
Table of Contents

Read Data	1
Q1 - Different Kernels	1
Q2 - Low Pass Filters	11
Q3 - Construct new filters from simple arithmetic.	11
Q4 - The Sobel Filter	13
Q5 - Median filter.	15
Q6 - Comparing filters	17
Q7 - Median Filter complexity	21
Q8 - Median Filter Implementation	21
Q9	22
Q10	23
Q11 - FFT and IFFT properties	25
Q12 - Filter Cameraman	25
Q13 - Filtering out patterns	28
Functions	32

Read Data

```
clear;
camera = double(imread('lab2/images_lab2/cameraman.png'));
wagon = double(imread('lab2/images_lab2/wagon.png'));
wagonnoise = double(imread('lab2/images_lab2/wagon_shot_noise.png'));
lines = double(imread('lab2/images_lab2/lines.png'));
circle = double(imread('lab2/images_lab2/circle.png'));
rectangle = double(imread('lab2/images_lab2/rectangle.png'));
```

Q1 - Different Kernels

We will apply three different kernels in the spatial domain with one sharpening, one smoothing and apply them in different sizes.

Firstly we begin with showing the original image that we will use.

```
figure;
imagesc(camera);
colormap(gray);
colorbar;
```



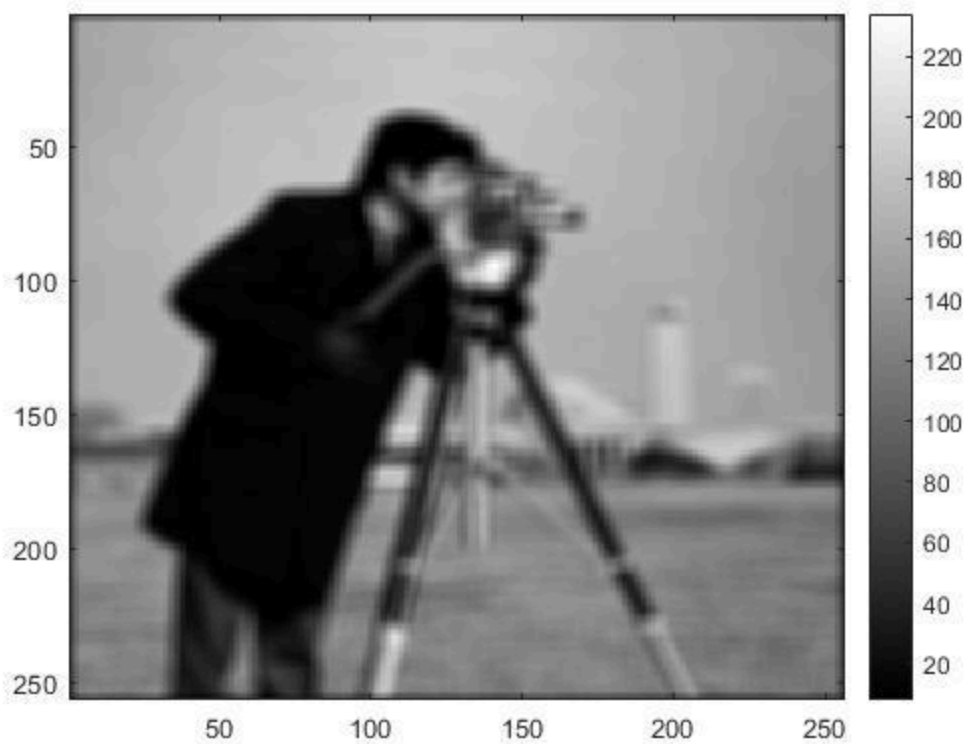
Now, let's introduce a mean filter of size 3×3 and apply the convolution using **imfilter**. The mean filtered image is shown below.

```
h1 = fspecial('average', 3);  
meancamera3 = imfilter(camera, h1);  
  
figure;  
imagesc(meancamera3);  
colormap(gray);  
colorbar;
```



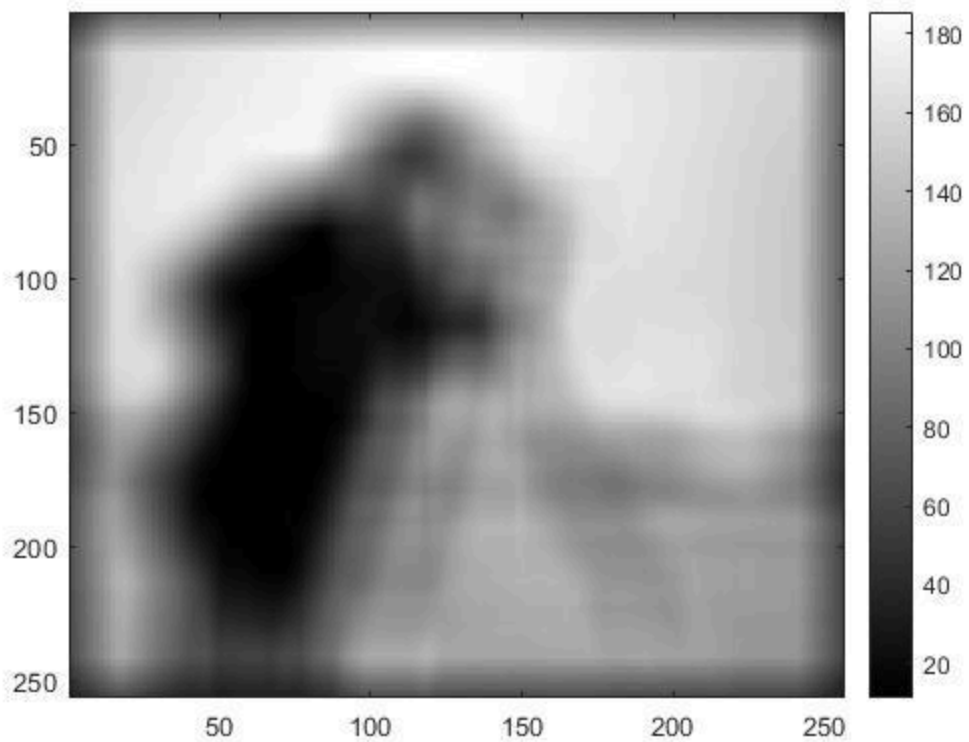
And using mean filter of size 7×7 we get

```
h2 = fspecial('average', 7);  
meancamera7 = imfilter(camera, h2);  
  
figure;  
imagesc(meancamera7);  
colormap(gray);  
colorbar;
```



And lastly, a 31×31 mean filter.

```
h3 = fspecial('average', 31);  
meancamera31 = imfilter(camera, h3);  
  
figure;  
imagesc(meancamera31);  
colormap(gray);  
colorbar;
```



