## Project Option

Requirements:

1. You are comfortable programming in Python
2. If you are an undergraduate, you may have only one partner. If you are a graduate student, you will work alone.
3. You can only use the libraries provided
4. If your code does not run when evaluated, it is **an automatic zero**. Make sure all the tests run.

Evaluation:

Programming 50%: In this project, there are low-level implementations of image processing functions. Resources will be provided, either mathematical formulas or visual resources for guiding you. Furthermore, we have added test cases for you to evaluate your process. There are also high-level implementations (drivers) where you will use the image processing functions in order to analyze the process and answer the final report.

Report 50%: Use the provided report template. Follow the steps that you see adequate to produce the requested images. Add the result images, provide analysis, and answer the questions. Use the driver's scripts to produce this information.

## Programming

Implement the functions located on the python scripts under the "src" folder. In the code provided, there are automated tests for you to track your progress. This test (and additional ones) will be used to evaluate your code. Use them as guidance as to what each function should return. You are more than welcome to explore and use them as a way to understand how each function will be called and what result is expected. However, do not make any changes in the tests. If you change the tests to make your code pass, it will fail when we run them with our tests. The drivers do not have automatic testing. This will be evaluated using your report. You may only use the provided libraries, do not remove them from the code, and do not include more libraries. We recommend using PyCharm, and instructions for setting up and getting started are provided on the video.

## **GitHub Code Link**

Classes to implement

I. Backend
   a. Utilities.py
   b. SelectiveImageAcquisition.py
   c. ImageSynthesisNoise.py
II. Drivers
   a. ImageAcquisitionDriver.py
   b. ImageNoiseDriver.py

REQUIRED Libraries

a. OpenCV: Documentation
b. Numpy: Documentation

VERY IMPORTANT CONCEPT

We will be working with masks and have values between 0 to 1. We multiply the masks with the image's FFT to apply the mask to the image. Be aware, if you try to visualize a mask (save it as an image), it will be all black because you only have values from 0 to 1. To visualize it, you have to map those values from 0 to 255. However, when applying the mask, use the raw binary data and not the normalized values. If you look closely at image (c), there are small white dots. You can see them because the mask was normalized from 0 to 255, but when performing the operation, the mask's values are 0s and 1s.

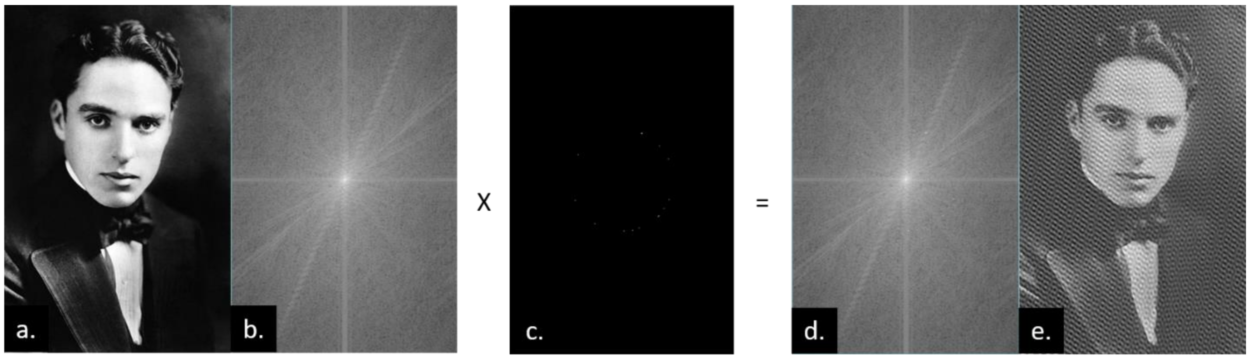
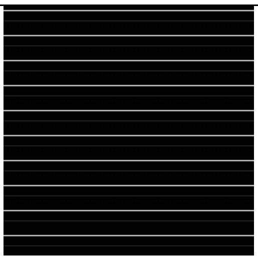
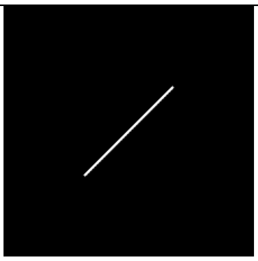
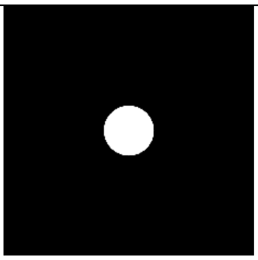
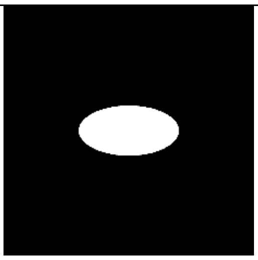
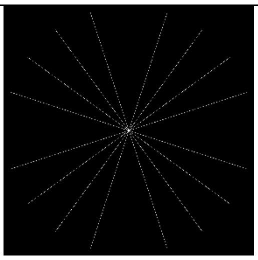
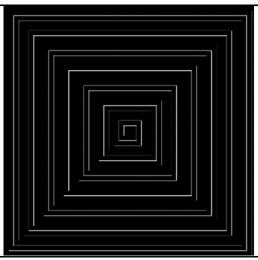


