2()*, *ifft2()* and *fftshift()* functions.

1.2.1 Linearity (Free Willy)

a. Load the **freewilly.mat** file enclosed to the exercise. Display it as an image using *imshow()*. don't normalize this file.

Figure 1.2.1.a.1

Explanation: We loaded the file using the *load()* built-in function.

b. As you can see, Willy the whale is imprisoned. Your job is to free Willy! To do that, you are told that the prison bars are made of a sinudoidal signal in the X axis that was added to the original image: $0.5 * \sin(2 * pi * \frac{f_x}{N} * x)$, where N is the number columns in the image. Given this image, find the spatial frequency of the prison bars fx. Explain your answer! (Hint: how many bars are there in the image? You may also plot the first row of the image if you find it helpful).

Explanation: To find the spatial frequency of the prison bars(the noise), we took a row of pixel data and built the function *'dip_peaksCounter()'* to count the number of extreme points in the data. Each point will describe the middle of a strip, hence we will discover how much bars of noise was added to the image. This number represent $f_x = 10[hz]$ in our case.

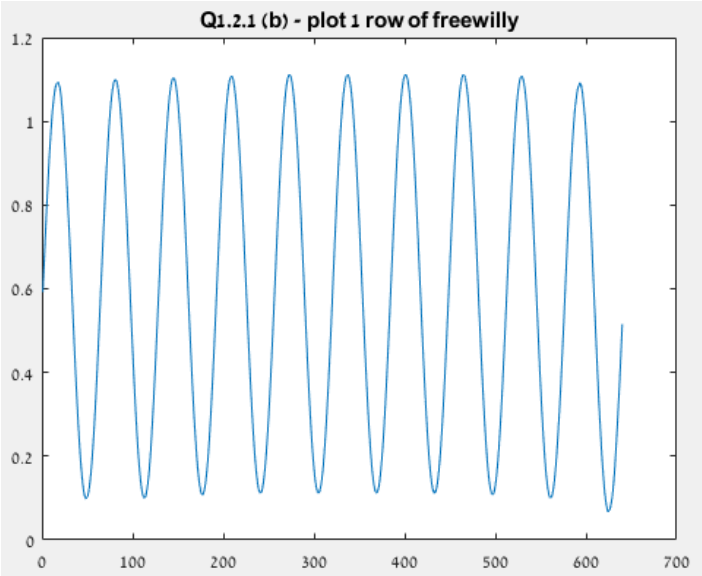


Figure 1.2.1.b.1

c. Based on your answer in section (b), create the image of the prison bars and display it. MAKE SURE it has the same dimensions as *frewilly.mat*.

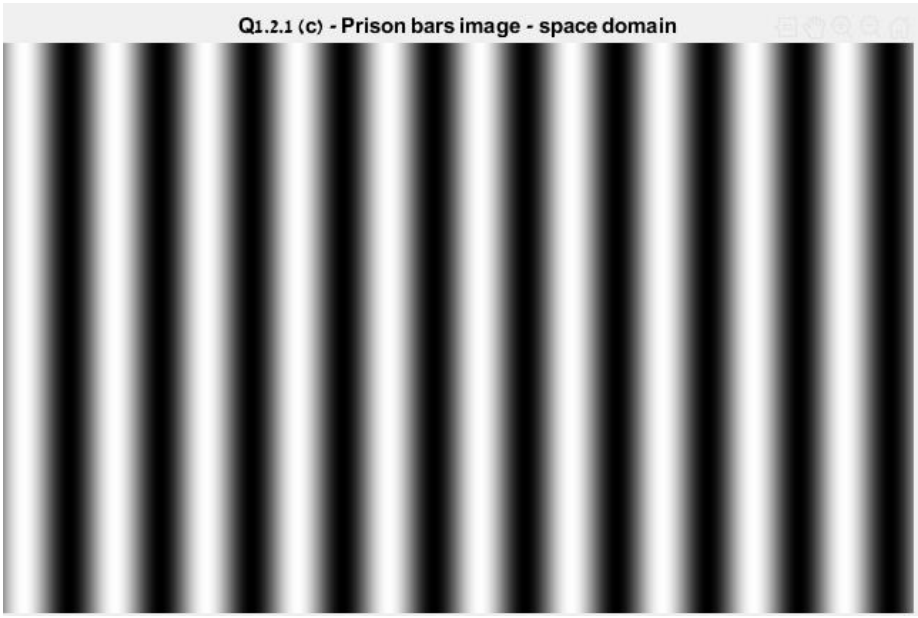


Figure 1.2.1.c.1

Explanation: Based on our answear, we wrote the function *' dip_create_prison(numOfPeaks, rows, cols)'* which is creating a matrix of the sinus

wave for each row. The image size is rows x cols. We displayed the image using *'imshow()'* with auto range.

Right now, we can try to free willy by subtract the noise image from the *freewilly* image - this will not give us a satisfactory result. We found out that the sinus wave surpassing the value 1 in some points(and obviously for each grayscale value if the noise succeeds its [1-value] value we will lose data), which resulting after subtracting the original pixel values of *willy* to be gone. The effect of this is the follow :

Figure 1.2.1.c.2

d. Compute the 2D-FFT of the prison bars image and display its amplitude (use the function abs()). Explain this result - give a mathematical proof that this is the Fourier transform that is expected here.

