1. Project Overview

In this project, you will process an image of a laser beam to measure the size and position of the beam.

In laser labs, we often need to know in detail the laser beam characteristics (think of the shape of the beam of a laser pointer on a screen): what is the beam size? Is it nice and round, or elongated? Has it moved? All of these properties can have strong effects on our experiments, and also tell us whether there are problems with the laser. In order to track all of these, it is common to directly image the laser beam – send a small portion of the beam to a camera, so that the image taken shows the laser beam shape. Where there is more light the image will be brighter etc. But since we care about changes of the beam that are hard to track by eye, we need to characterize the beam shape (called "beam profile") in a quantitative way; We do that by fitting a known function – a 2D gaussian – to the imaged beam shape, and tracking the relevant quantities resulting from the fit.

In this project, you will process real data taken from my and/or Hadas's lab and use code as a tool in the same manner that researchers do on a daily basis (I – Noam - made a script in python to do this very thing this summer).

Your code is required to take an image file and the optical system parameters as an input, and output to a .txt file a measurement of the beam size, as well as various plots related to the measurement.

2. Background

2.1 Laser beam shape

Real laser beams tend to have a gaussian shape in space, i.e. if you project the beam onto a screen or into a camera, the amount of light that reaches each point in space will form a two-dimensional gaussian.

$$I(x,y) = C * e^{-\frac{(x-x_0)^2}{2\sigma_x^2}} e^{-\frac{(y-y_0)^2}{2\sigma_y^2}} + B \qquad 1$$

Here x and y mark the position in space on a plane the laser hits, like a screen or a camera sensor. x_0 and y_0 are the positions of the center of the beam, σ_x and σ_y are the widths of the gaussians in their respective axes. C is a constant that is proportional to the total amount of light. B is a background level, and in general is a positive shift above the zero line.

The gaussian shape is maintained over each axis independently:

$$I(x) = \int dy\, I(x,y) = C' * e^{-\frac{(x-x_0)^2}{2\sigma_x^2}} + B', \qquad 2$$

and the same for $I(y)$.

2.2 The data

In this project you will work with an image of the laser beam - it will contain a measurement of the amount of light that reaches each point on the camera sensor.

$$counts[n,m] = I\left(x = \frac{n}{Pixels\ in\ x} L_x, y = \frac{m}{Pixels\ in\ y} L_y\right) \qquad 3$$

Where n, m are the pixel index, $Pixels\ in\ x/y$ are the number of pixels in each axis, and L_x and L_y are the dimensions of the sensor[1]. Each pixel will only contain a single number, the count, proportional to the amount of light, without any colour information. Below you can see an example of such an image of a laser beam cross section (fig. 1).

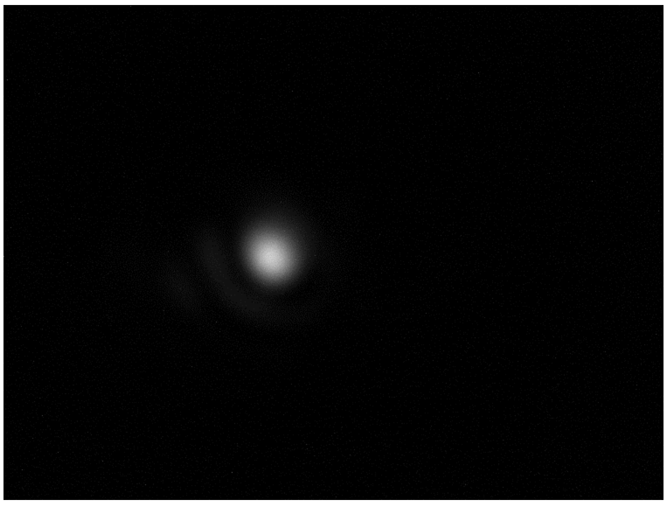

Figure 1 – An example of an image of a laser beam. The image has no colour information, only the amount of light on each pixel. The brighter the pixel (higher counts), the more light reached that spot on the camera sensor.

