**LASER MANIPULATION GUIDE**

This guide covers the GS Algorithm folder in the SLM CODES folder. Specifically the SLM\_FFT.py library. The “TEST CODE…” document is where I ran a lot of the codes, but, of course, they can run within the SLM\_FFT.py script as well.

#### **Fast Fourier Transforms and Gerchberg Saxton Algorithm**

Understanding the relationship between the fourier plane and the image plane and how to manipulate that relationship with the GS Algorithm is quintessential to running and producing holograms with the laser. We use a series of fourier transforms to take our desired image and create a mask for the SLM in order to see our image in the laser.

**Uses:**

*Numpy*

*OpenCV (cv2)*

*Matplotlib.pyplot*

*Pillow (PIL)*

*Glob*

*Fast Fourier Transforms (FFTs)*

Numpy has an implementation to do fourier transforms and in the lasermanipulation.py document, those definitions are expanded on for convenience. (Refer to python file for code)

*Fourier transform– ft():*

**Parameters:**

x = image to be transformed

**Description:**

The code above redefines a fourier transform. Here we use *fft.fft2* instead of *fft.fft* because we are using a fourier transform over 2 dimensions rather than a singular one, like an EKG or audio file. We then shift the fourier image back to where we want it.

*Inverse Fourier Transform– invft():*

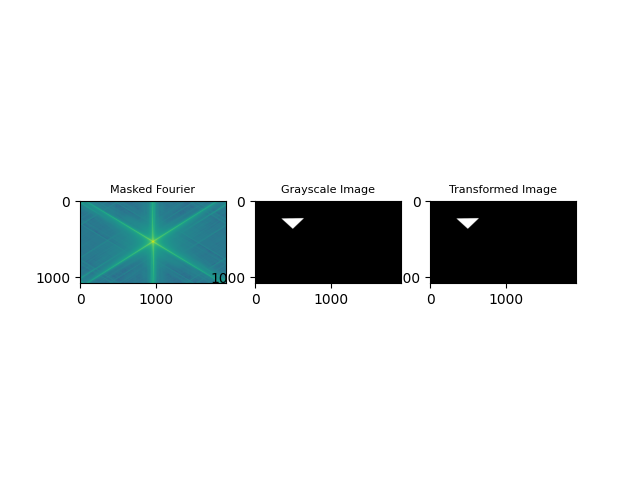
**Parameters:**

x = image to be transformed

**Description:**

This code does the exact same thing but in reverse. First shifting the fourier image back to where it was and then transforming back to the image plane. Again using *fft.ifft2* because we are working in 2 dimensions.

Below is an image, its fourier transform, and the inverse fourier transform (labeled as transformed image).

**

*Fig 1: Figure made from fouriertest() code in lasermanipulation.*

*Gerchberg-Saxton Algorithm (GSA):*

*GS():*

**Parameters:**

initial = image to be transformed (laser)

fourier = desired image

iterations = amount of runs through the algorithm

**Description:**

In order to use the GS algorithm we have to use the complex versions of images, so we redefine them by finding the peaks and multiplying them by e^i(phase). Numpy uses *np.angle()* to define the phase here. Then we take the fourier transform and do the same thing to our new image and redefine it with complex variables. Finally we take the inverse fourier transform and bring it back to the image plane.

We iterate this many times. 100 iterations seems to be a sweet spot and will get you good clarity of the images without taking up too much time. It takes about 1 minute for the algorithm to iterate through the 100, then it will display the phase mask you need for the SLM in order to replicate the image on the camera.

*GSplot():*

**Parameters:**

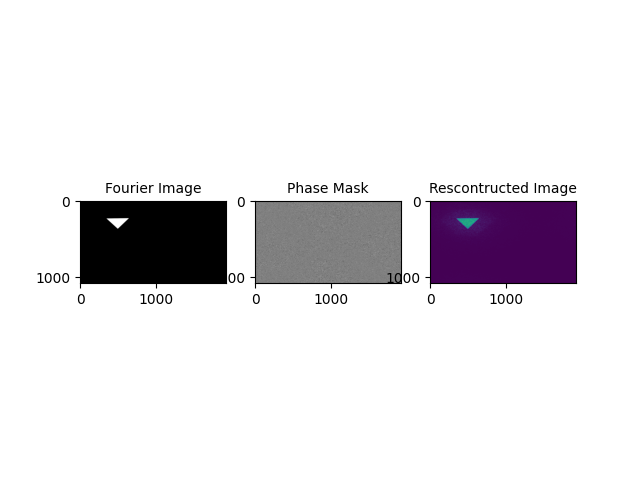
initial = image to be transformed (laser)

fourier = desired image

iterations = amount of runs through the algorithm

**Description:**

You may also use GSplot() to see something similar to what’s below.