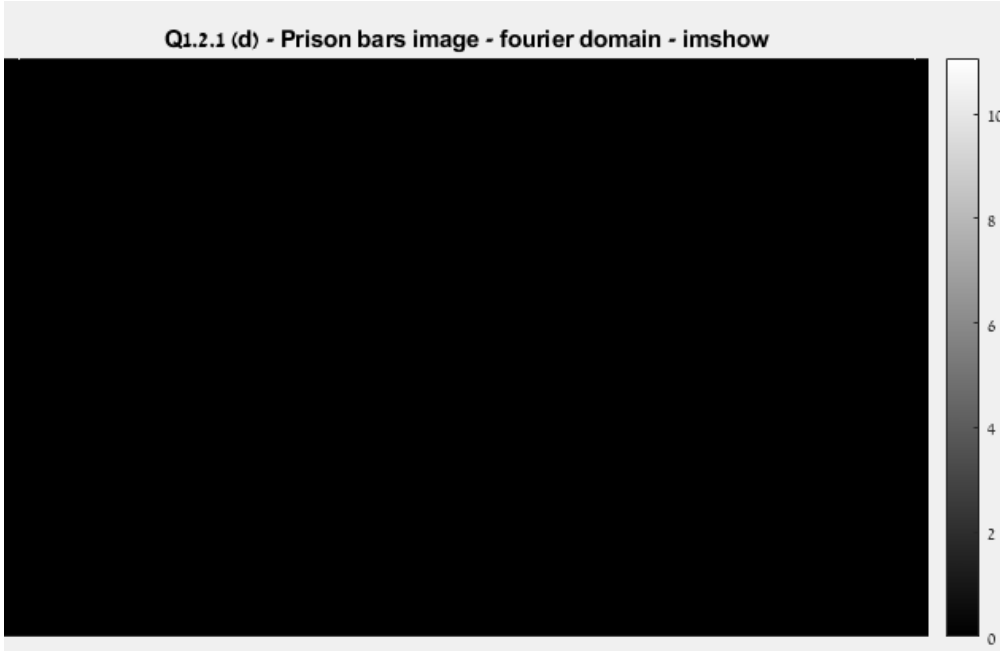


Figure 1.2.1.d.1

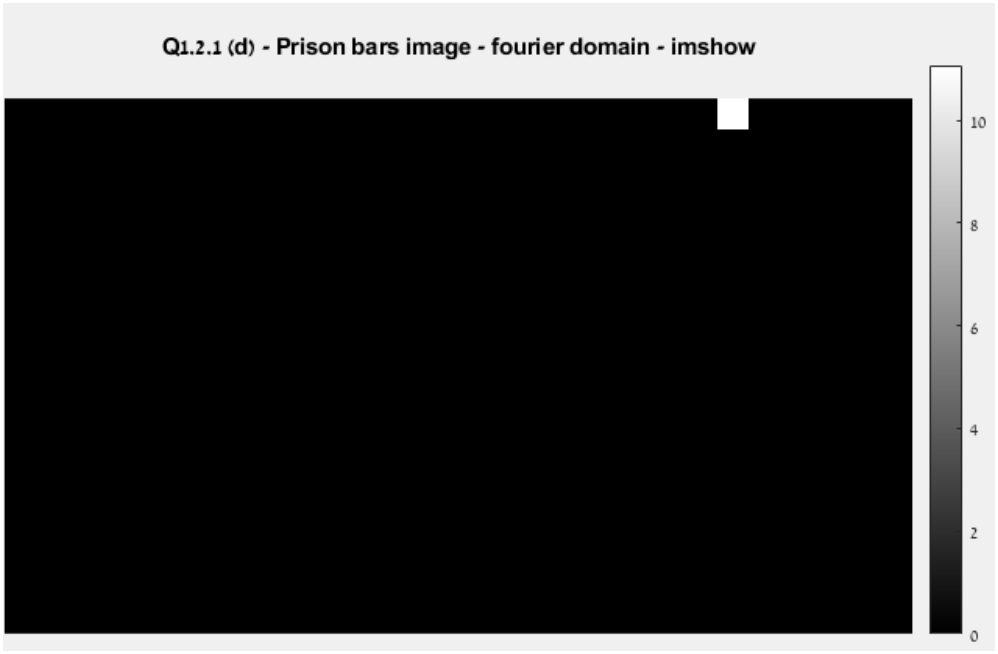


Figure 1.2.1.d.2 – zoom in

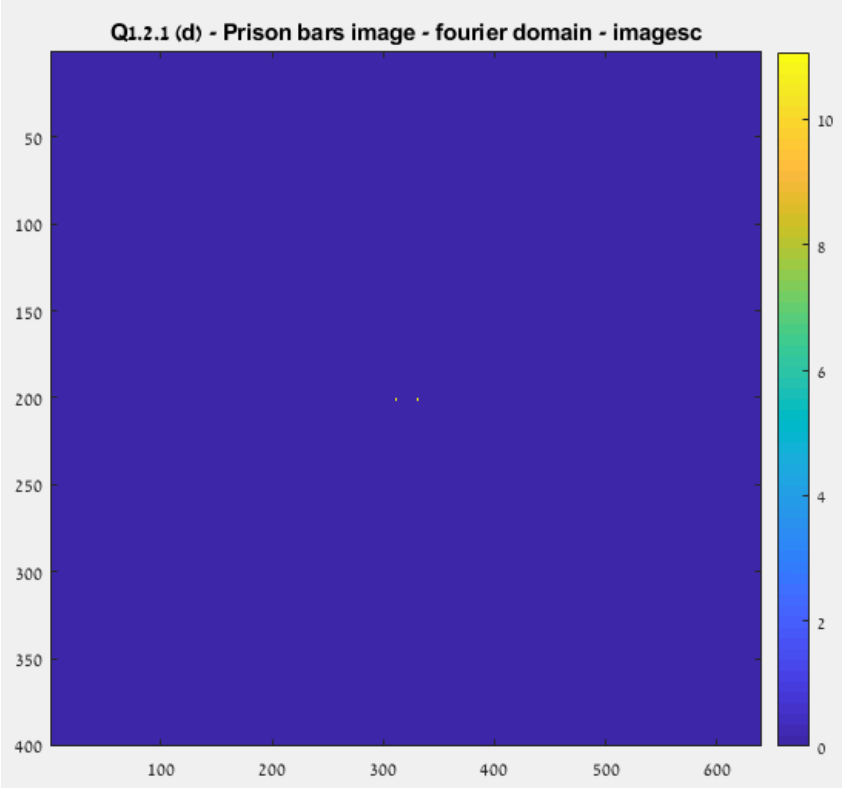


Figure 1.2.1.d.3

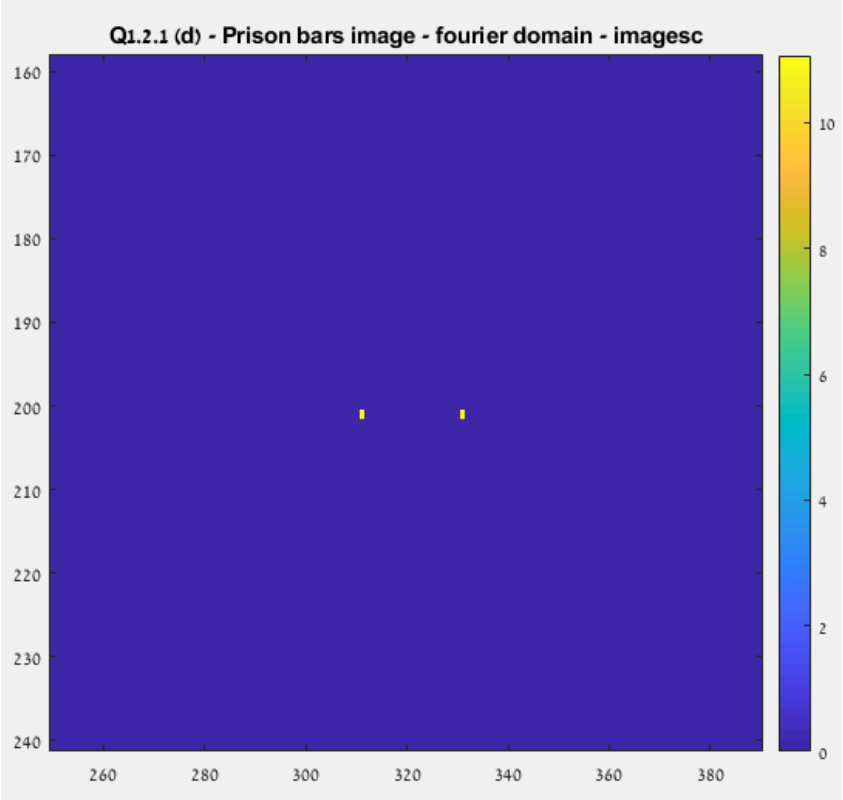


Figure 1.2.1.d.4 – zoom in

Explanation: The Fourier transform of sinus wave is 2 delta function, as follow :

$$f[x,y] = \frac{1}{2}\sin\left(2 * \pi * \frac{f_x}{N} * x\right)$$

$$\mathrm{F}[m+1, n+1] = \sum_{n=0}^{N-1}\sum_{m=0}^{M-1}\frac{1}{2}\sin\left(2 * \pi * \frac{f_x}{N} * (m+1)\right)e^{-\frac{j2\pi um}{M}}e^{-\frac{j2\pi vn}{N}}$$

$$= \sum_{n=0}^{N-1} e^{-\frac{j2\pi vn}{N}} \sum_{m=0}^{M-1}\frac{1}{2}\sin\left(2 * \pi * \frac{f_x}{N} * (m+1)\right)e^{-\frac{j2\pi um}{M}}$$

$$= C * \delta[n] * (\delta[v - f_x] - \delta[v + f_x]), C = const$$

As we can see, we got 2 delta function in the Fourier domain, which translate to 2 pixels in the images (2 spikes of values). We showed the FFT with *'imshow()'* (no shift), and for more clear visualization we showed it with '*imagesc*' and shift.

e. Explain how you can free Willy (i.e. filter out the prison bars) through the Fourier domain. Based on your answer, write a function Free_Willy(Willy) that returns and displays Willy without the prison bars.

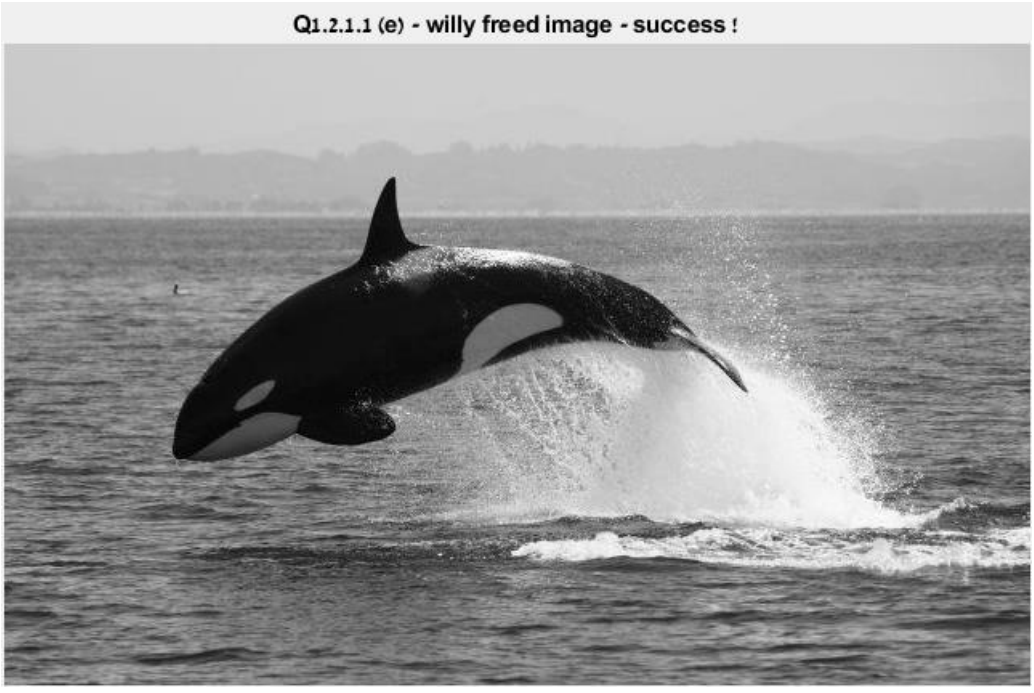


Figure 1.2.1.e.1

Explanation : After we computed the FFT of the *prison bars* image and *freewilly* image, we created a 'binary map' of the delta frequency (using simple comparison between the two FFT's. Then, we zeroed out thus frequency resulting the pixels to be blued out (according to the *colorbar*). Finally, we computed the IFFT for the result of freed willy.

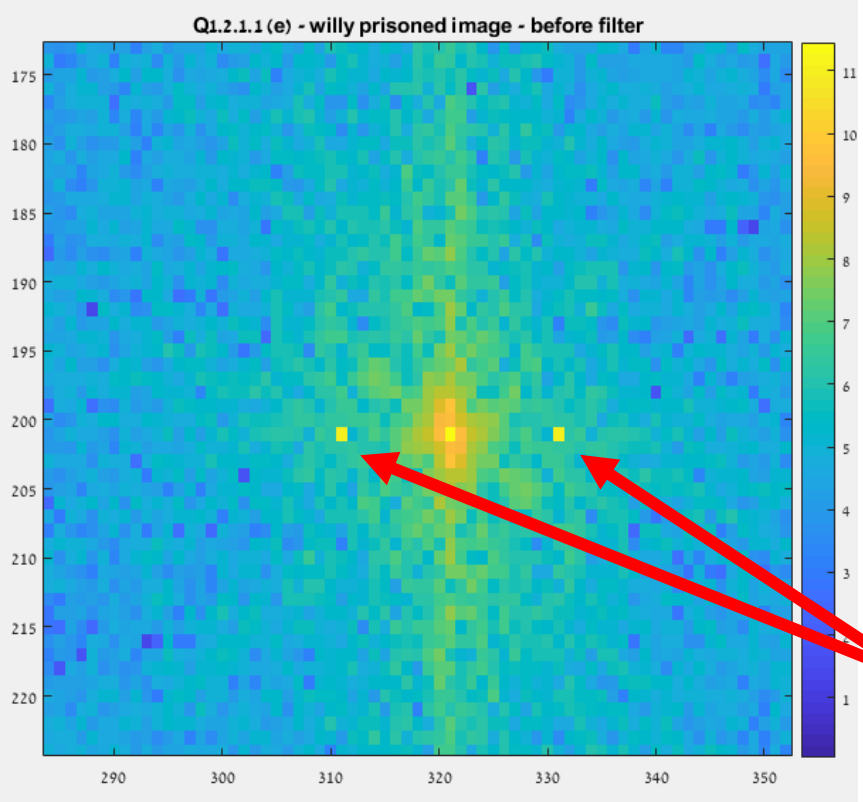


Figure 1.2.1.e.2

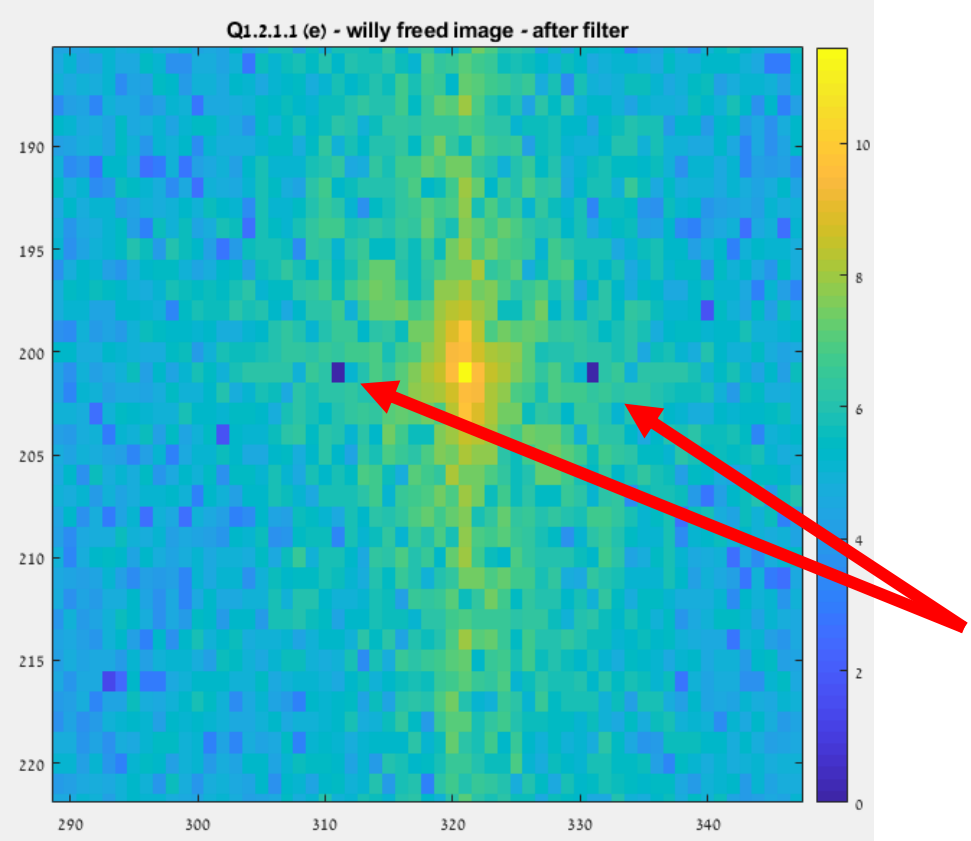


Figure 1.2.1.e.3

1.2.2 Scaling, translation and separability

Note : we used *imagesc* function to demonstrate the FFTs phase and amp, we know how to use *imshow* as shown previous, we felt that this is the best tool for understanding what we see.

a. Initialize a 128128 all-zeros matrix. At the center of it, place a 40x40 all-ones square (in pixels 44:83 in each dimension). Display the image and its 2D-FFT. Explain why it looks the way it does.