As you can maybe already see in this image, real data is dirty. The beam is not a perfect 2D gaussian. This becomes clearer when looking at a 1-dimensional plot of counts vs. x position in figure 2. Here, in order to get a 1-d graph from the 2-d image, we summed over the y axis:

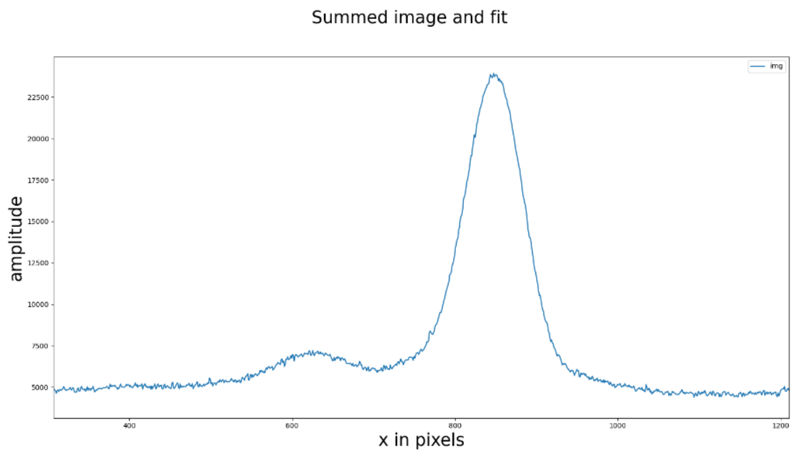

Figure 2 - A graph showing the counts in the pixels as a function of the x position. This graph is plotted after taking the sum over the y axis.

[1] More precisely: $count\ in\ pixel\ in\ position\ [n,m] = \int_{\frac{n-\frac{1}{2}}{L_x}}^{\frac{n+\frac{1}{2}}{L_x}} dx \int_{\frac{m-\frac{1}{2}}{L_y}}^{\frac{m+\frac{1}{2}}{L_y}} dy\, I_{(x,y)}$ $,but\ we\ can\ simply\ ignore$ this complexity.

$$counts_{x[n]} = \sum_{m=0}^{Ny_{pixels}} counts[n, m]$$ 4

As you can see, the graph has all kinds of features and noise over the pure gaussian signal. So, in order to find the beam position (center of the gaussian) for example, we cannot simply find the pixel with the maximum counts. So instead, in order to take a measurement of the beam properties, we can use statistical fitting.[2]

2.3 Fitting a function to data

There are various mathematical approaches to fit a function to data. The basic concept is to start with a function that should match the data (based on some prior knowledge, such as a theoretical explanation) and adjust some free parameters of the function until the "distance" between the data and the calculation according to the function is minimized. In this project you will use a fitting procedure that is already implemented in python (you do not need to write it yourself).

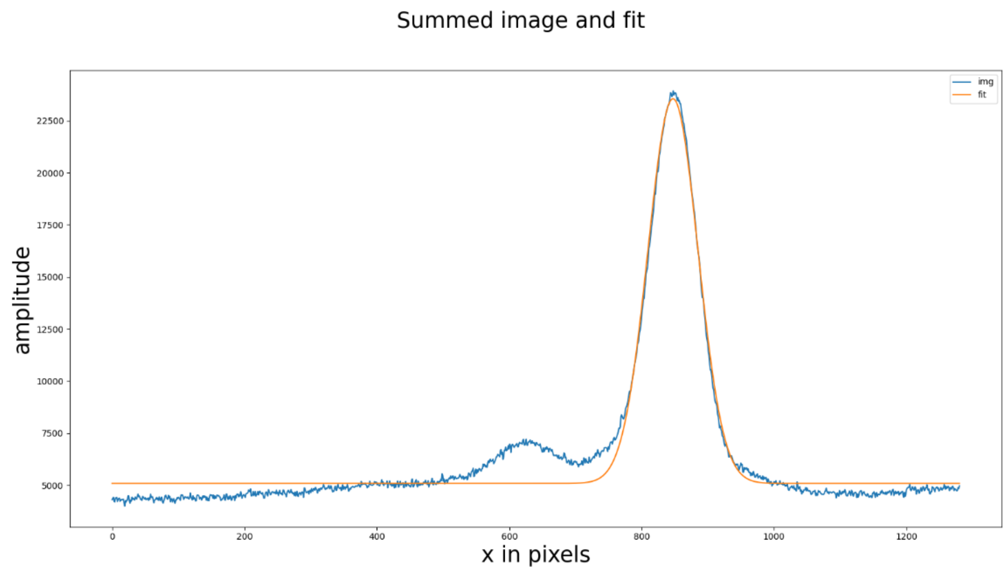

Figure 3 - The fit (orange line) plotted over the data from figure 2 (blue).