*Notes:*

The GUI does this function but also takes a picture with the camera as it shows the phase mask allowing you to not have to save the phase masks in order to use them.

If you do need to save a phase mask, running the code as usual and using ctrl+S will give you the best quality. Matplotlib will also save at the same quality but it will have a large border. Worst case scenario: you can use OpenCV to save, however, you will have to convert the image to 8-bit and its quality will severely drop.

*fouriertest()* will take in an image, similar to *ft()*  or *invft(),* and will return a set of 3 images showing how the fourier transform worked on the image.

#### **Calibrating the SLM**

The SLM is already calibrated; we know it can create just slightly more than a 2pi phase difference. However, should you ever need to recalibrate it, there is a python program to do that.

The program contains several definitions that work in tandem to make the process smooth.

*PlotImage():*

**Parameters:**

y = name of the file

photodirectory = path to said file

**Description:**

This simply opens an image from your data set and allows you to select the ideal spot to isolate for data. Use the zoom feature of matplotlib to isolate a black space between the fringes (white bars) then input those coordinates into the next function. These coordinates will become parameters for all subsequent functions. Specifically, the upper left corner (xul, yul) and the lower right corner (xlr, ylr).

*showIsolation():*

**Parameters:**

y =name of image

xul = x-coordinate of upper left corner

yul = y-coordinate of upper left corner

xlr = x-coordinate of lower right corner

ylr = y-coordinate of lower right corner

photodirectory = path to image

waittime = how long the window stays open

**Description:**

This function will draw a red box around the isolation area that will be used for calibration. This is a good check to make sure that the coordinates found from the *plotImage()* function are viable. If not, try changing the coordinate values by 5 or so and you should get close. The isolated area does not need to be large either. Even a 3x3 selection will suffice, so long as it fits entirely between the fringes.

*Calibrate():*

**Parameters:**

samplesize = amount of photos you are using for calibration

xul = x-coordinate of upper left corner

yul = y-coordinate of upper left corner

xlr = x-coordinate of lower right corner

ylr = y-coordinate of lower right corner

photodirectory = path to image

increment = increment between photos

**Description:**

The next step is to actually calibrate the data. From the SLM we can input our sample size and increment. If you want to take 10 pictures across the entire scope of the SLM (255), then your increment will be 25. Note that 25\*10 is not 255. In this case, it is hard to get even increments so take your final picture at 255, but label it as 250. The naming scheme is very important for the function. It will only work if your images are labeled as the SLM level they were taken at (10.jpg, e.g.). The program will take these images and find the mean value of the isolated area, creating a graph of that mean v. the SLM level. It will also open each picture as it goes through and gives you a good representation of how your data looks.

You are looking for the fringes to travel up exactly once. That is, the graph should look like a singular hump (with some noise). Starting at a black spot should allow you to travel through an entire fringe and then back to black. Should you feel like your data is not great, try different spots on the images and use *AverageCalibrate()* to find the average value of the graph.

*AverageCalibrate():*

**Parameters:**

none

**Description:**

This will layer the images in the desired photo directory together. I would recommend using this with your graphs to find an average of how well the data is performing.

*Notes:*

*Michelson Interferometer:*

The first step to the calibration is creating a Michelson interferometer. It will definitely need a box around it and you MUST do your measurements without touching the table (even laptops). Talk to Prof. Gagnon about the process of making one

The SLM will need 3-4 hours to warm up and stop moving its fringes. It may seem unimportant but it will mess up your data, so you will have to wait to take the data until it's warmed up. If you need to do a calibration, I recommend planning ahead.

Take 5-10 pictures while the SLM level is zero to get a good idea of what your margin of error is. This is also a good way to make sure the SLM is not still warming up and moving the fringes by itself.

#### **Gaussian Beams**

Should you need to make a fake laser beam, you can use the set of gaussian functions to create whatever kind of gaussian beam needed.