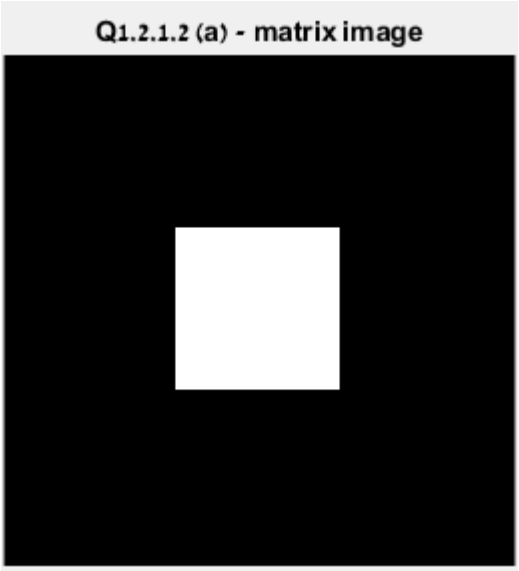


Figure 1.2.2.a.1

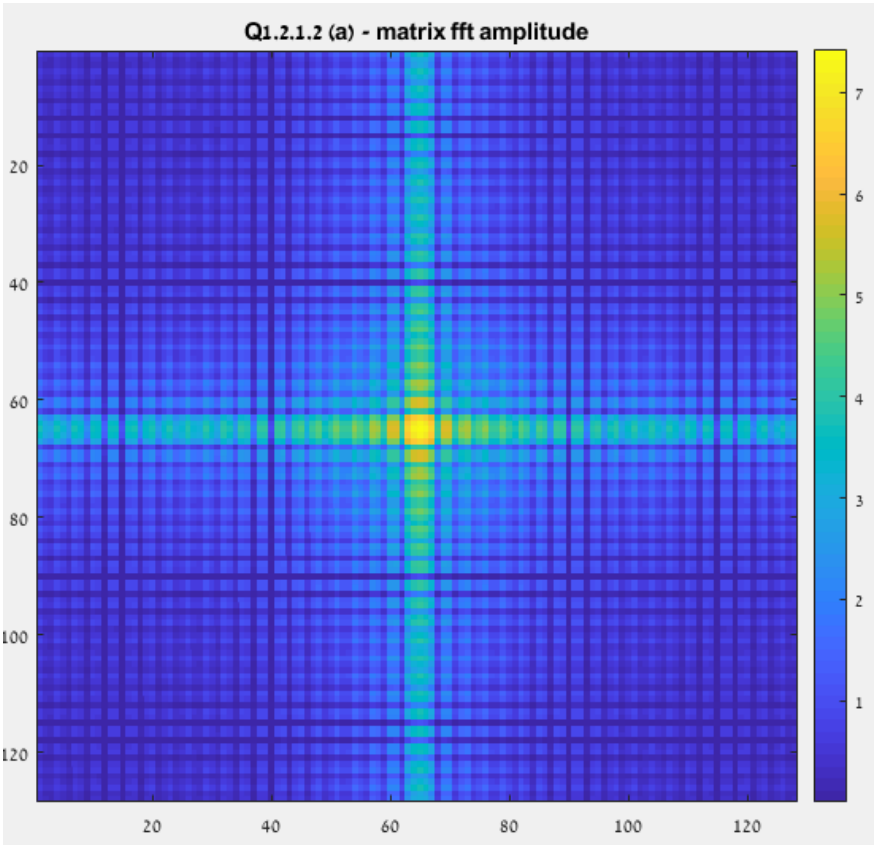


Figure 1.2.2.a.2

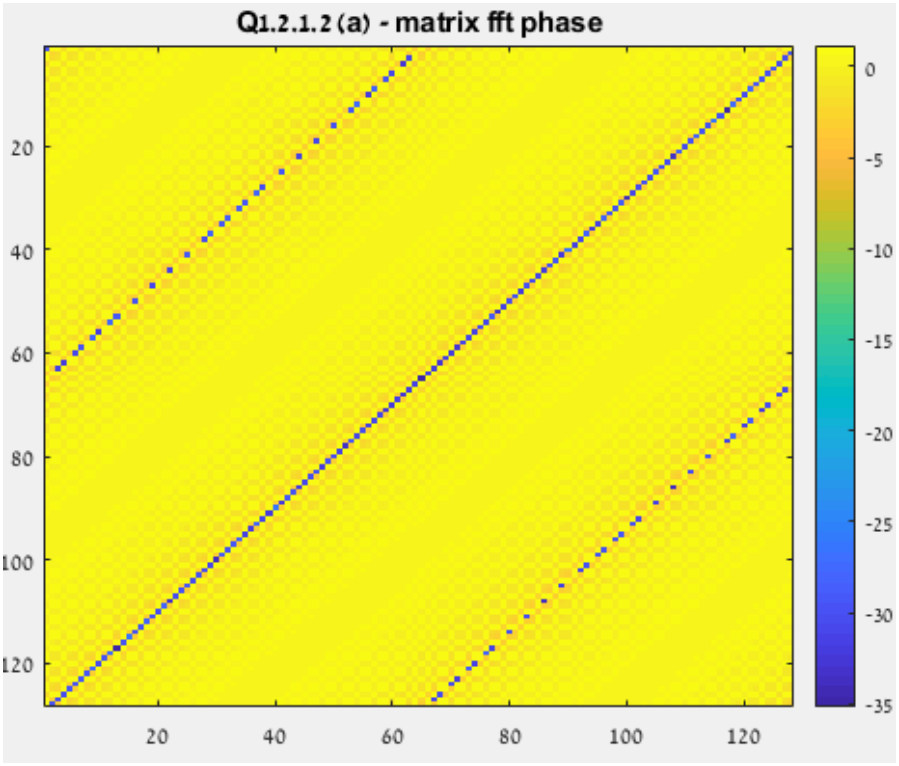


Figure 1.2.2.a.3

Explanation: The original image is composed from a all-one square. This means that the image is composed from 2 rectangular windows functions at each axis. The FFT of the window function is a sinc, so we expect to see the FFT of multiplication of 2 sincs exactly as shown in the lecture. We can see that we got centerd fourier spectrum shown more clearly after application of the log transformation.

$$\textit{Rectangle} \qquad \text{rect}[a,b] \Leftrightarrow ab\frac{\sin(\pi ua)}{(\pi ua)}\frac{\sin(\pi vb)}{(\pi vb)}e^{-j\pi(ua+vb)}$$

Figure 1.2.2.a.4 – FFT formulas

b. As Initialize a 128 x128 all-zeros matrix. At rows 64:103 and columns 64:103, place a 40x40 all-ones square. Display the new image and its 2D-FFT. Does the FFT looks the same as in section (a)? Now Display only the FFT amplitude of the previous and the new image. Are hey they identical? Explain why. Base your answer on the mathematical relationship between the two images and their 2D-Fourier transforms.