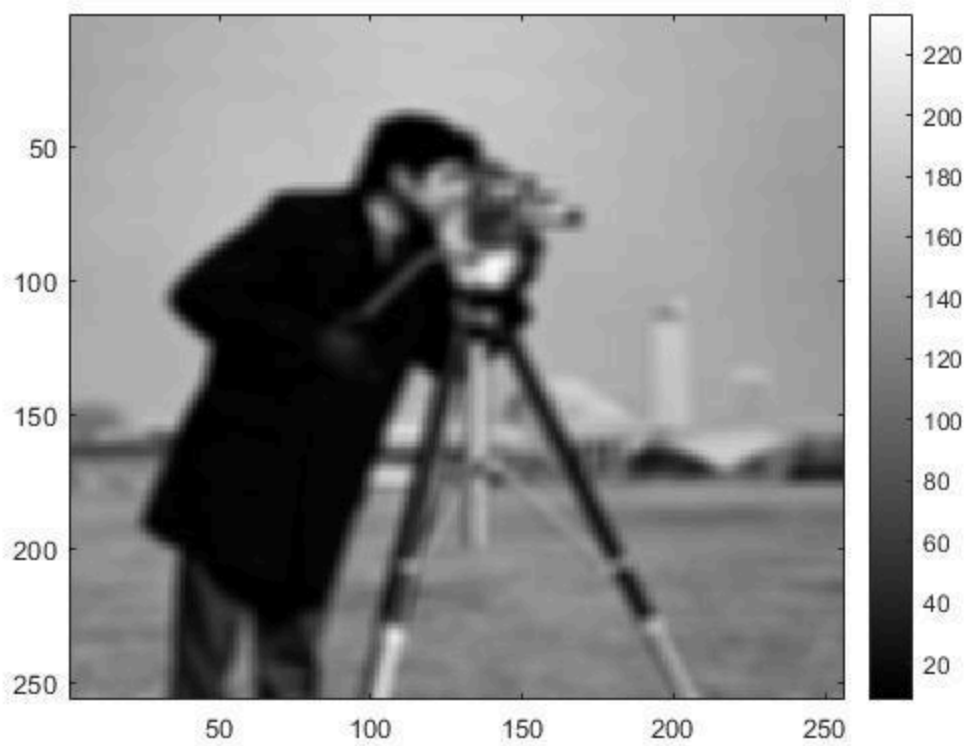
Lets introduce gaussian filters and perform the same calculations as above. Since the gaussian filter is based on the gaussian distribution We also have an additional parameter in addition to the size of the kernel, namely σ . We have assumed this to be $\sigma = 3$ for this exercise. We use the function `imgaussfilt` due to the documentation recommending to use that one instead of `imfilter`.

```
gausscamera3 = imgaussfilt(camera, 3, FilterSize=3);  
  
figure;  
imagesc(gausscamera3);  
colormap(gray);  
colorbar;
```



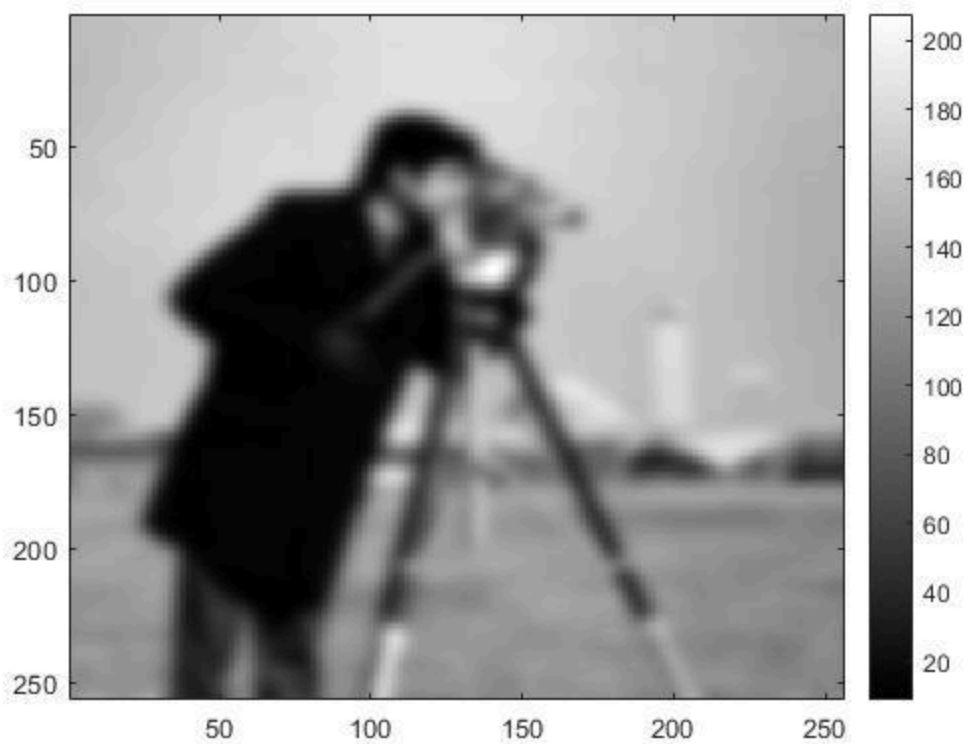
And using a gaussian filter size of 7×7 we get

```
gausscamera7 = imgaussfilt(camera, 3, FilterSize=7);  
  
figure;  
imagesc(gausscamera7);  
colormap(gray);  
colorbar;
```



And lastly using a gaussian filter size of 31×31 we get

```
gausscamera31 = imgaussfilt(camera, 3, FilterSize=31);  
  
figure;  
imagesc(gausscamera31);  
colormap(gray);  
colorbar;
```



For a sharpening (high pass) filter, we will use a unsharp masking using `imsharpen` function. Below we do this for a 3×3 filter.

```
siz = 3;  
  
msharpcamera3 = imsharpen(camera, 'Radius',siz);  
  
figure;  
imagesc(msharpcamera3);  
colormap(gray);  
colorbar;
```




And once again for the mask of size 7×7

```
siz = 7;  
  
msharpcamera7 = imsharpen(camera, 'Radius',siz);  
  
figure;  
imagesc(msharpcamera7);  
colormap(gray);  
colorbar;
```



And lastly, one last time for the mask of size 31×31

```
siz = 31;  
  
msharpcamera31 = imsharpen(camera, 'Radius',siz);  
  
figure;  
imagesc(msharpcamera31);  
colormap(gray);  
colorbar;
```