*Figure 1. (a) original image, (b) FFT representation of the image, (c) mask, (d) noisy FFT of the image, (e) noisy image*

## Utilities.py

These are general functions that you will use multiple times.  We have placed the name of each function for you to implement. You can add more functions, but do not remove functions from the document. Use OpenCV and NumPy functions in this file.

## SelectiveImageAcquisition.py

| Trajectory | Visualization | Description |
|---|---|---|
| Linear / Cartesian |  | def cartesianPattern(mask_size, percent):<br><br>The cartesian pattern will be a simple line by line scan. However, for this implementation the number of lines to be scanned will be a percentage of the image height. For instance, if the image is 10 pixels high and the percentage is 50%, the trajectory should be 5 equidistant lines starting from the top of the image mask. |
| Band |  | def bandPattern(mask_size, width, length, angle):<br><br>The band pattern will be a simple rectangle shaped mask. The parameters will be the length, width and the angle of rotation of the band. |
| Circle |  | def circlePattern(mask_size, radius):<br><br>This pattern will be a solid circle with it's center at the center of the image mask.The radius of the circle is the only parameter. |
| Ellipse |  | def ellipsePattern(mask_size, major_axis, minor_axis, angle):<br><br>This pattern will be a solid ellipse with it's center at the center of the image mask. The three parameters are the major and minor axis lengths, and the angle of rotation for the ellipse. |
| Radial |  | def radialPattern(mask_size, ray_count):<br><br>This pattern consists of different rays that go through the center of the image mask. The width of the rays will be one pixel and the length will be the same as the image width. he first ray will be a horizontal ray and each ray after that will be equidistant depending on the number of rays required. |
| [Bonus] Spiral |  | def spiralPattern(mask_size, sparsity):<br><br>This pattern will be a square spiral shape. The sparsity indicates the distance between the he spiral layers in pixels. |

## ImageSynthesisNoise.py

*Note: The formula for the Ring mask is not provided. Apply what you learned from the following to implement the ring mask.

**Ideal lowpass filter (ILPF):** The simplest low pass filter that cutoff all high-frequency components of the Fourier transform that is at a distance greater than distance $D_0$ from the center.

$$H(u,v) = \begin{cases} 1 & D(u,v) \le D_0 \\ 0 & D(u,v) > D_0 \end{cases}$$

- Where $D_0$ is a specified non-negative quantity (cutoff frequency), and D(u,v) is the distance from the point (u,v) to the center of the frequency rectangle.
- The center of the frequency rectangle is (M/2, N/2)
- The distance from any point (u,v) to the center D(u,v) of the Fourier transform is given by:

$$D(u,v) = \left[(u - M/2)^2 + (v - N/2)^2\right]^{1/2}$$

**M** and **N** are the sizes of the image.

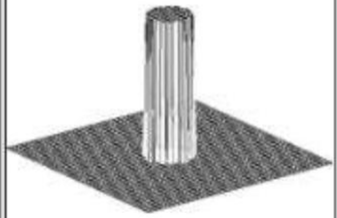
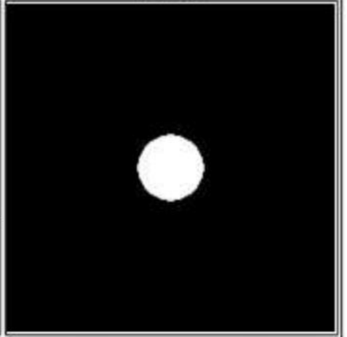
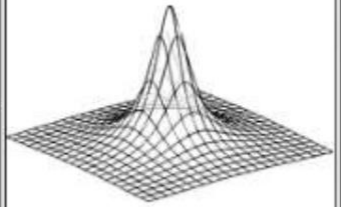
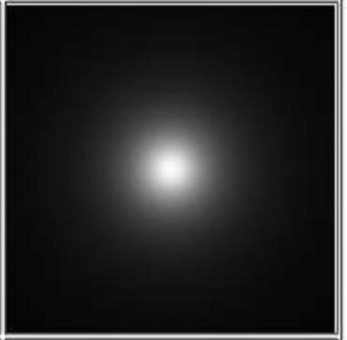
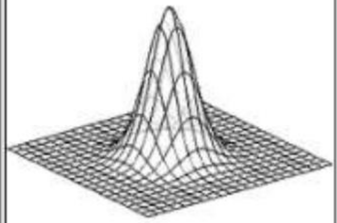
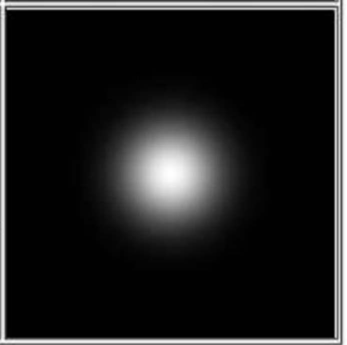
**Butterworth lowpass filter (BLPF):** of order n, and with cutoff frequency at a distance $D_0$ from the center.