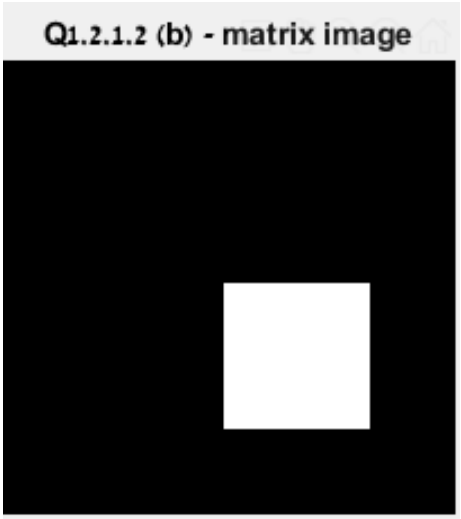


Figure 1.2.2.b.1

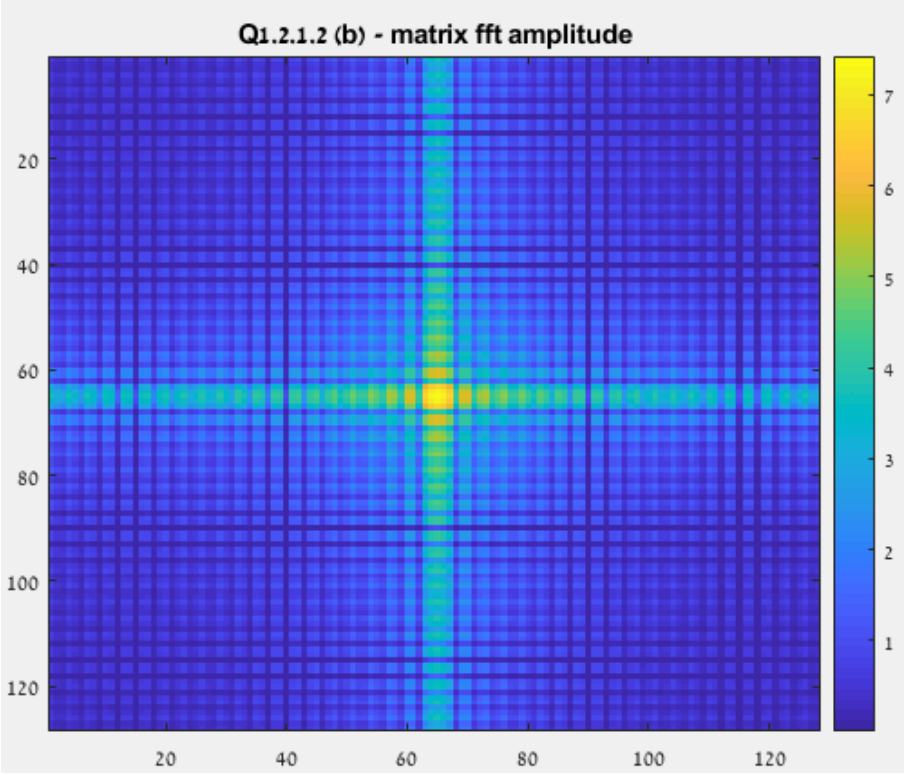


Figure 1.2.2.b.2

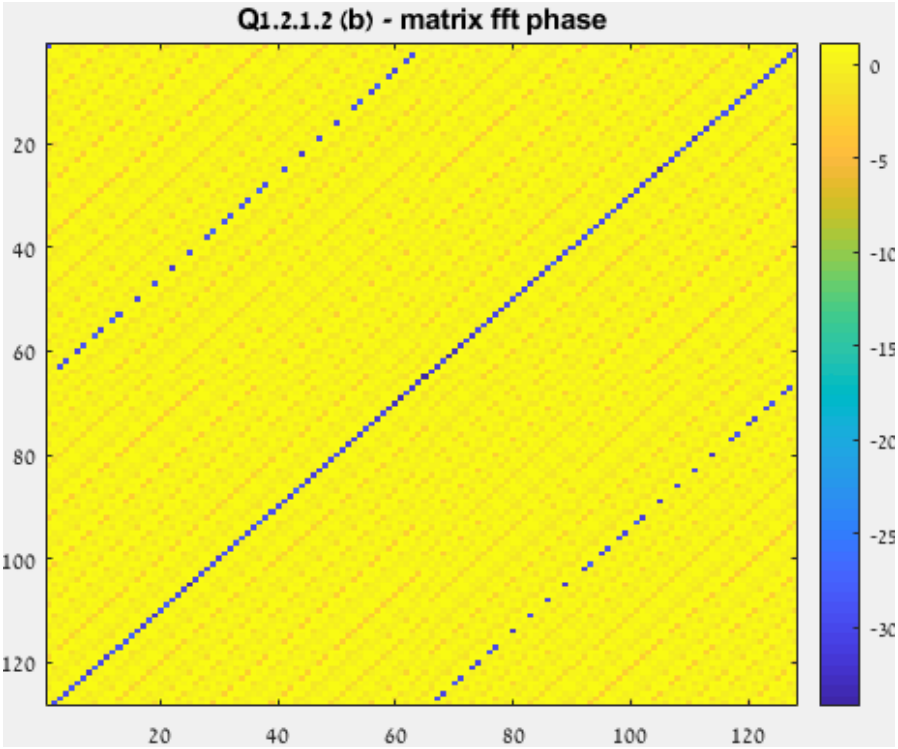


Figure 1.2.2.b.3

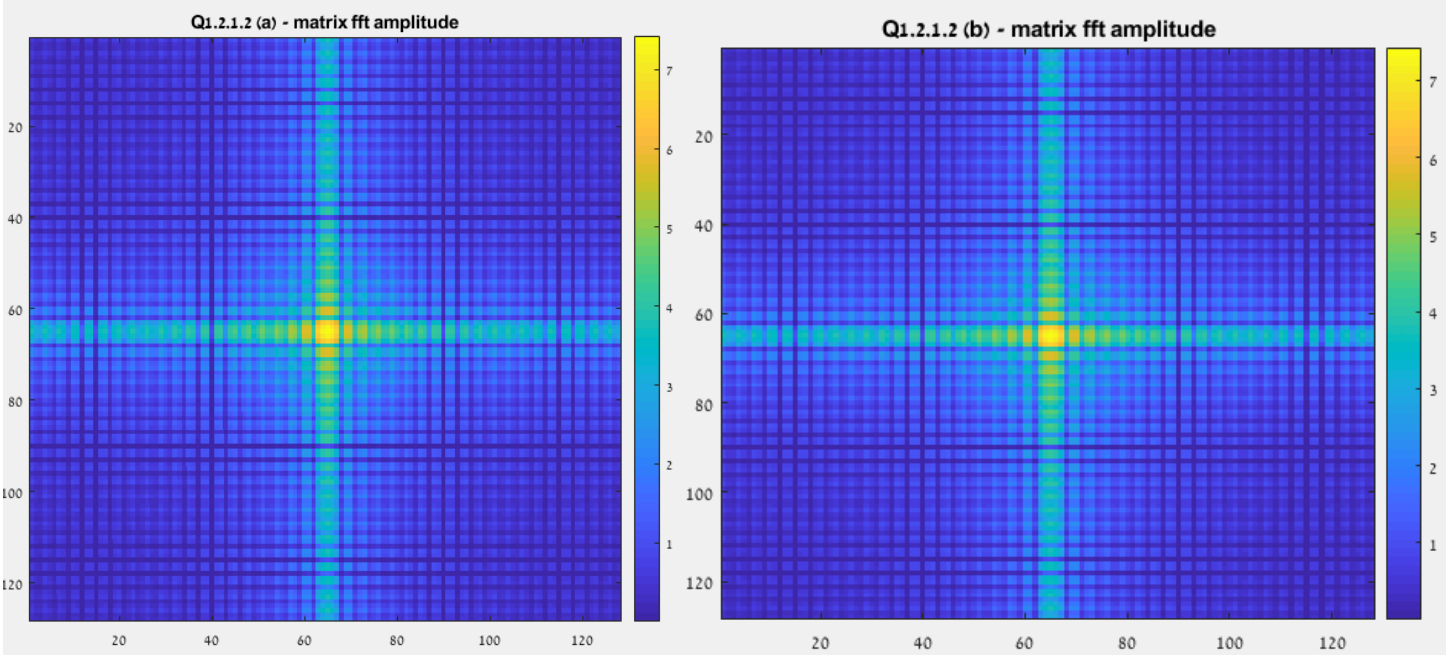


Figure 1.2.2.b.3

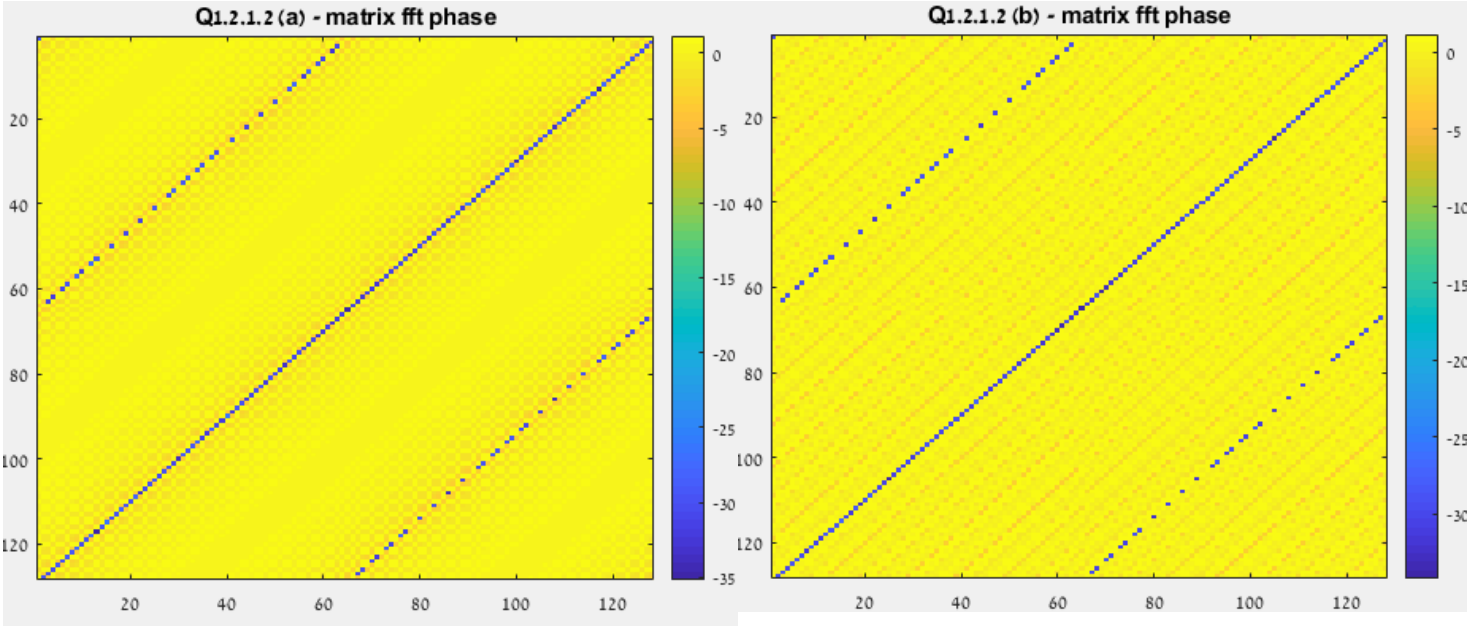


Figure 1.2.2.b.4

Explanation: We can't notice any change in the magnitude because the spectrum of the image didn't change (still a square). We learned that shifting signals in the space domain results shifting the phase in the Fourier domain – as we see in the above images. The formulas as follow :

Translation $f(x, y)e^{j2\pi(u_0 x/M + v_0 y/N)} \Leftrightarrow F(u - u_0, v - v_0)$

$f(x - x_0, y - y_0) \Leftrightarrow F(u, v)e^{-j2\pi(ux_0/M + vy_0/N)}$

Figure 1.2.2.b.5

c. Initialize a 128x128 all-zeros matrix. At the center of it, place a 80x20 all-ones rectangle. Display the new image and its 2D-FFT. Does the FFT looks the same as in section (a)? Explain why. Base your answer on the mathematical relationship between the two images and their 2D-Fourier transforms.