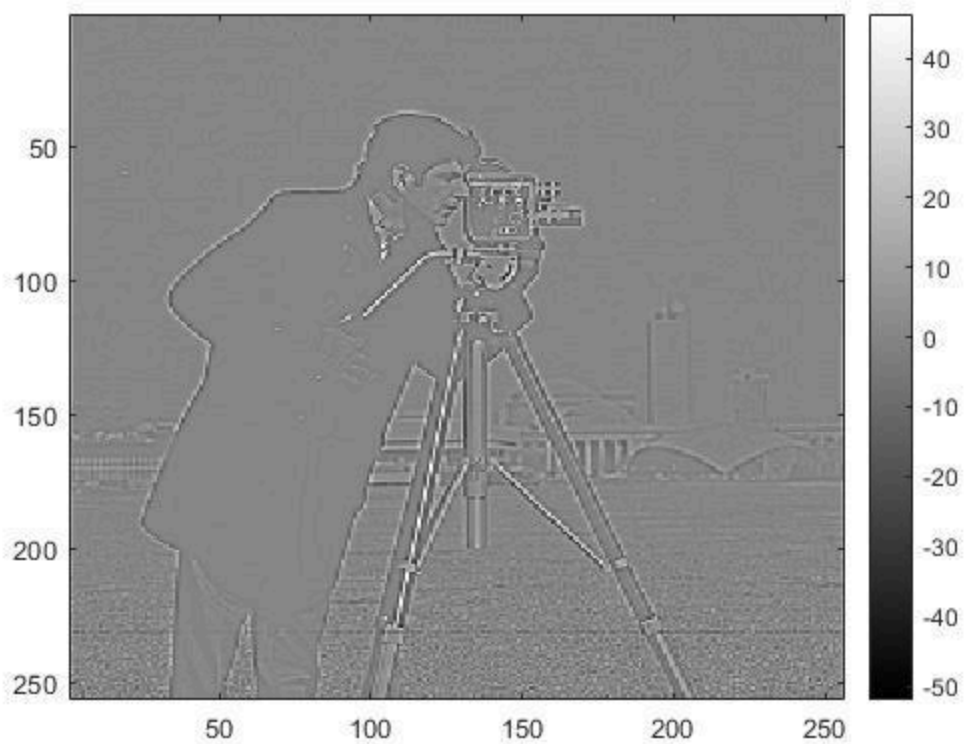
Q2 - Low Pass Filters

In `fspecial` we have the filters **average**, **disk** and **gaussian**. All of which are instances of low-pass filters. They calculate new pixel values by incorporating neighbouring pixel values and "average" the rapid changes. The **average** filter does this by calculating the sample mean of the defined neighbourhood, **disk** uses the same notion but uses a circular notion of distance from the center and the **gaussian** uses a gaussian distribution where pixels far away affect the new pixel value less.

Q3 - Construct new filters from simple arithmetic.

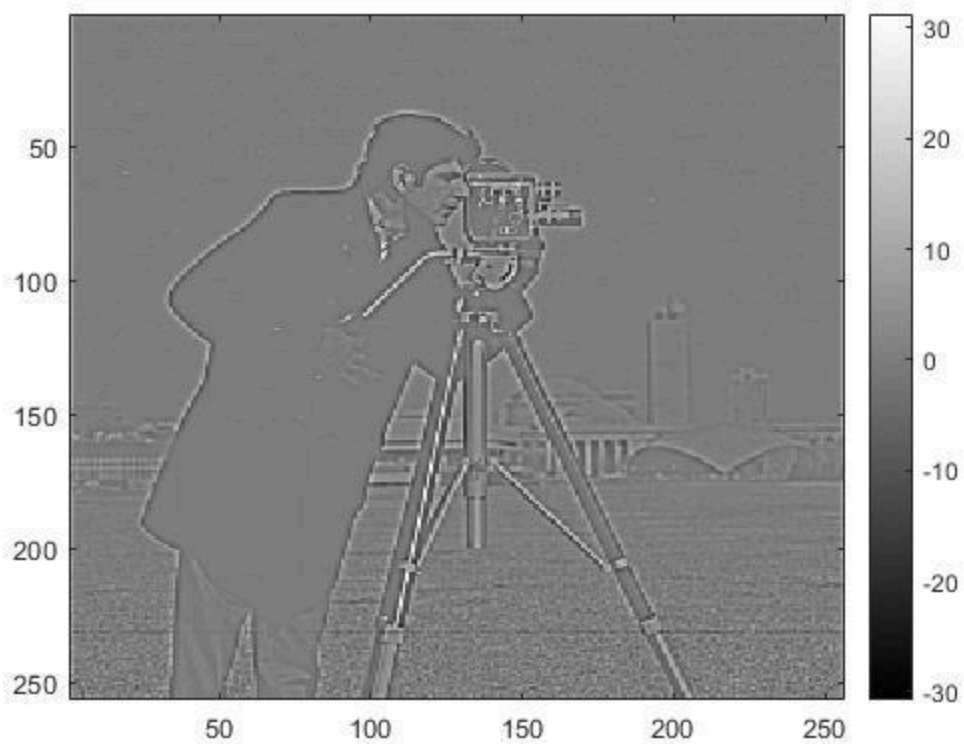
A low pass filter can be denoted $lp(x, y)$. We know from the textbook that a high-pass filter may be constructed by subtracting lowband filter from identity filter;

```
flowpass = fspecial('gaussian',3);  
fhighpass = -flowpass;  
fhighpass(2,2) = fhighpass(2,2)+1;  
figure;  
imagesc(imfilter(camera,fhighpass));  
colormap(gray);  
colorbar;
```



Low pass filter removes high frequencies and high pass filter removes low frequencies. Therefore we can create a band pass filter by applying both low and high pass filter to an image

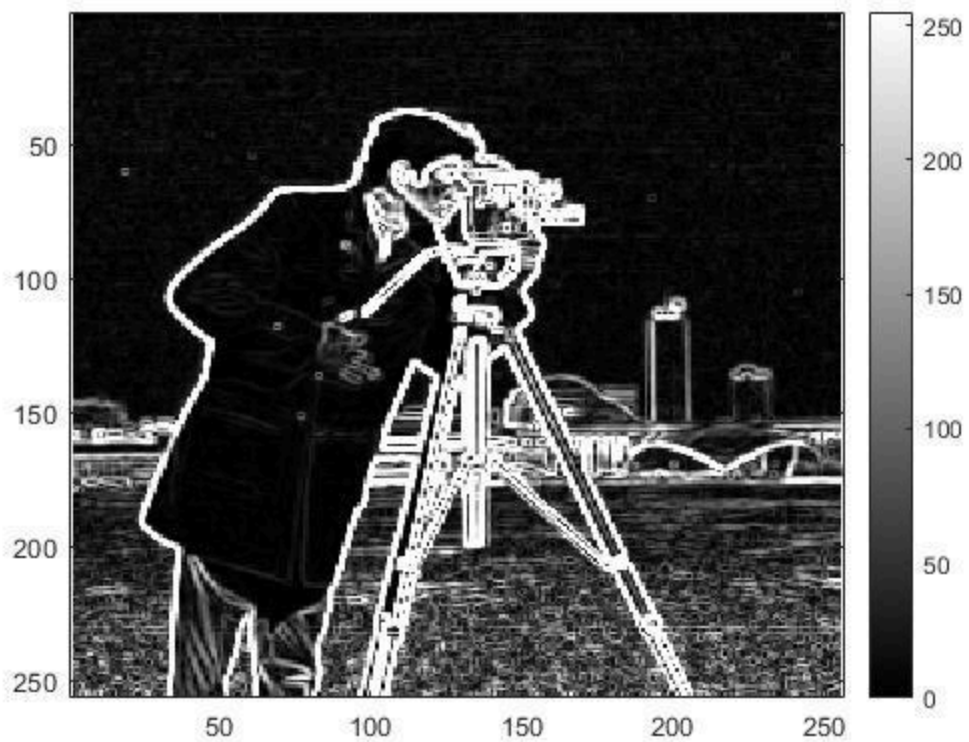
```
figure;  
imagesc(imfilter(imfilter(camera,f_lowpass),f_highpass));  
colormap(gray);  
colorbar;
```