*GaussianCut():*

**Parameters:**

x0 = initial x-value

xsize = width of curve

X = height of Gaussian

**Description:**

In order to plot a Gaussian distribution, you will need a sigma value. This function takes any gaussian identified from its initial *x* value, width (*xsize)*, and height (*X)*. It will then show the curve along with the sigma value.

*BuildGaussian():*

**Parameters:**

x0 = initial x-value

y0 = initial y-value

xsize = width of x dimension

ysize = width of y dimension

sigma = sigma value

**Description:**

This function takes inputs of the initial value and width in both the x and y directions along with a sigma value and builds a 3D model of a Gaussian which is examinable in 3D space.

*GaussianMask():*

**Parameters:**

x0 = initial x-value

y0 = initial y-value

xsize = width of x dimension

ysize = width of y dimension

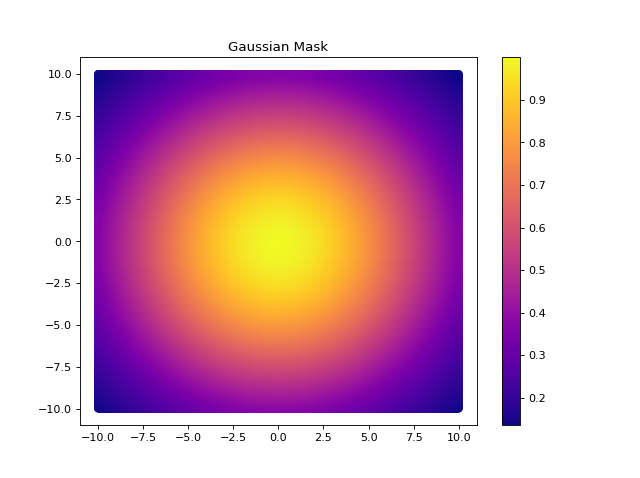
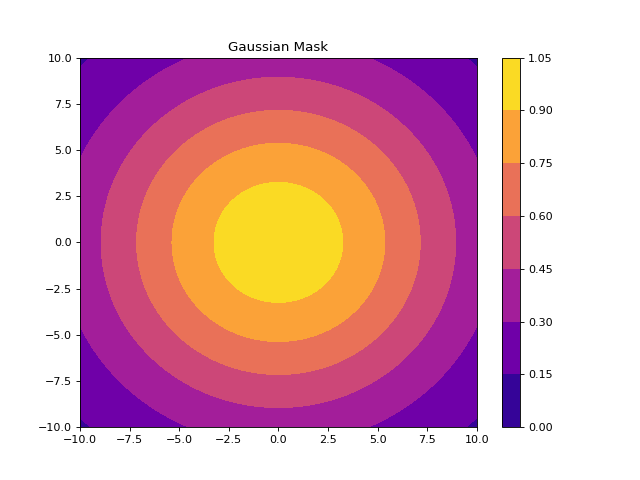
sigma = sigma value

maskType = type of mask (look in description for comprehensive explanation)

**Description:**

The Gaussian Mask takes in the same parameters as *BuildGaussian()* but it also takes into consideration mask type. This function takes the Gaussian and projects it onto the XY plane. The mask types allow you to specify how this projection will be used. There are two types of classifications: gradient v. contour, and for download v. not for download. A gradient map will take a much longer time to load but will produce the best looking map (best for making fake lasers), and a contour map which will produce a contour map of the laser (creating rings). A non-downloadable projection will be in color and have axes. Alternatively, the downloadable ones will be in b/w and will not have axes so that they are easy to use as ‘lasers.’

|  | Downloadable | Non-Downloadable |
| --- | --- | --- |
| Contour | ‘dlc’ | ‘contour’ |
| Gradient | ‘dlg’ | ‘gradient’ |

For reference, below are examples of the *‘contour’* and  *‘gradient’* mask types.

‘*Contour’* Mask ‘*Gradient’* Mask

*Notes:*

The function *FullGaussian()* will create a phase mask, gaussian cut, and 3D render should you feel the need. It also takes both contour and gradient mask parameters.

The program is not currently designed to download anything for you. So if you do use the dlc/g options, you will need to save it yourself

It will not come out in 1920x1080 so you will need to change the dimensions. I recommend programs like Inkscape or FireAlpaca both to quickly make manipulations to the image and also to create any images you would like for the SLM