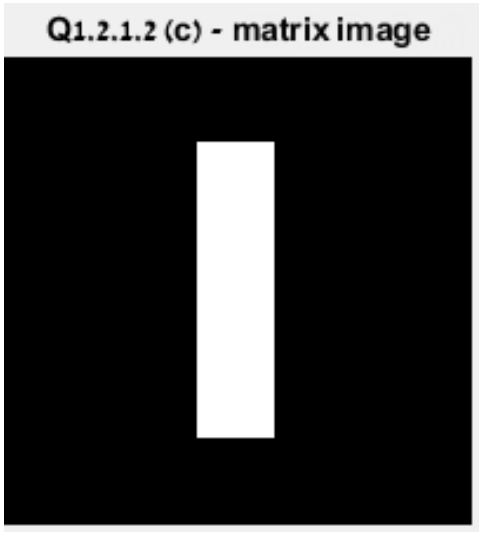


Figure 1.2.2.c.1

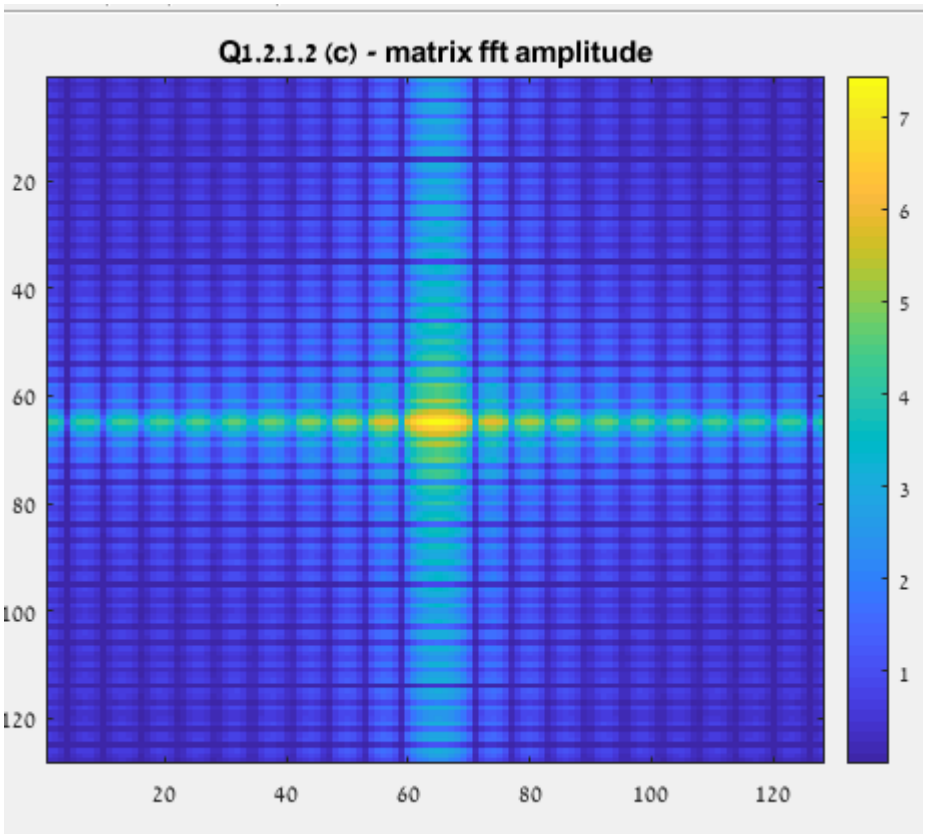


Figure 1.2.2.c.2

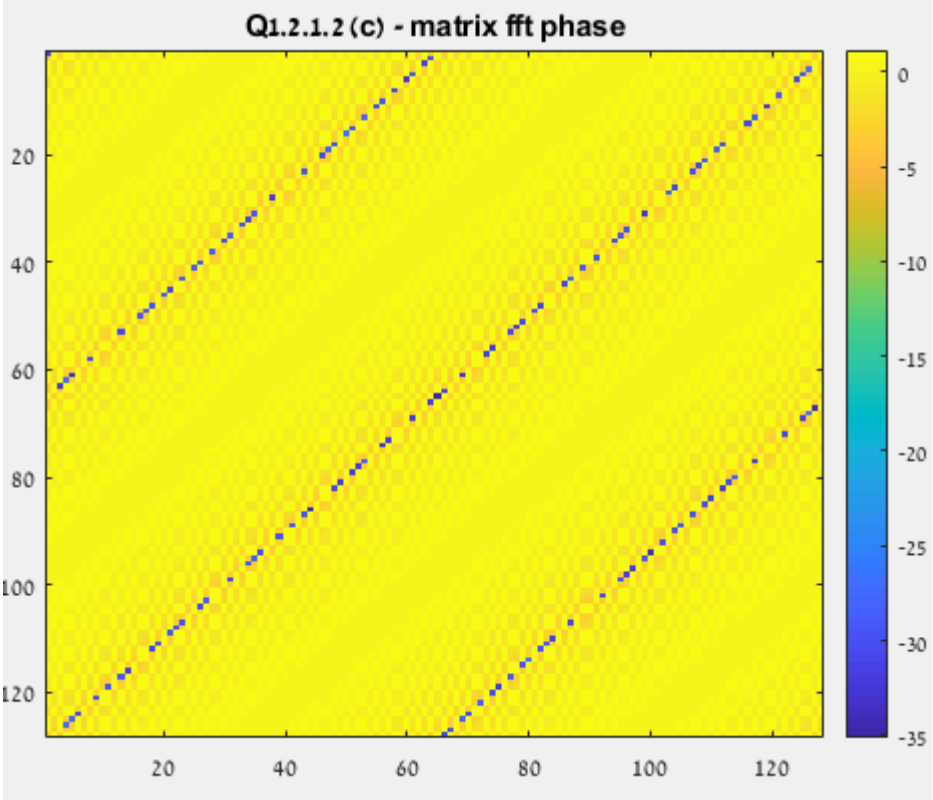


Figure 1.2.2.c.3

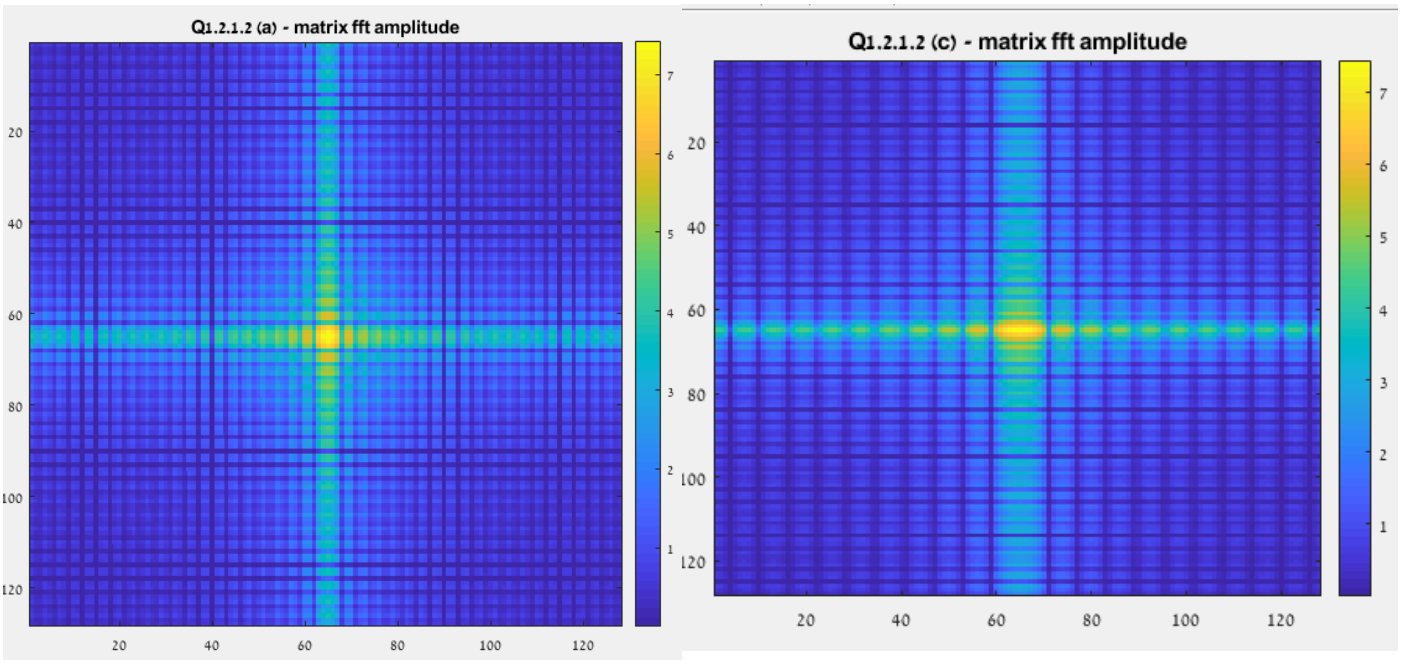


Figure 1.2.2.c.4

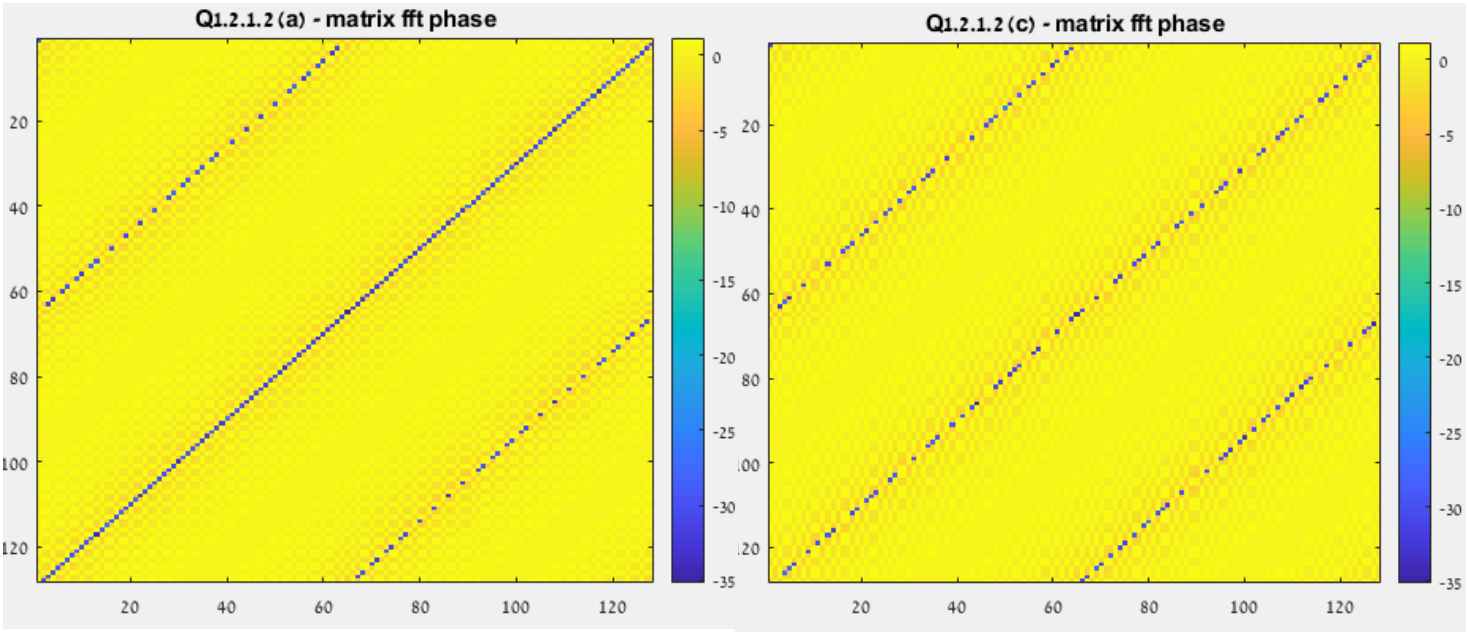


Figure 1.2.2.c.5

Explanation: The FFT of a rectangular feature a pattern surrounding the center of the image. We can observe that the pattern extends further in the direction that is perpendicular to the longer side of the rectangle, although we can also see the pattern corresponding to the shorter side. This is a concrete example of the anamorphic property of the FFT of the images. A longer length in the space dimension of the images will correspond to a compacted representation in the spatial frequency dimension after applying the FFT since this is the inverse of the space dimension.
Note : We used the web to get this explanation with the correct mathematical definitions.

$$\mathcal{F}(x(at)) = \frac{1}{|a|}X(\frac{f}{a})$$

Figure 1.2.2.c.6

d. Can you represent the image from section (c) using two 1D vectors? Explain how.

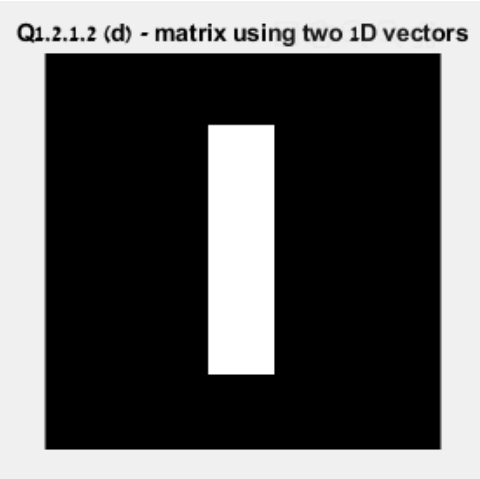


Figure 1.2.2.d.1

```
%(d) - Can you represent the image from section (c) using two 1D vectors?
vector_col = zeros(128,1);
vector_row = zeros(1,128);
vector_row(54:74) = 1;
vector_col(24:104) = 1;
matrix = vector_col * vector_row;
figure;
imshow(matrix, []);
title('Q1.2.1.2 (d) - matrix using two 1D vectors');
```

Figure 1.2.2.d.2 – MATLAB code for creating the image

Explanation: We simply created 2 row vectors that represent the indexes of columns and rows which need to intersect to create the rectangle matrix of ones. Thus, the matrix can be represented by simply multiply the col vector and the row vector.

e. Explain how you can compute the 2D-FFT of an image using 1D-FFTs if the image is separable into two 1D vectors. Write a function sep_t2(v1,v2) that receives a pair of 1D vectors (of lengths N1 and N2 respectively) and returns the 2D-FFT (a N1*N2 matrix) based on the 1D-FFTs of the vectors (DO NOT use t2() here). Apply this function on the two vectors you described in section (d) and display the resulting 2D-FFT. Is it identical to the 2D-FFT of the image from section (c)?