Q4 - The Sobel Filter

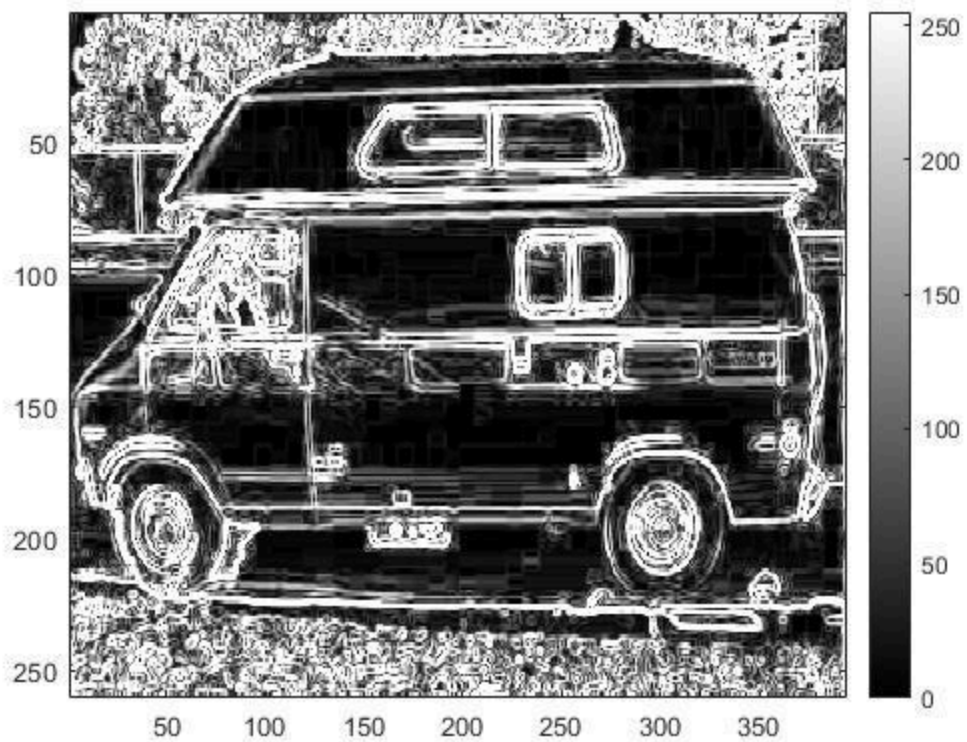
Lets apply the filter on the cameraman image. Since this is a directional kernel, some arithmetic must be done on the two images.

```
h7 = fspecial('sobel');  
camerasobelx = imfilter(double(camera), h7);  
camerasobely = imfilter(double(camera), h7');  
  
figure;  
imagesc(uint8(sqrt(camerasobelx.^2 + camerasobely.^2)));  
colormap(gray);  
colorbar;
```



We also do the same of the wagon image

```
wagonsobelx = imfilter(double(wagon), h7);  
wagonsobely = imfilter(double(wagon), h7');  
  
figure;  
imagesc(uint8(sqrt(wagonsobelx.^2 + wagonsobely.^2)));  
colormap(gray);  
colorbar;
```

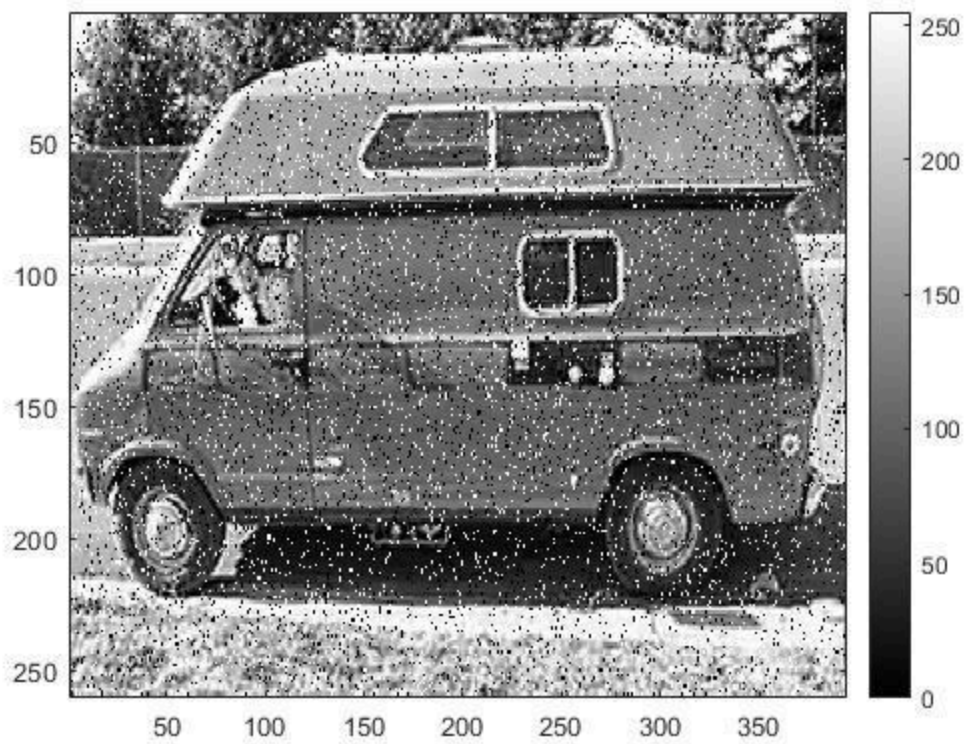


Q5 - Median filter.

We will perform median filtering on the image. Let's plot both the original image and the filtered one.

Below is the original image with pepper and salt noise.

```
figure;  
imagesc(wagonnoise);  
colormap(gray);  
colorbar;
```



And the median filtered image.

```
figure;  
imagesc(medfilt2(wagonnoise));  
colormap(gray);  
colorbar;
```




Q6 - Comparing filters

Let us compare the gaussian filtered image with the median filtered image and mean filtered image.

Below is the original image.

```
figure;  
imagesc(wagonnoise);  
colormap(gray);  
colorbar;
```



And now we use the median filter

```
figure;  
imagesc(medfilt2(wagonnoise));  
colormap(gray);  
colorbar;
```



The mean filter of size 3×3

```
figure;  
h8 = fspecial('average', 3);  
imagesc(imfilter(wagonnoise, h8));  
colormap(gray);  
colorbar;
```



And lastly, the gaussian filter with $\sigma = 2$

```
figure;  
imagesc(imgaussfilt(wagonnoise, 1));  
colormap(gray);  
colorbar;
```



What we see is that for this particular noise, the median filter works really well, this is due to the median being a "robust" estimator of centrality and is not easily skewed by outliers. So when there appears some extreme values in the neighbourhood of the pixel, the median still estimates a more appropriate value as it is not skewed by the extremes.