In our case the function is a 1-d gaussian (Eq. 2), the parameters are the amplitude C', the position x_0, and the width σ_x. In principle, one could fit directly the 2D gaussian (eq. 1). But fitting over a one-dimensional gaussian is easier and more reliable. In addition, summing over the other axis reduces the noise in the data.

2.4 Pointing stability

The position of the beam moves over time due to a variety of reasons such as effects of temperature change, vibrations, air currents, and construction work on a floor above the lab while we are trying to take very precise measurements.

The name for this property is Pointing Stability. To quantify it we take a series of images over time, measure the position of the beam in each image using a fit and then calculate the root mean square (RMS) of the results we have obtained. The RMS of the position is defined below:

[2] For those taking Lab A – this is the exact process implemented in Eddington. Note that the uncertainty of the measurement in each pixel is the same, thus for our purposes you can ignore the errors for the fit.

$$x_{RMS} = \sqrt{\frac{1}{N}\sum_{i=0}^{N}(x_i - \bar{x})^2} \qquad 5$$

Here N is the number of measurements, x_i is the position of the i'th measurement, and $\bar{x}$ is the average of the x position in all measurements. Similarly, you can calculate the RMS to the movement in the y axis by replacing x for y.

This can be calculated also for the radial distance i.e. the distance of each measurement from the average position defined by

$$r_i = \sqrt{(x_i - \bar{x})^2 + (y_i - \bar{y})^2}. \qquad 6$$

Giving us the RMS in r :

$$r_{RMS} = \sqrt{\frac{1}{N}\sum_{i=0}^{N} r_i^2}$$

<u>3. The project - Program Requirements and Layout</u>

3.1 input

Your code will accept an input file named input.txt, that will sit in the same directory as the code and the directory containing the images.

The input file will include the name of the directory containing the image files (you can assume that the directory contains only the images), and the size of the camera sensor in mm in the x axis in the following format (example input file in Moodle):

```
Name of the directory containing the image file

the size of the camera sensor in mm in the x axis.
```

Given this input file, your code will need to load each image into a numpy array using the tifffile library and calculate the pixel size in μm (assuming that the pixel is square, i.e. the pixel size in x and y is the same. The number of pixels in all images in the directory is the same and shows the whole detector. The x axis is the longer axis).

3.2 fitting

For each image, you will need to fit the data to 2 1-d gaussians (for each axis), to find the beam properties (width and position)

Using `scipy.optimize.curve_fit()`, your code should fit a gaussian function to the data.

You need to create a fitting function, a python function that takes an axis and parameters and returns the value of the function in each point on the axis.

The fit should be with respect to the image after summing the values of the pixels over one of the axes (you can use `numpy.sum()`). Then repeat the entire process for the other axis.

See more details in “implementing the fit” (section 4.1)

From the resulting fit, your code needs to calculate the size and position of the beam **<u>in $\boldsymbol{\mu m}$.</u>** The beam size is defined as twice the standard deviation: $W_0 = 2 \cdot \sigma_0$.

Note that the σ you will get from the fit will be in units of <u>pixels</u>. To convert the size of the beam from pixel to physical dimensions you will calculate the size of the pixel according to the sensor size given in the input. The physical size of the beam will be:

$$Size\ of\ the\ beam\ in\ \mu m = pixel\ size\ in\ \mu m * W_{0_{the\ size\ of\ the\ beam\ in\ pixels}}, \tag{7}$$

and similarly for translating the position from pixels to μm.

3.3 plotting

<u>For the first image only</u>, your code needs to plot the fit over the data in both axes, as shown in figure 3. A separate graph is needed for each axis. (see more details in section 3.3). Your code needs to display both plots to the user.

This is done as a “sanity check” for the fit – if the fit is working correctly for the first image, it will usually work for the rest of the images as well, since they are typically quite similar to each other.

3.4. Pointing stability

After the fit and plot is working correctly for the first image, you should repeat the fitting process for the rest of the images. Your code should save (ideally into a numpy array) the beam positions from the fits of each image (x_0 and y_0).