$$H(u,v) = \frac{1}{1 + \left[D(u,v)/D_0\right]^{2n}}$$

**Gaussian lowpass filter (GLPF)**

$$H(u,v) = e^{-D^2(u,v)/2D_0^2}$$

The corresponding formulas and visual representations of these filters are shown in the following table. In the formulae, $D_0$ is a specified non-negative number (cutoff frequency). D(u,v) is the distance from the point (u,v) to the center of the filter. Butterworth low pass filter (BLPF) of order n.

| Lowpass Filter | Mesh | Image |
|---|---|---|
| Ideal: $H(u,v)=\begin{cases}1 & \text{if } D(u,v)\le D_0\\ 0 & \text{if } D(u,v)>D_0\end{cases}$ |  |  |
| Butterworth: $H(u,v)=\dfrac{1}{1+\left[D(u,v)/D_0\right]^{2n}}$ |  |  |
| Gaussian: $H(u,v)=e^{-D^2(u,v)/2D_0^2}$ |  |  |

The highpass filter is often represented by its relationship to the lowpass filter.

$$H_{HP}(u,v) = 1 - H_{LP}(u,v)$$

**Ideal highpass filter (IHPF):**

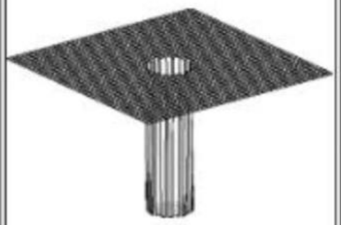
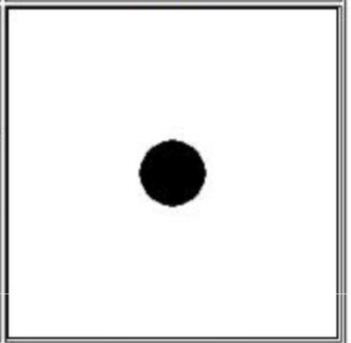
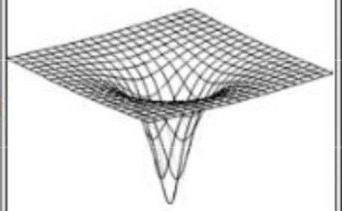
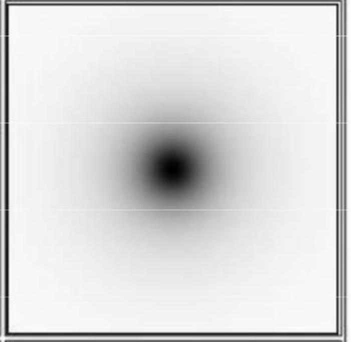
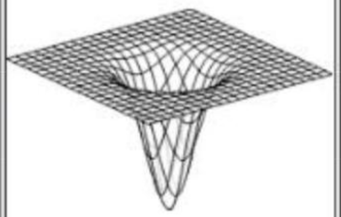
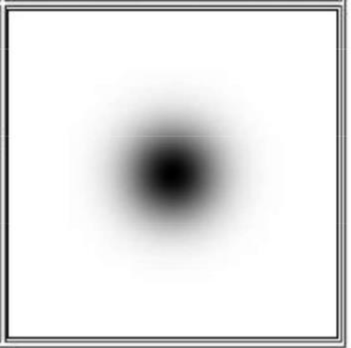
$$H(u,v) = \begin{cases} 0 & D(u,v) \le D_0 \\ 1 & D(u,v) > D_0 \end{cases}$$

**Butterworth highpass filter (BHPF):**

$$H(u,v) = \frac{1}{1 + [D_0/D(u,v)]^{2n}}$$

**Gaussian highpass filter (GHPF):**

$$H(u,v) = 1 - e^{-D^2(u,v)/2D_0^2}$$

| Highpass | Mesh | Image |
|---|---|---|
| Ideal |  |  |
| Butterworth |  |  |
| Gaussian |  |  |