Q7 - Median Filter complexity

The median filter is a nonlinear filter and specifically an order-statistic filter. Which means that before calculating the new pixel value, the pixel values in the neighbourhood must be sorted and it can't be applied as a conv matrix filter.

Q8 - Median Filter Implementation

Below is my implementation of a median filter.

```
m = mymedianfilt(wagonnoise);

figure;
imagesc(m);
colormap(gray);
colorbar();

% Median Filter
% function m = mymedianfilt(img)
%     [a, b] = size(img);
```

```

%     newI = zeros(a, b, 'double');
%     padded = padarray(img, [1, 1], median(img(:)));
%
%     for i = 2:a
%         for j = 2:b
%             med = zeros(1, 9, 'double');
%             it = 0;
%             for k = 1:3
%                 for l = 1:3
%                     it = it + 1;
%                     med(it) = padded(i+k-1, j+l-1);
%                 end
%             end
%             newI(i,j) = median(med);
%         end
%     end
%     m = newI;
% end

```



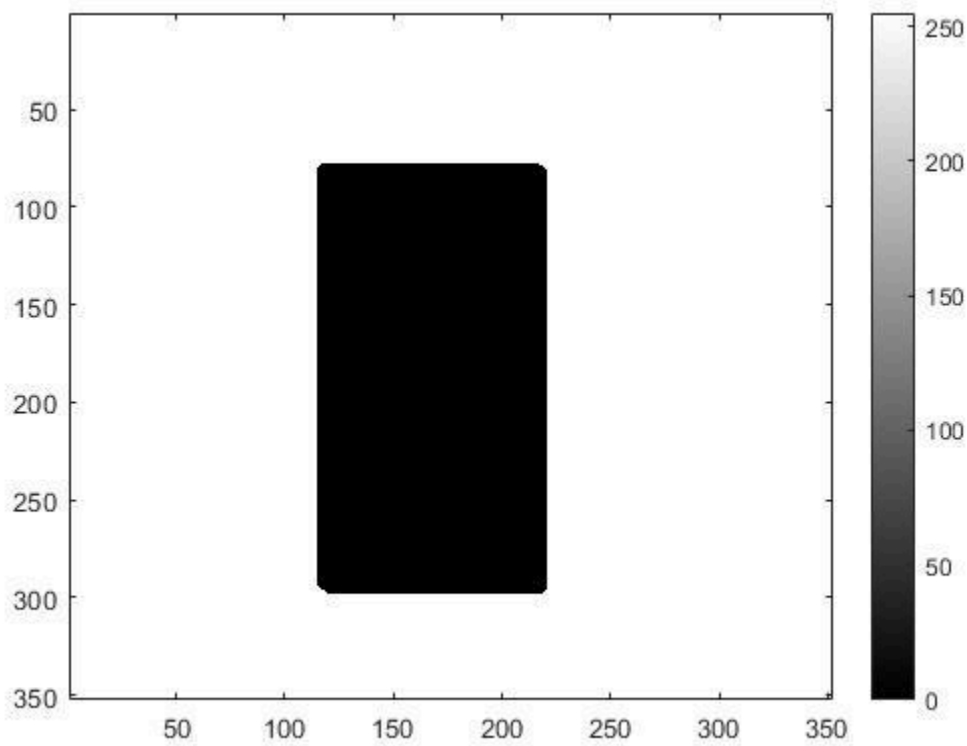
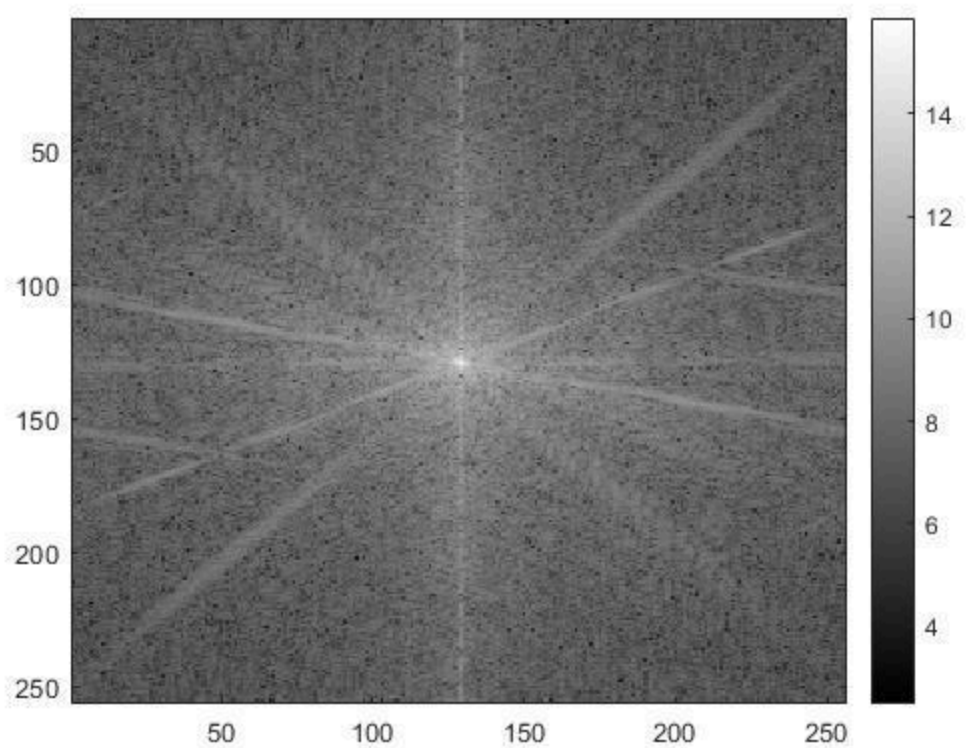
Q9

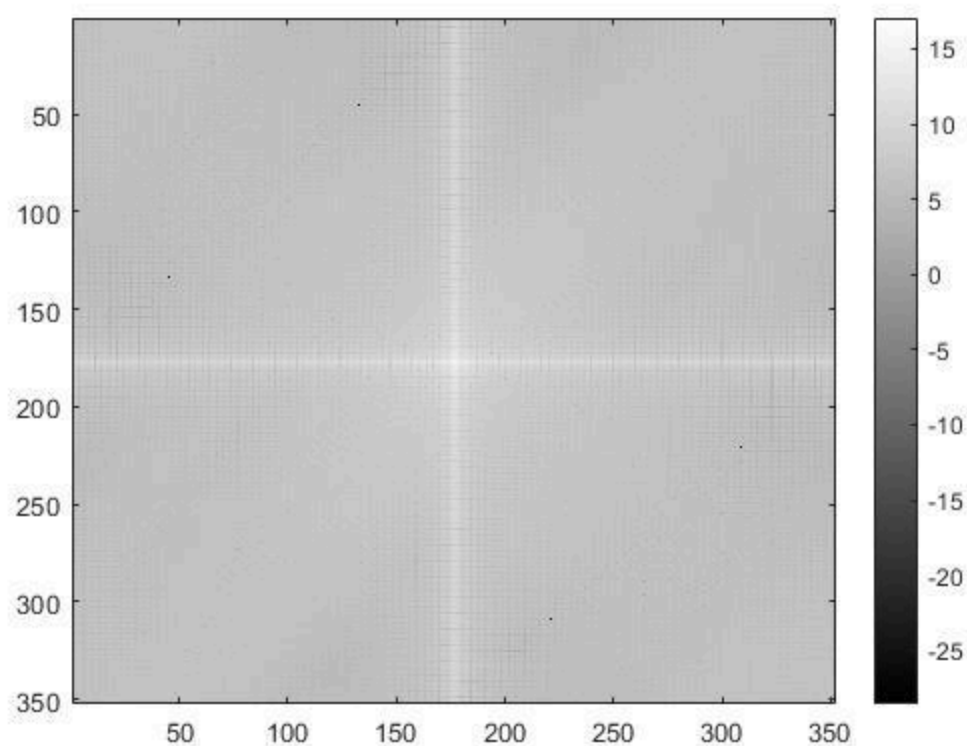
You get a black border due to MATLAB having to access pixel values outside the border of the image, and the default is to use zero padding.