If you want to loop over files in a directory, getting as the loop index the names of the file one at a time, you can use the line:

```
for file_name in os.listdir(input_directory):
```

here, the loop index `file_name` contains the names of the files in `input_directory`, one at a time.

After obtaining all beam positions, your code should calculate the pointing stabilities (RMS) from the data you extracted based on Eq. 5 and 6.

3.5. Output

Finaly, you need to create an output file named Beam_Size_measerment_results.txt

The format of the output file is (also see example of output file in the Moodle):

```
Beam size x in micrometers =

Beam size y in micrometers =

Pointing stability in axis x in micrometers =

Pointing stability in axis y in micrometers =

Pointing stability in axis r in micrometers =

Beams average x position in pixels =

Beams average y position in pixels =

Pixel size in micrometers =

Name of the directory of the images =
```

Numerical values should be formatted to 3 significant figures (e.g 123, 1.23, 1.23e-7).

4. Further information and clarifications

4.1 Implementing the fit

To do the fit you will use the `scipy.optimize.curve_fit()` function as you saw in the tirgul.

You need to create a fitting function of the gaussian for the `curve_fit()`, give it initial guesses, and give boundaries for the parameters of the fit. You should consider physical boundaries and limitations when choosing these parameters, given that you don't know in advance the size and position of the input image.

Make sure to set reasonable initial guesses, and boundaries for the parameters of `curve_fit()`.

For example, what is the widest gaussian you can measure? Is a parameter strictly positive? And so on.

The amplitude of the gaussian will be close to the maximum value. From Eq. 2 $C' = Max\left[I_{(x)}\right]$.

The position, x_0 or y_0, will be close to the center of the image.

The width σ, will be smaller than the size of the image. See figure 1.

The background B, will be around the minimum or the average of the data.

As for the boundaries, the position of the gaussian will be within the image. The amplitude is positive and has a theoretical maximum of $2^8 * the\ size\ of\ the\ axis\ you\ summed\ over$. The 2^8 is the largest number that can be stored in the pixels of images you will get. The width will be larger than 1 pixel, so the camera will see it properly, and smaller the size of the sensor. The background is positive, and smaller the amplitude.

4.2 Checking the results: plotting

To make sure the results of the fit are valid and correct, you will plot the fit against the data.

To do this you will use the `matplotlib.pyplot.plot()` function.

Example code in the tirgul.

Your code should plot in the same figure the data and the resulting fit.

You will need to repeat this for each axis.

Make sure to give proper axes labels, plot labels, and titles to your figures.

4.3 libraries

To use the libraries, you'll need to install them by running the following line in CMD:

$$pip\ install\ numpy\ scipy\ matplotlib\ tifffile\ imagecodecs$$

For mac users, replace the pip with $pip3.12$.

$$pip3.12\ install\ numpy\ scipy\ matplotlib\ tifffile\ imagecodecs$$

numpy will allow you to manipulate numerical arrays easily and efficiently. Summation, finding the maximum, creating ranges for the plots and so on.

Scipy will allow you to do the statistical fitting.

Matplotlib will allow you to plot the results.

tifffile and imagecodecs will allow you to handle the images of .tiff format[3].

It is highly recommended to search out the documentation for the libraries and functions you will be using.

[3] There is no need to import in your code imagecodecs, only to install the library on your computer. It is necessary for tifffile to work correctly. In the parlance of coding, this is called a dependency.

There is no need to use loops in the code, numpy will have all the functions you will need to manipulate the data efficiently easily.

4.4 Submission

The submission in the Moodle should be a single .zip file, that includes all the .py code files that are required for you code to run. The name of .py file that need to run should start with:

“Final_Project_Main”.

Include in the .zip file the outputs you get from the example inputs in the Moodle, as well as the graphs of the fits, saved as a .PNG.

4.5 Extra Credit – 5 Points

As extra Credit to the Project, create a plot of the position of the beam as a function of time, and a scatter plot of all the positions.

For the time plot, the images are taken at 10Hz. The figure should contain both the x and y as two separate plots on the same figure.

For the scatter plot, the figure should contain the first 25 positions, as (x_0, y_0) points.

The code should show the user the graphs, and the .zip file of the submission should contain the .PNGs of the graph from the example file, the same as the fit graphs.

The extra credit is worth 5 points to the final grade to those how succeed.