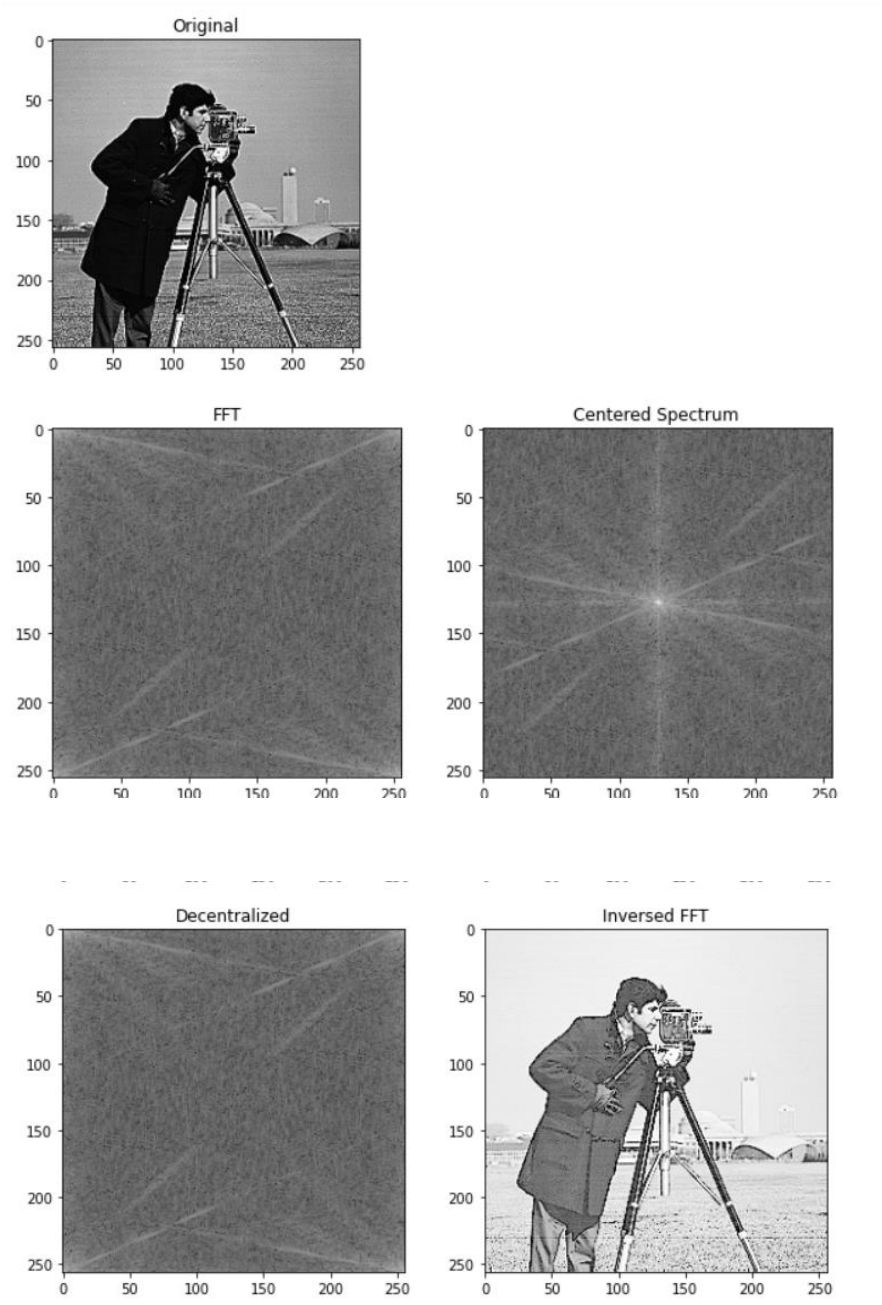
Filtering in the frequency domain

Part I

We use the Numpy library to perform our Fourier transformations. First, we perform a 2-dimensional discrete Fourier Transform. Then, we shift the zero-frequency component to the center of the spectrum. Next, we perform the inverse of it and compute the 2-dimensional inverse discrete Fourier Transform. Every step's image is as follows:

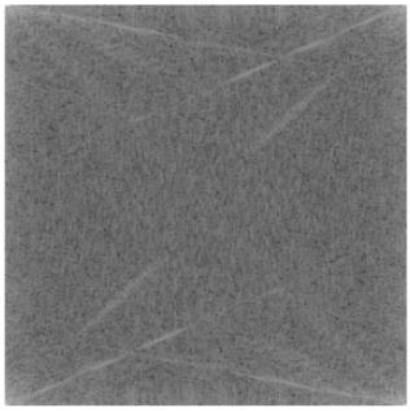


Part II

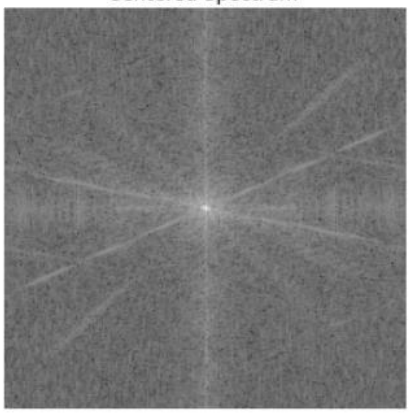
We pad our image with the nearest rows and columns in this part. As it was not specified in the question, we chose the size to be 10. The previous procedure was performed on this new image again, and the results are as follows:

Padded image

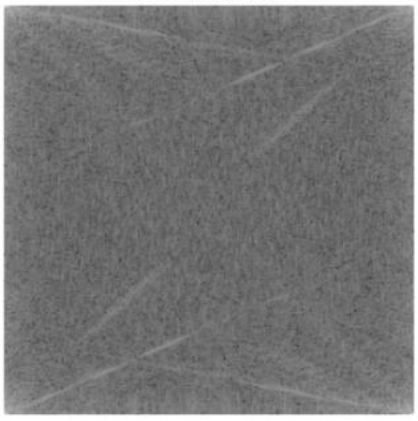
FFT



Centered Spectrum



Decentralized

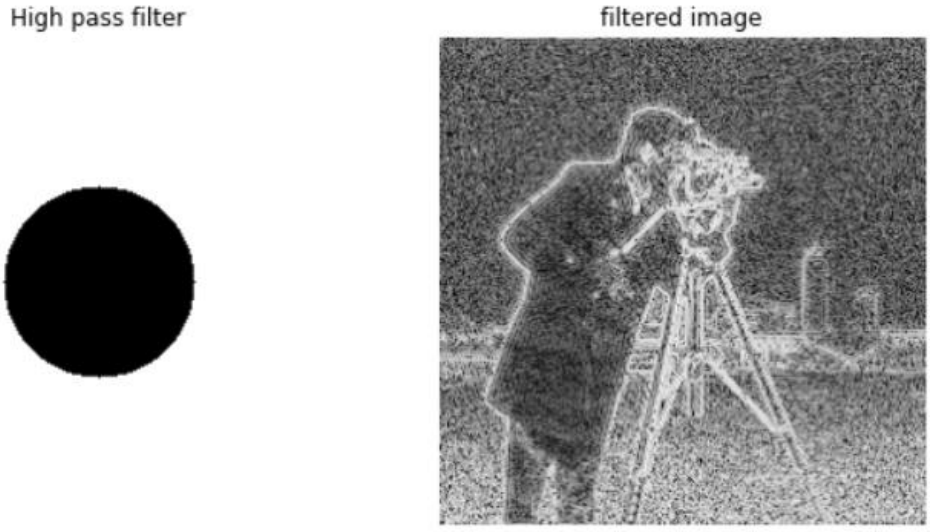


Inversed FFT

Part III

This part was a more challenging one while the code was implemented manually. First, we Computed the Ideal low-pass filter matrix. For this purpose, we chose our threshold (D0) to be 50. Then, we multiplied our centered Fast Fourier Transformation image by the filter, and using inverse transformations, we rebuilt the image. The results were as follows:

We perform the same steps for high-pass filtering but with the reverse of Ideal Low-Pass Filter matrix. As you can see, high-pass and low-pass filtering focus on exact opposite parts of the image.



Part IX

We performed Gaussian smoothing (Gaussian blurring) using different kernels in this part. We used the cv2 library for this task. Unsurprisingly, as the kernel increased, the image became more blurred. The results for different kernel sizes are as follows:

Original

Kernel = 3

Kernel = 5

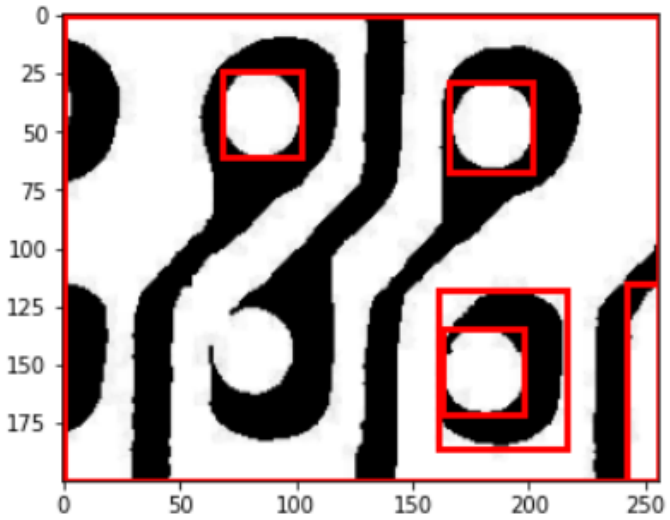
Kernel = 7

Kernel = 9

Morphology Operators

We were asked to count the number of holes and their corresponding diameters in the circuit. To count the holes, we used cv2.contour. First, we needed to invert our image to be able to use the framework. Then, to reform the circles to better and more precise ones, we eroded and dilated them with a kernel of the same size to preserve the actual size. Then, we count the contours larger than the threshold as our holes to avoid noise. Then, we drew a box around the found holes. The result was as follows:

```
Diameter of hole 1 : 85.00019836425781
Diameter of hole 2 : 37.88866424560547
Diameter of hole 3 : 37.811805725097656
Diameter of hole 4 : 323.4596252441406
Diameter of hole 5 : 70.30641174316406
Diameter of hole 6 : 37.866920471191406
```