Q10

```
f = fftshift(fft2(camera));  
figure; imagesc(camera); colormap(gray); colorbar();  
figure; imagesc(log(abs(f))); colormap(gray); colorbar();  
  
f = fftshift(fft2(rectangle));  
figure; imagesc(rectangle); colormap(gray); colorbar();  
figure; imagesc(log(abs(f))); colormap(gray); colorbar();
```







Q11 - FFT and IFFT properties

We see that the transform below has symmetric values.

```
f = fftshift(fft2(rand(1,5)));
f;
```

Now we will do some filtering. We set $f(1, 2) = 0$

```
f(1, 2) = 0;
im = ifft2(ifftshift(f));
im;
```

```
% Now we will do the filtering symmetrically. We set  $f(1, 4) = 0$ 
f(1, 4) = 0;
im = ifft2(ifftshift(f));
im;
```

We see that if we don't filter symmetrically, we get a complex valued image.

Q12 - Filter Cameraman

We add a circular filter and only pass low frequencies. Below is the image before filtering

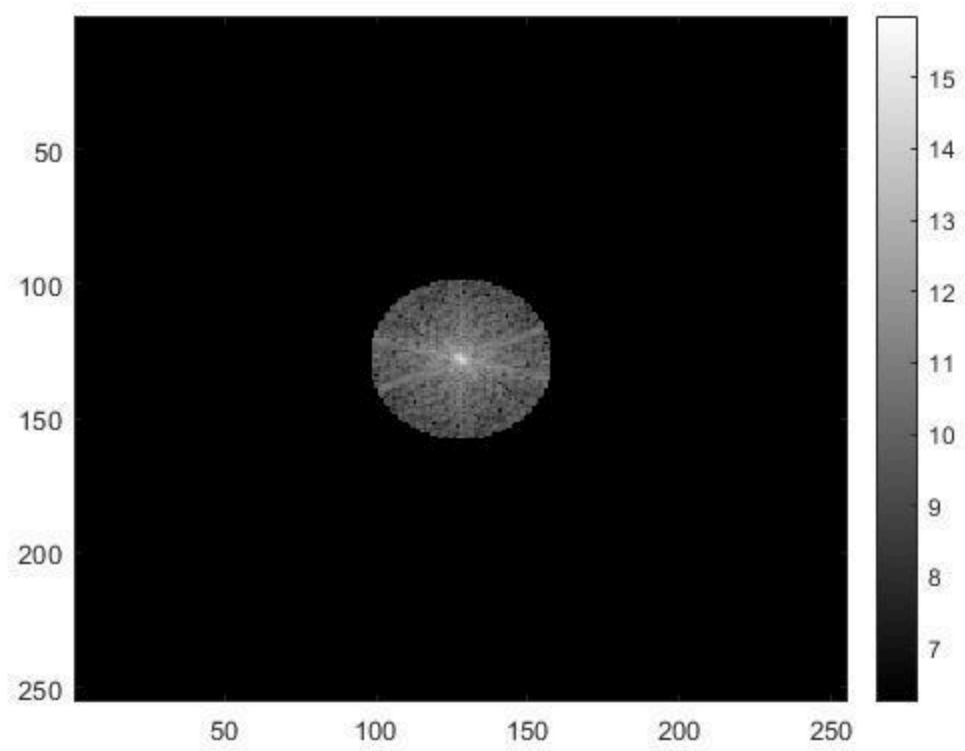
```
cameraFFT = camera(1:255, 1:255);
```

```
figure; imagesc(cameraFFT); colormap(gray); colorbar();
```



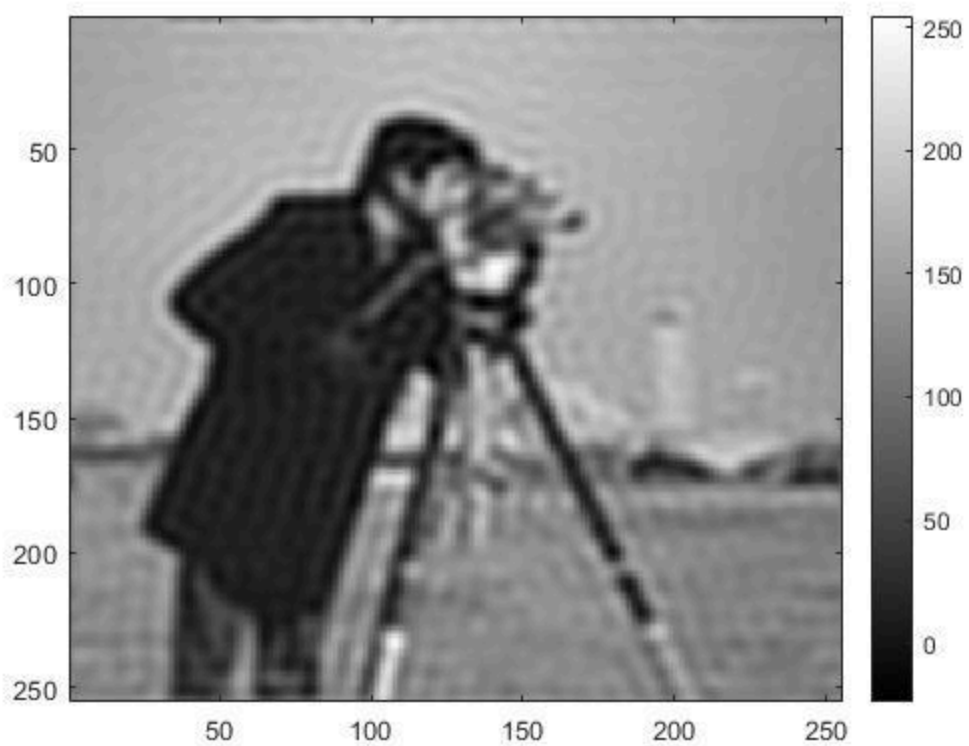
And we filter out the high frequencies.

```
r = 30;  
f = fftshift(fft2(cameraFFT));  
[xgrid, ygrid] = meshgrid(1:size(f,2), 1:size(f,1));  
mask = ((xgrid-128).^2 + (ygrid-128).^2) >= r.^2;  
f(mask) = 0;  
figure; imagesc(log(abs(f))); colormap(gray); colorbar();
```



And finally we reconstruct the image.