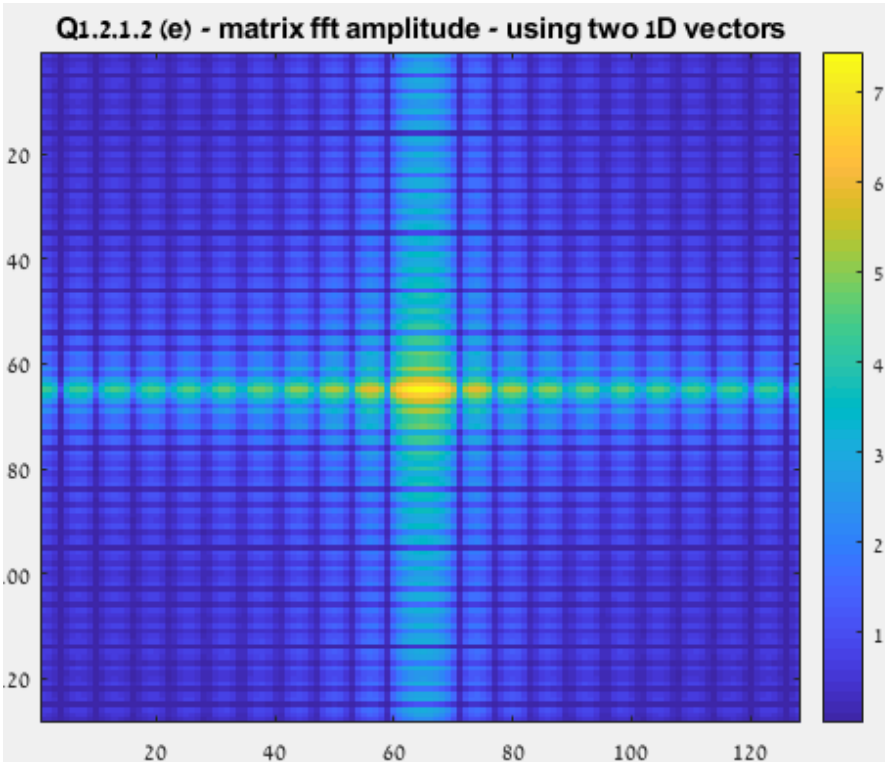


Figure 1.2.2.e.1

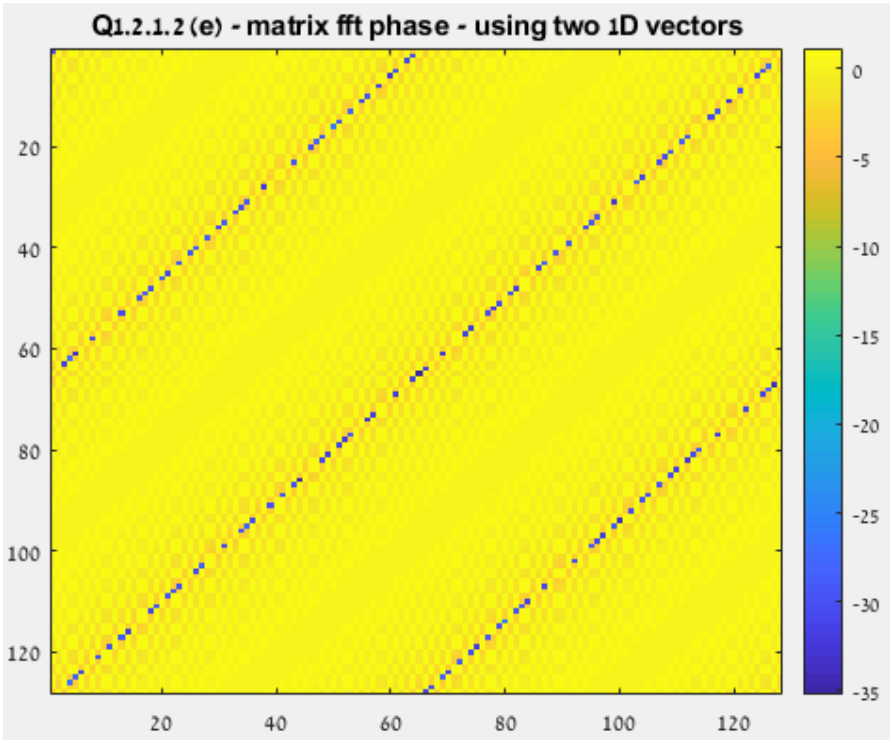


Figure 1.2.2.e.2

Explanation: We used the separability property of the Fourier transform. A 2D spectrum can be displayed compute by multiplication of two 1D FFT.

Separable functions $f(x,y) = f(x)f(y) \Leftrightarrow F(u,v) = F(u)F(v)$

Figure 1.2.2.e.3

Proof for the separability property :

$$f[x,y] = f[x]f[y] \rightarrow \mathrm{F}[m+1,n+1] = \sum_{n=0}^{N-1}\sum_{m=0}^{M-1} \mathrm{f[m+1,n+1]}\, e^{-\frac{j2\pi um}{M}} e^{-\frac{j2\pi vn}{N}}$$

$$= \sum_{n=0}^{N-1} f[n]\, e^{-\frac{j2\pi un}{N}} \sum_{m=0}^{M-1} f[m]\, e^{-\frac{j2\pi um}{M}} = F[m]F[n]$$

```
function fft = sep_fft2(v1,v2)
% Q1.2.2 (e) - compute matrix 2D FFT from 2 1D vectors
% v1 - input, row vector
% v2 - input, col vector
% fft - 2D computed FFT
fft_v1  = dip_fft2(v1);
fft_v2  = dip_fft2(v2);
fft = fft_v1 * fft_v2;
end
```

Figure 1.2.2.e.4 – MATLAB code for creating the 2d FFT

We can see that we got the exact results like (c).

2 Wavelet Transform

In this exercise we will learn the tools for 2D Wavelet Decompostion and their applications to images.

2.1 Wavelet tools

1. Read the beetle.jpg image (enclosed to this exercise), convert it into grayscale image and normalize to [0; 1].

Figure 2.1.1.1

Explanation: We read the image, converted it to grayscale and normalized it as requested.

2. Write Extract the wavelet decomposition using MATLAB's wavedec2() function: use the 'haar' wavelet, you may choose any level of decomposition between 3-5.