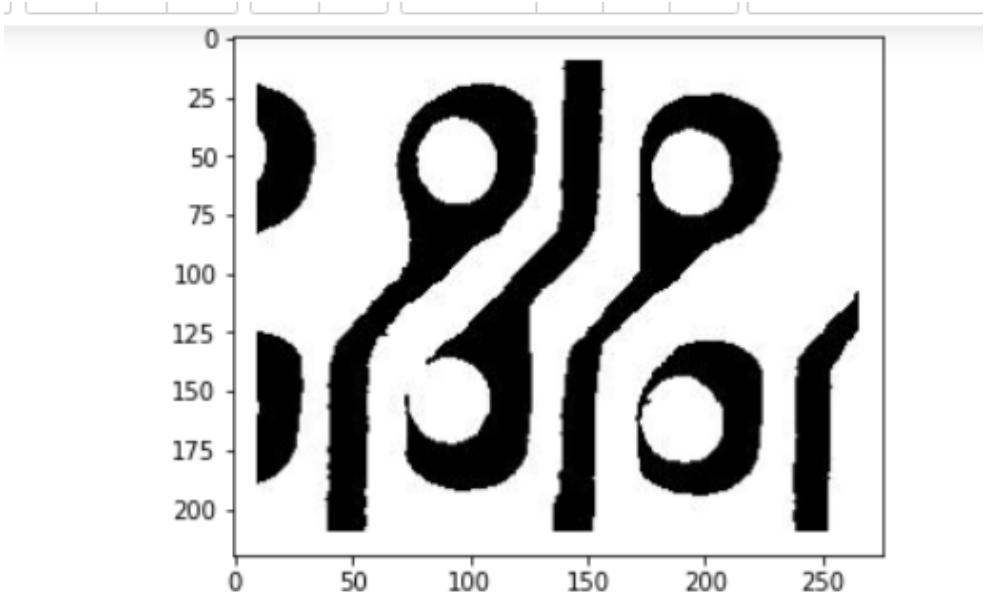


As one can see, the result was not totally accurate as we counted some holes wrongly and missed one. But the gain was in finding the real diameters of holes, which were around 38.

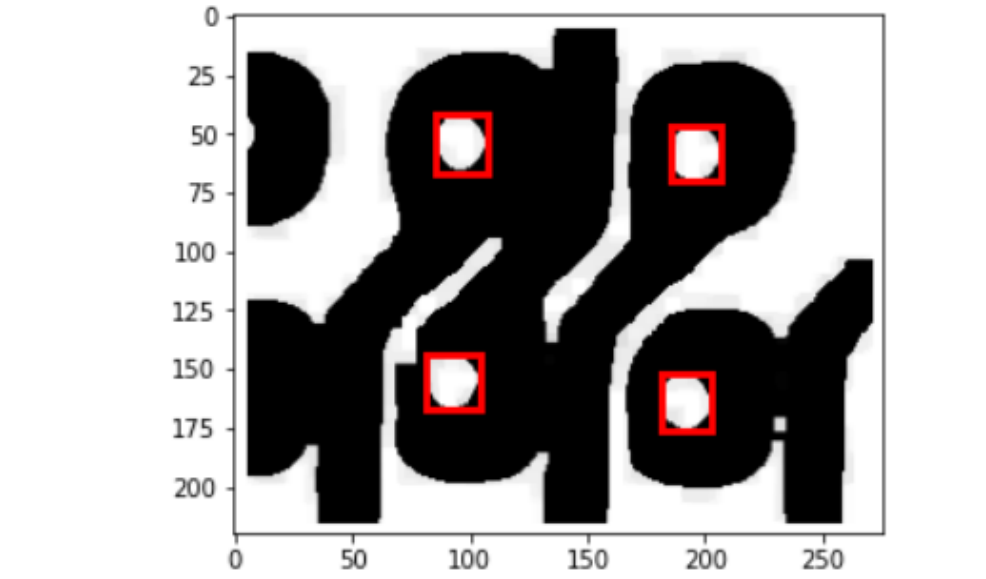
To get accurate results, we should change our image to complete that one hole and avoid counting margins as holes.

To do so, we zero-padded our image to prevent miscountings. Also, we performed erosion and dilation with different kernels to complete that one circle. Because it makes the holes smaller and the margins thicker. The image scales changed by doing so, but the holes are now distinctly recognizable.

The original padded image and the final result and their diameters are as follows:




```
Diameter of hole 1 : 24.18340492248535
Diameter of hole 2 : 26.32655143737793
Diameter of hole 3 : 23.482954025268555
Diameter of hole 4 : 24.51757049560547
```



Now we have done the task. We have 4 holes, each with a diameter of 38.