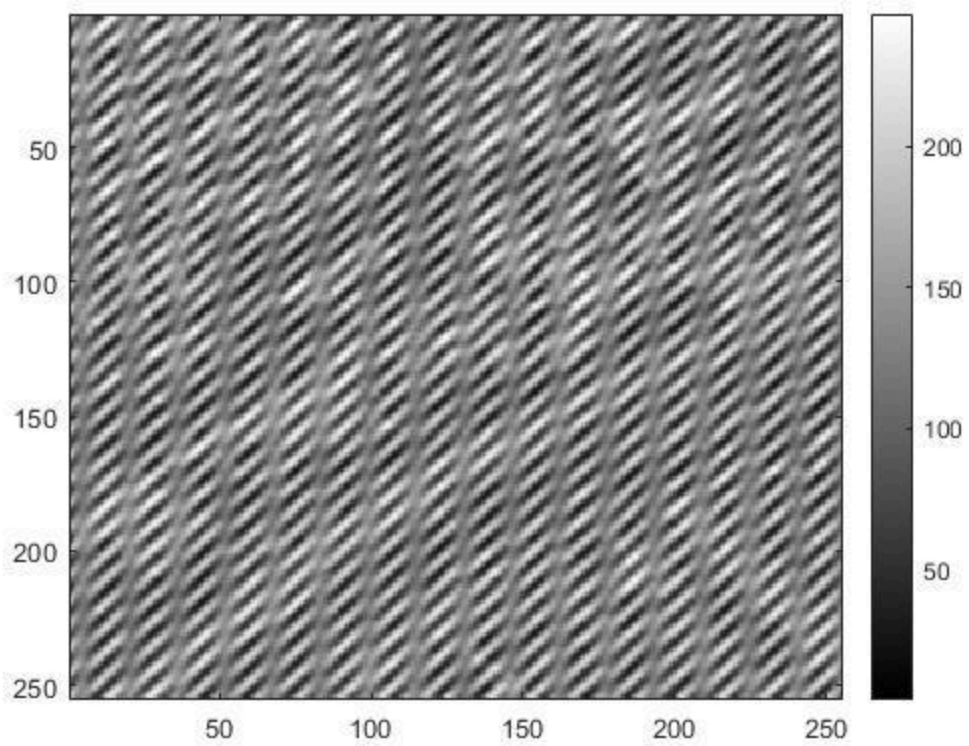
```
newim = ifft2(ifftshift(f));  
figure; imagesc(newim); colormap(gray); colorbar();
```



Q13 - Filtering out patterns

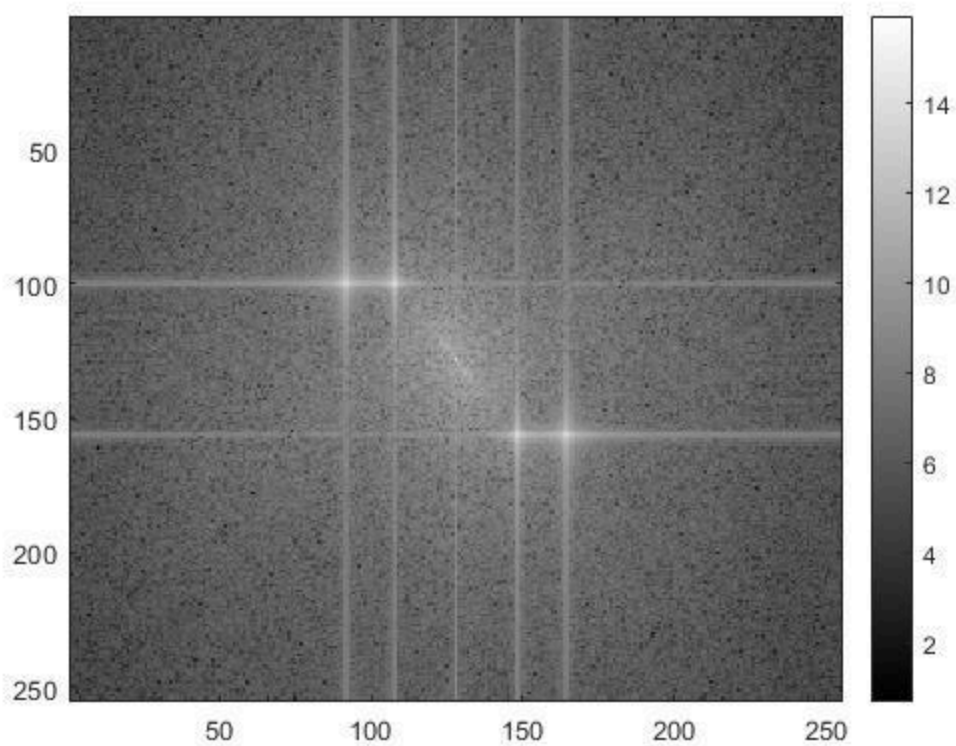
The image we will filter is shown below.

```
freqdist = double(imread('lab2/images_lab2/freqdist.png'));  
freqdist = freqdist(1:255, 1:255);  
figure; imagesc(freqdist); colormap(gray); colorbar();
```