3. Use Matlab's detcoef2 and appcoef2 functions to find the detail and approximation coefficients for each level.

4. Display the results for each level.

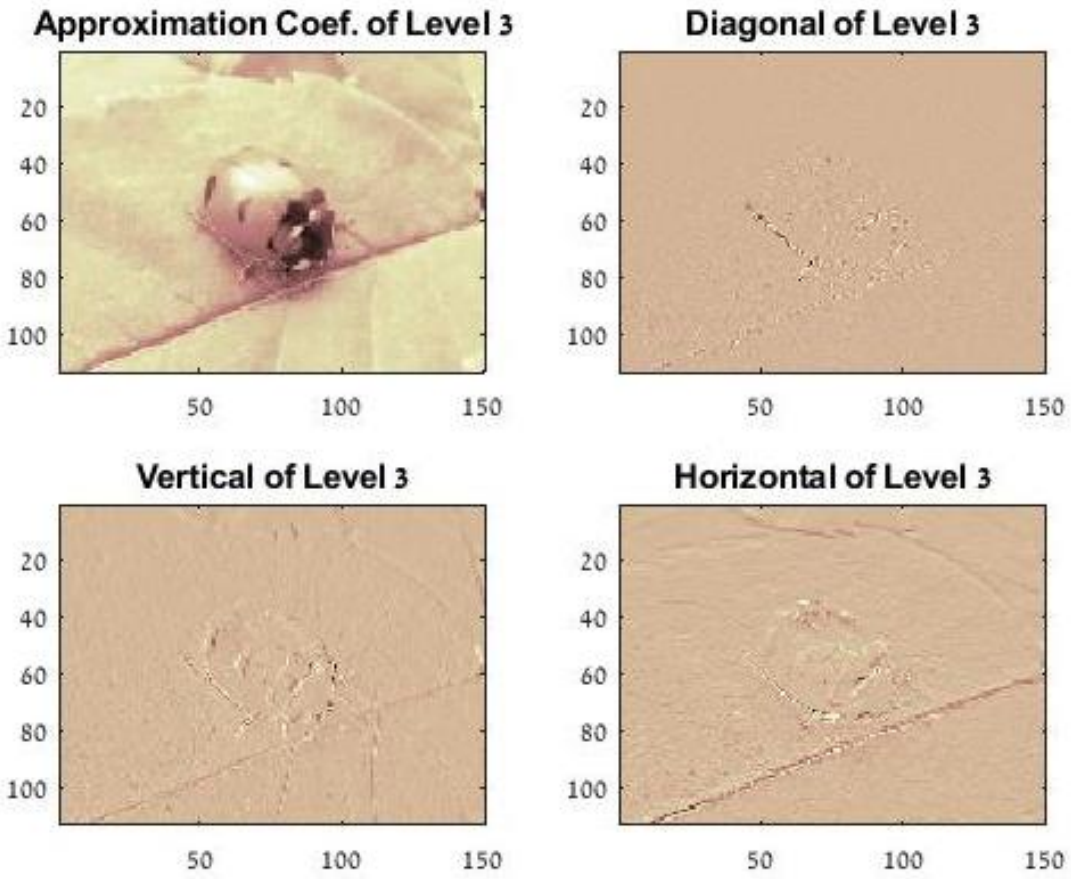


Figure 2.1.4.1

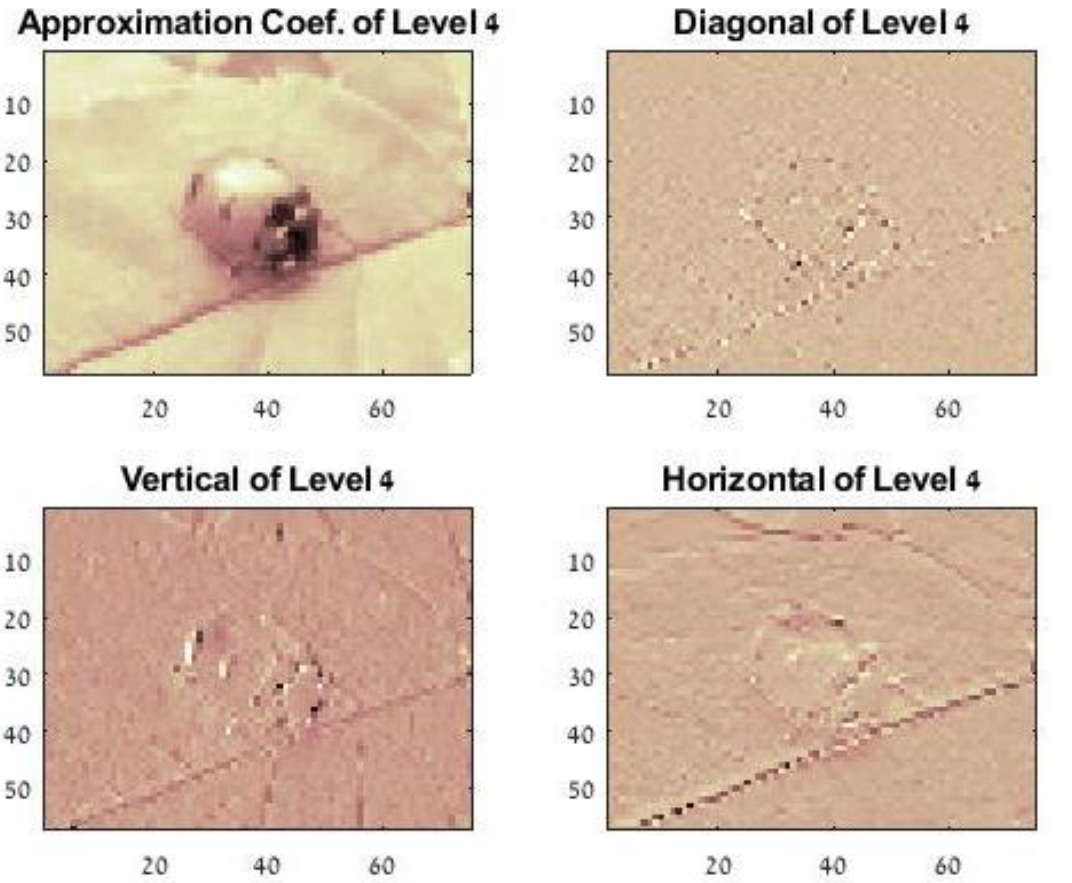


Figure 2.1.4.2

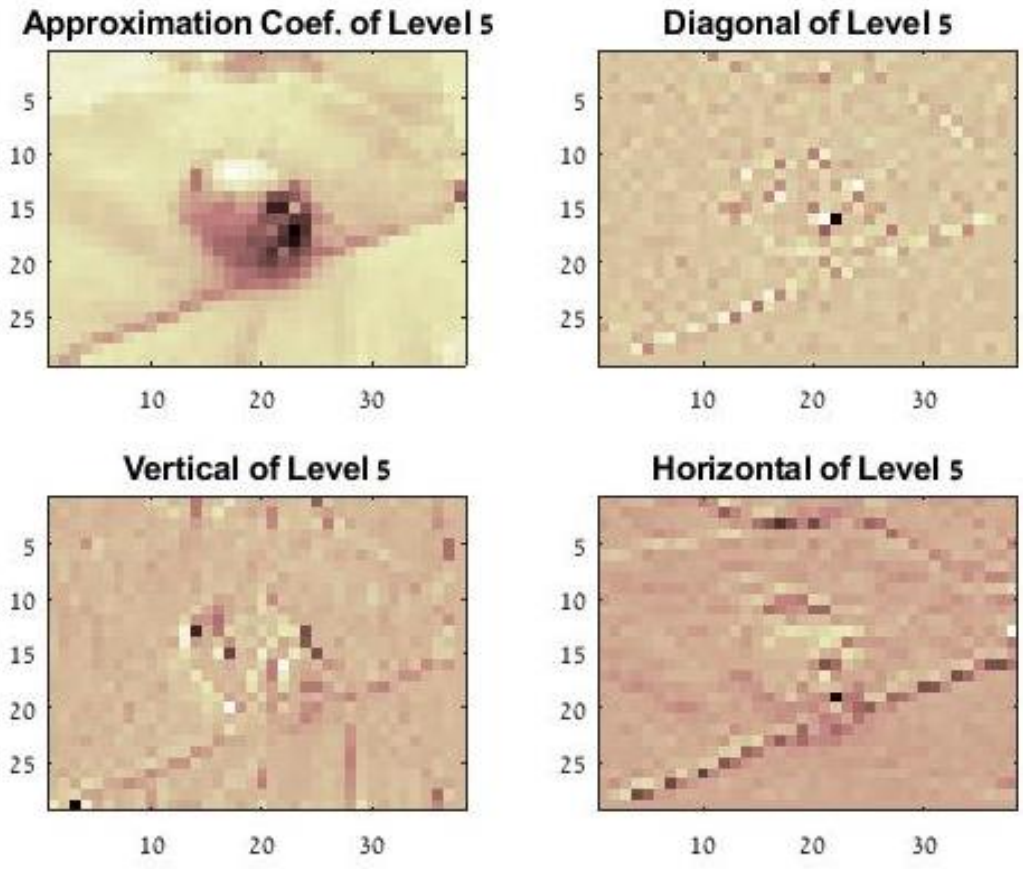


Figure 2.1.4.3

5. Explain the results. What are the differences between each level? What can you learn from each level?

Explanation:
When we increased the level of decomposition the image become more pixelated, due to heavier compression.
The following table consist of each level effect on our image :

Level	Number Of pixels	% from original data size (from 100%)
3	113*150 = 16950	1.569 %
4	57*75 = 4275	0.39 %
5	29*38 = 1102	0.1 %

Table 2.1.5.1

For this compression we used the Haar wavelet which consider 'basic' – we approximate the image by the following operations, according to the lecture :

Operation 1 : LPF- average two similar samples.
Operation 2 : HPF-subtract two similar samples.
When we subtract two similar samples (or HPF we get matrixes that are easier to compress due to their sparsity.

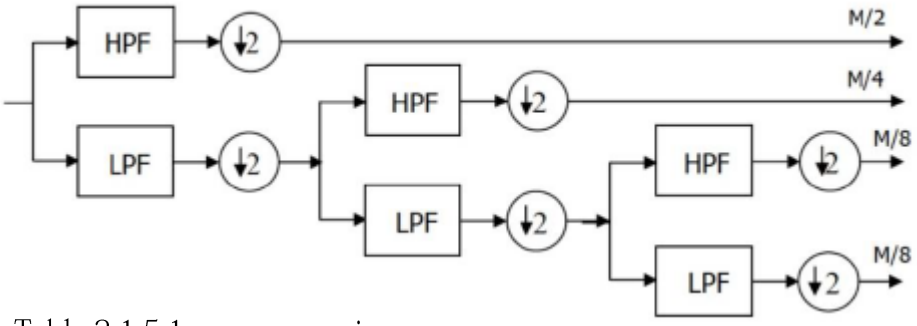


Table 2.1.5.1 – compression process

In this process compressing the image with the *'Haar'* wavelet-method we produced additional 3 matrix: V(vertical), H(horizontal), D(diagonal).

The vertical matrix represents the vertical change in the vertical axis (y-axis). i.e. - subtracting every two nearby rows.

The horizontal matrix represents the horizontal change in the horizontal axis (x-axis). i.e. - subtracting every two nearby columns.

The diagonal matrix represents the diagonal change in the vertical axis (y-axis) and the horizontal axis (x-axis). i.e. - subtracting every two nearby columns and then subtracting every two nearby rows..

:

2.2 Denoising

a. Read the beetle.jpg image, convert it to grayscale and normalize to [0; 1].

Figure 1.2.2.a.1

Explanation: We read the image, converted it to grayscale and normalized it as requested.

b. Create noisy image by adding gaussian noise to the original image using *imnoise()* function.

Figure 1.2.2.b.1

Explanation: We added a gaussian noise with zero mean and variance=0.01 using the function *'imnoise(Image, gaussian')'*

c. Remove the noise- use the functions *wthrmngr()* and *wdencmp()* with the wavelet decomposition as input, you may choose the other parameters.

d. Display and explain the result, use the function psnr() before and after the denoising to measure your result.

Figure 1.2.1.d.1

Explanation:

In order to find the wavelet decomposition, we used the function

```
[C,S] = wavedec2(img_noise,3,'haar');
```

Then we find level-dependent thresholds for the wavelet by using

```
THR = wthrmngr('dw2ddenoLVL','sqrtbal_sn',C,S);
```

With the threshold we can denoising the image by cleaning the wavelets coef. using :

```
[XC,CXC,LXC,PERF0,PERFL2] =
wdencmp('gbl',img_noise,'haar',3,THR(1,1),'h',2);
```

When the wavelet base is 'haar', its level(3), the threshold (THR), and parameter 's' or 'h' which determined if the filter will be hard 'h' or soft 's'.

After denoising the image, we can see that the image has some distortion. This cause due the uncorrelated noise that we applied. For example, in areas with low correlation (the leaf) we get more distortion.

We checked the peak SNR of the noisy image and the original image using *'psnr()'*, we got SNR ≅ 20.05
Then we check the peak SNR of the image after using the wavelet decomposition relative to the original image, we got SNR = 28.98. As expected we get better SNR after using the wavelet decomposition.

3 Hough Transform

In this exercise we will learn the Hough transform both for line and circles detection. You may NOT use the following MATLAB functions: hough(), houghlines(), imndcircles().

1. Read the *circuit.tif* image and mormalize it *to [0; 1].* This is a Matlab built-in image that you don't need to download (read it directly using *imread('circuit.tif))*.

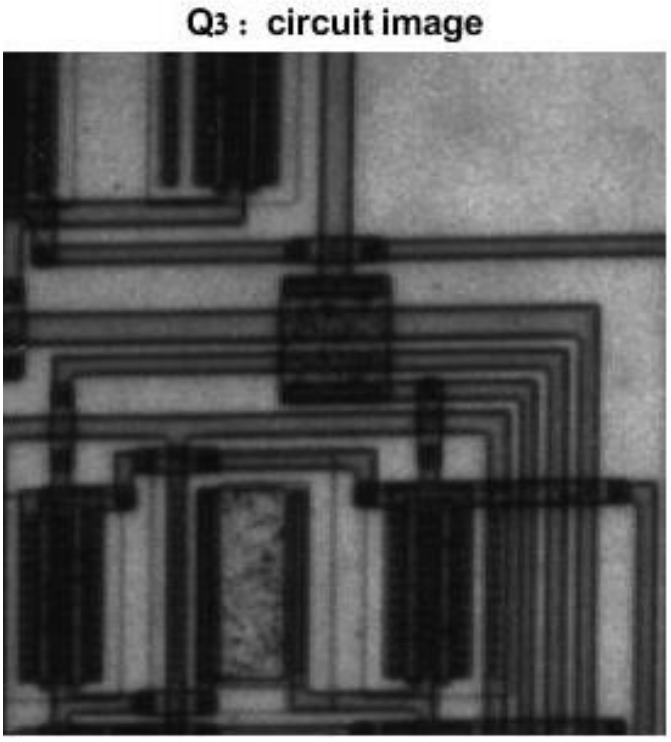


Figure 3.1.1

Explanation: We read the image, converted it to grayscale and normalized it as requested.

2. Extract the edges using MATLAB's *BW=edge(I)* and rotate it by 15 degrees using *BW=imrotate(BW,15).*

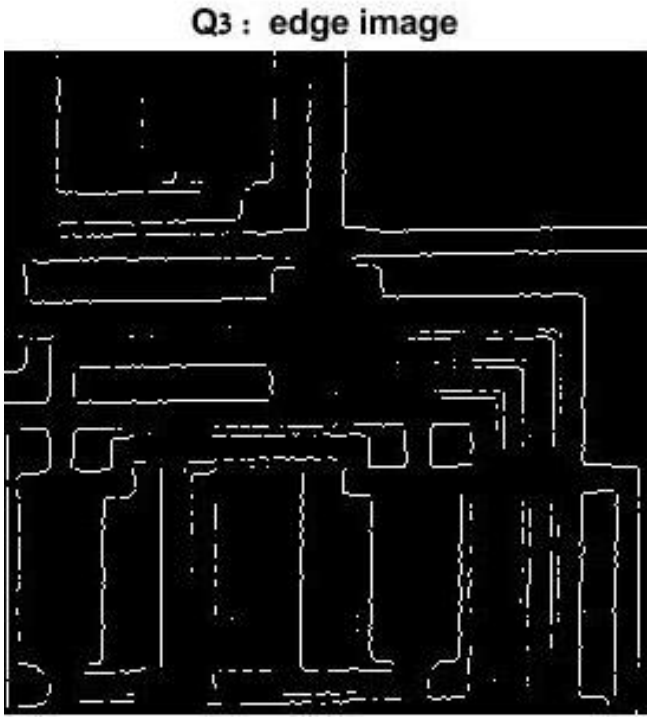


Figure 3.2.1

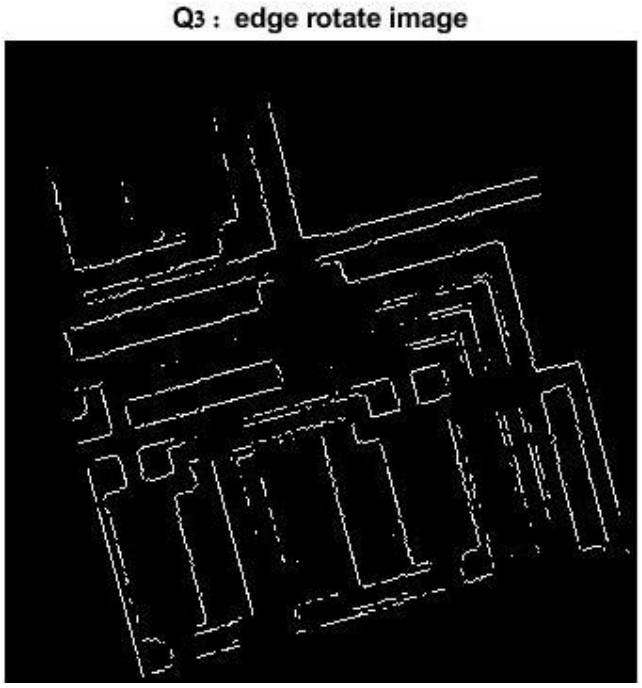


Figure 3.2.2

Explanation: As requested, we used the mentioned functions to find the edges of the image and rotate it by 15 degree.