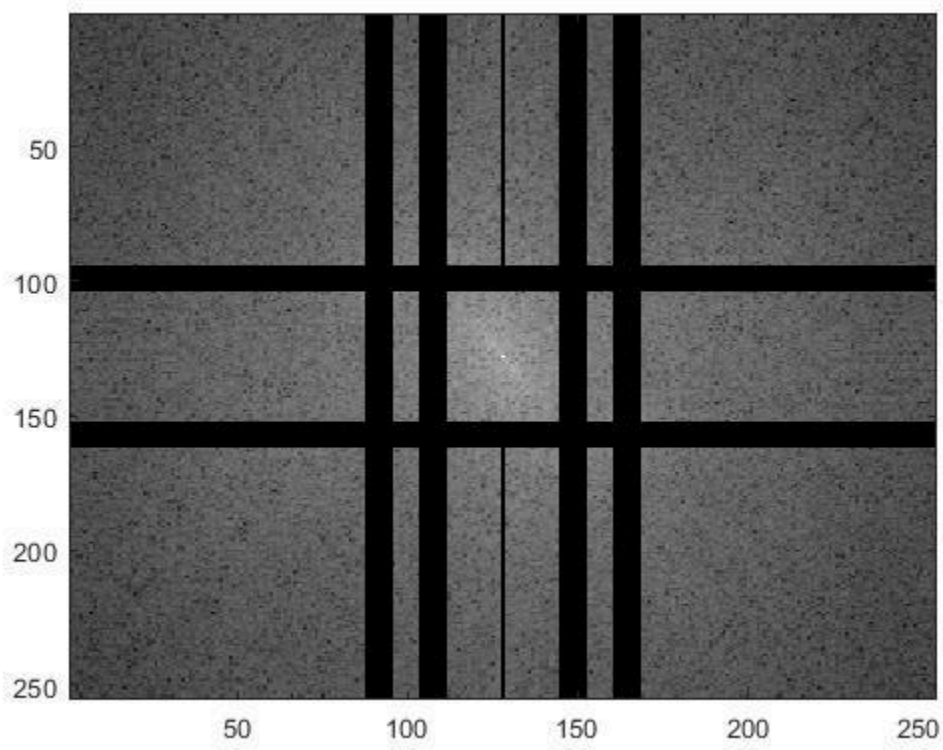
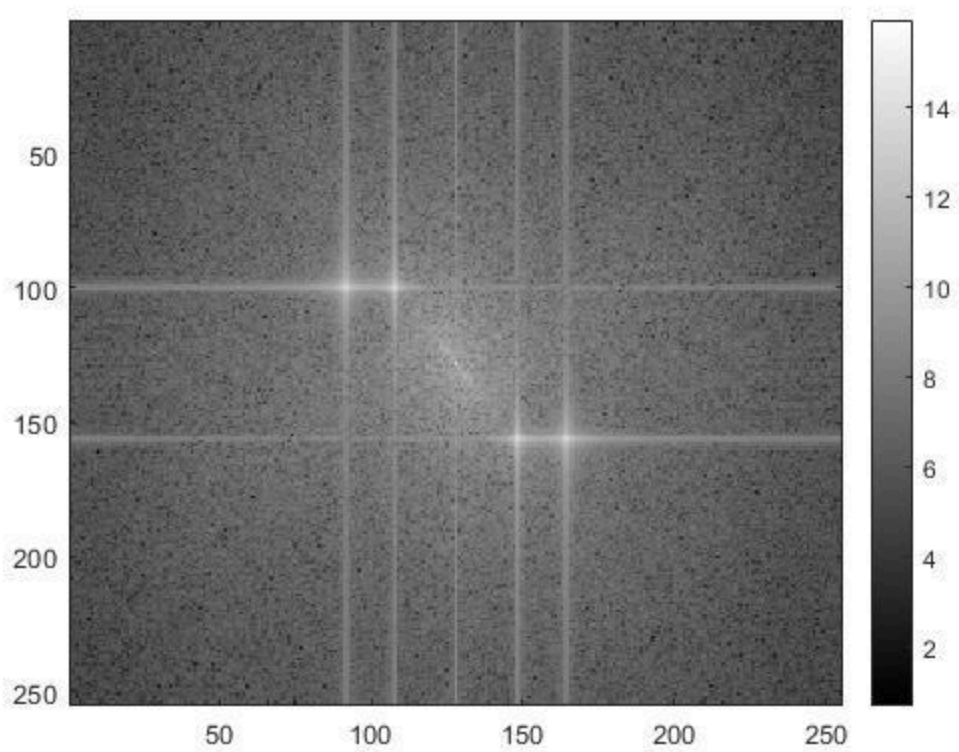
Frequency representation of image. We see some artifacts. Lets try to remove them.

```
f = fftshift(fft2(freqdist));  
figure; imagesc(log(abs(f))); colormap(gray); colorbar();
```



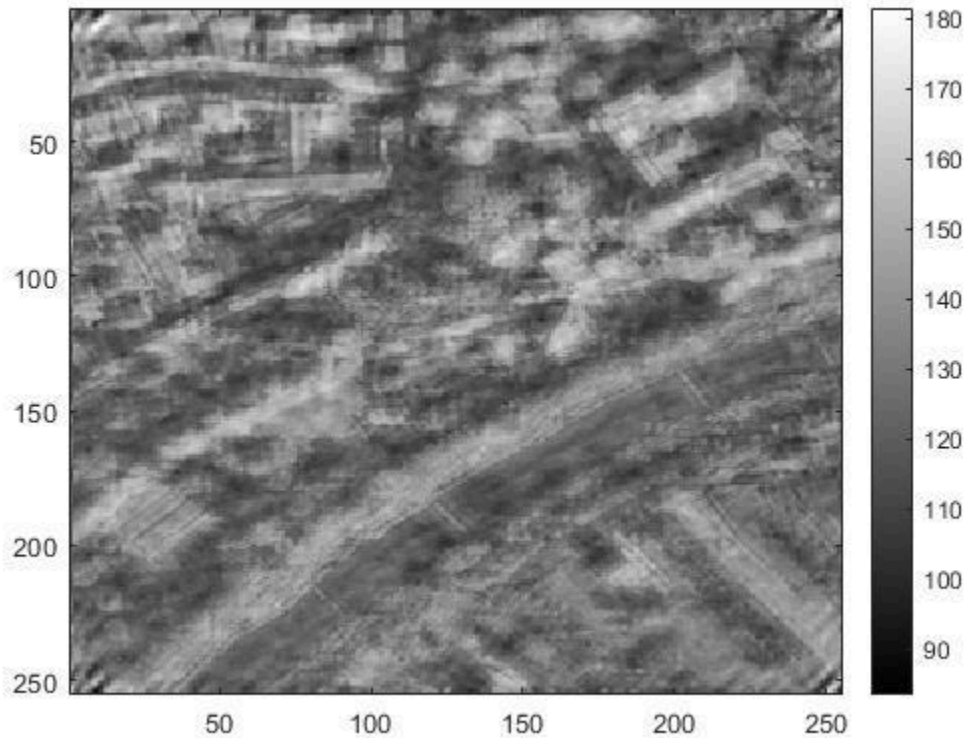
Remove unwanted pattern. We do this by removing all bright lines which relate to the regular pattern that we see on the image. That is the wave pattern we want to keep.

```
f = fftshift(fft2(freqdist));
figure; imagesc(log(abs(f))); colormap(gray); colorbar();
f(95:103,1:255) = 0;
f(153:161,1:255) = 0;
f(1:255,104:111) = 0;
f(1:255,end-111+1:end-104+1) = 0;
f(1:255,88:95) = 0;
f(1:255,end-95+1:end-88+1) = 0;
f(1:100,128:128) = 0;
f(156:end,128:128) = 0;
figure; imagesc(log(abs(f))); colormap(gray);
```



And lastly we show the image that has been reconstructed.

```
newim = ifft2(ifftshift(f));  
figure; imagesc(newim); colormap(gray); colorbar();
```



Close all figures

```
close all;
```

Functions

Median Filter

```
function m = mymedianfilt(img)  
    [a, b] = size(img);  
    newI = zeros(a, b, 'double');  
    padded = padarray(img, [1, 1], median(img(:)));  
  
    for i = 2:a  
        for j = 2:b  
            med = zeros(1, 9, 'double');  
            it = 0;  
            for k = 1:3  
                for l = 1:3  
                    it = it + 1;  
                    med(it) = padded(i+k-1, j+l-1);  
                end  
            end  
            newI(i, j) = median(med);  
        end  
    end
```

```
        end
    end
    newI(i,j) = median(med);
end
end
m = newI;
end
```

Published with MATLAB® R2021a