3. Write your own *dip_hough_lines(BW)* function that calculates the Hough Matrix for finding lines.

In order to construct the Hough matrix follow these steps:

1. Create an empty matrix of size $\left[2\sqrt{M^2+N^2}+1\right]\times 181$ corresponding to values for $r \in \left[-\sqrt{M^2+N^2}, \sqrt{M^2+N^2}\right.$ and $\theta \in [-90, 90]$
   (a) For every white pixel in the image, $x, y \in BW$:
      i. For every value of θ:
         A. $r = x\cos(\theta) + y\sin(\theta)$
         B. Add 1 to the Hough matrix at the location corresponding to r, θ

Figure 3.3.1 – instruction for this algorithm

Explanation: For every white pixels we built 180 lines (for each 1 degree θ) that which pass at this pixel – for each of them we will accumulate the number of lines that match those (r,θ) in the Hough matrix. If we look at the end of the process at this matrix, each cell will represent the number of white pixels that the specific line (represent by the **(r, θ) values = also the cell position**) pass through them. It can help us estimate which of those lines are in high probability an edge in our image.

4. Display the Hough Matrix (using *imshow(M,[])*) and explain the result.

Figure 3.4.1

Explanation: We can see that the Hough matrix is mostly black (a lot of values near 0) which means that those lines match to a few points, hence they are not edges. The white points (represent lines) have the highest probability to be apart from edges.

5. Find the 5 most significant lines in the rotated BW image and plot them on the grayscale circuit image (you will need to rotate it by 15 degrees as well). Use MATLAB?s *houghpeaks* function to find the peaks of your Hough matrix.

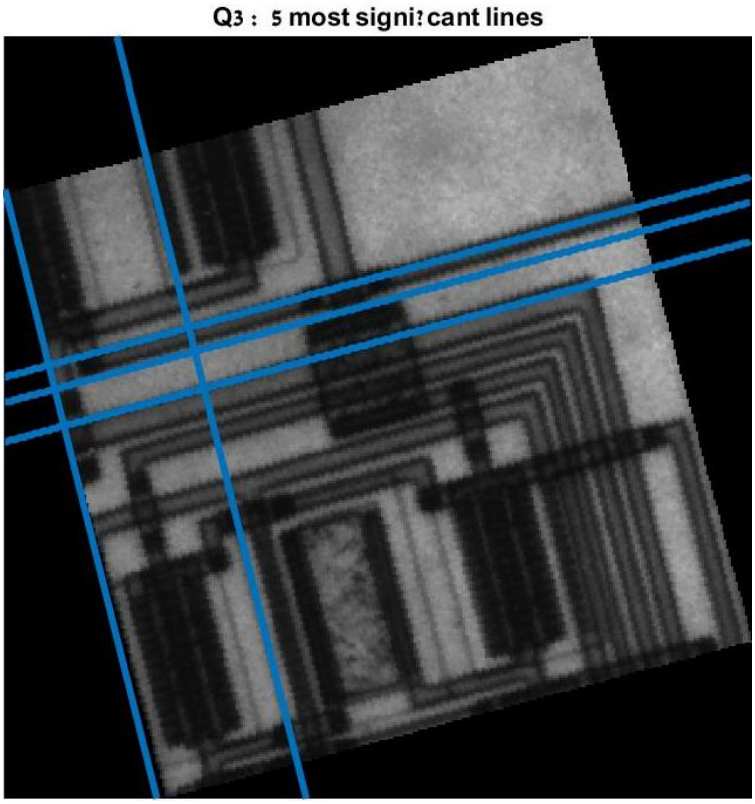


Figure 3.5.1

Explanation: We used the function *houghpeaks()* to find the 5 maximum cell values of the Hough Matrix and we display those lines by extracting the 5 positions (r, θ). The equation of a line can be found by the following :

$$r = xcos(\vartheta) + ysin(\vartheta) \rightarrow y = -xcot(\vartheta) + \frac{r}{\sin(\vartheta)}$$

6. Explain Read Matlab's built-in coins.png image and normalize it *to [0; 1].* Write your own *dip_hough_circles(BW)* function that calculates the Hough Matrix for finding circles. Repeat steps 1-5, but this timefind and plot the 5 most significant circles in the image. Note: This time your Hough matrix is 3D, so you can display one slice (2D image). Hint: you should bound your search for radius values *15<R<35.*

Figure 3.6.1

Explanation: We read the image and normalize it to [0,1]. The image contains 10 coins, after getting the edges of the image we got an image of 10 circles.

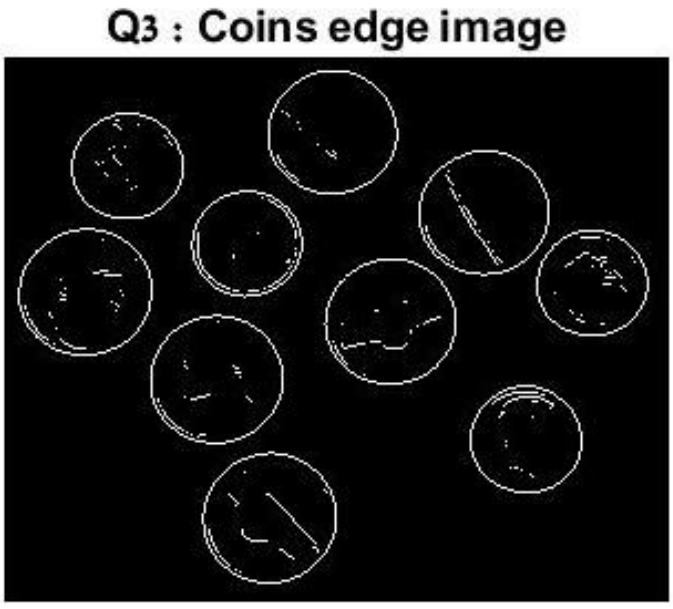


Figure 3.6.2

We wrote the function *'dip_hough_circles(BW)'* by the following algorithm :

In order to construct the Hough matrix **for circles** follow these steps:

1. Create an empty matrix of size $M \times N \times 21$ corresponding to the size of BW
   (a) For every pixel in the image, $x, y \in BW$:
      i. For every value of $R \in [15, 35]$:
         A. For every value of $\theta \in [-90, 90]$:
            $a = x - R\cos(\theta)$
            $b = y - R\sin(\theta)$
            Add 1 to the Hough matrix at the location corresponding to (a, b, R). (Don't forget to round the values)

Figure 3.6.3

```
peaks = zeros(5,3);
for i = 1:5
        [val,idx] = max(H(:));
        [idx1,idx2,idx3] = ind2sub(size(H),idx);
        peaks(i,:) = [idx1,idx2,idx3];
        H(idx1,idx2,idx3) = 0;
end
```

Figure 3.6.4

The idea is same as finding lines, but now we are dealing with 3 parameters that represents a circle equation, resulting 3 matrixes *(a,b,r)*.

For every pixel (which have an x and y coordinates = row,col) we created circles with the (a,b) centers and radius r, all of them are changing for each R. We accumulate the number correspond to *(a,b,r)* in the 3D Hough matrix.

We reconstruct the circles in the Hought matrix with the circle equation $(x-a)^2 + (y-b)^2 = r$, we showed an example with R=15,35 using imshow(), shown in Figures 3.6.5 and 3.6.6.

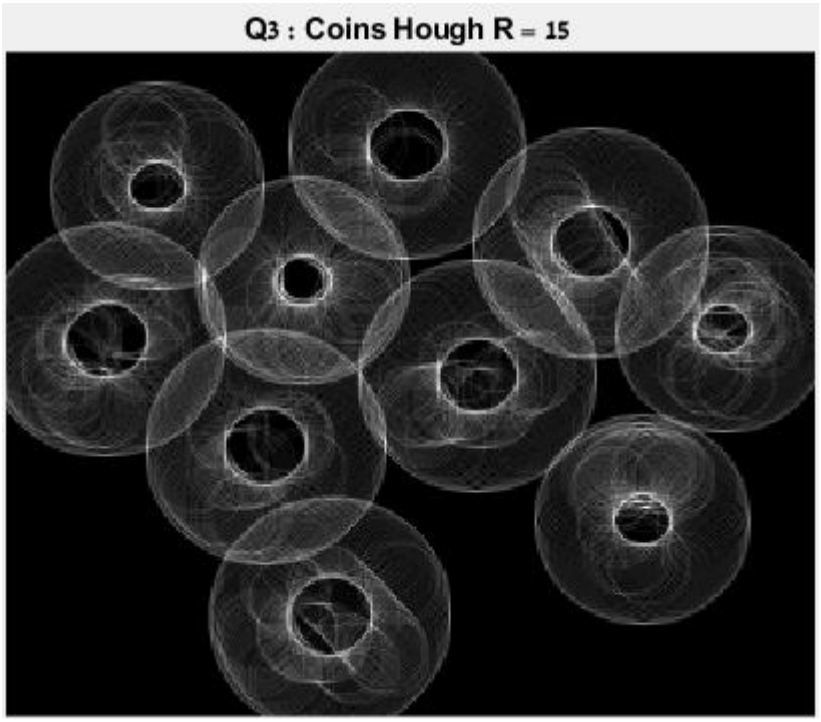


Figure 3.6.5

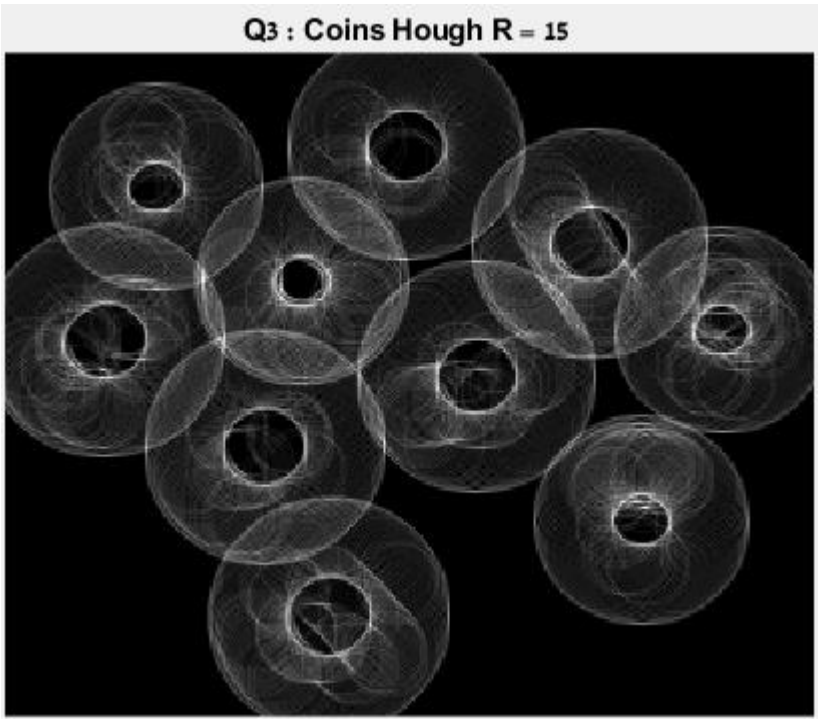


Figure 3.6.6

To find the most significant circles in the matrix we used the provided steps (Figure 3.6.4).
Next, we used the function `insertShape` to add the 5 most significant circles on the original image.

Figure 3.6.7

We can see that the algorithm found 5 circles represent 5 coins in our image.