Proposing an alternative method for the trap constant calculation for optical tweezers

TU Delft, Faculty of Applied Sciences,		Delft, 7th July 2020
BSc program Applied Physics	4914058	van Loon, Reinaart
Supervisor: Dr.ir. M.W. Docter	4645197	Sangers, Jeroen

1 Abstract

This report contains our findings on methods of finding the trap stiffness of an optical trap or optical tweezer setup and the conversion of an MATLAB data processing script to Python.

Optical trapping is a technique in which a small particle is being held in place by a focussed laser beam, commonly used in the field of biophysics.

The trap stiffness in relation to laser output power was determined by imaging a particle in an optical trap and repeating this for multiple laser powers. The resulting image stacks were processed by an automated MATLAB script that tracks the movement of the bead. The resulting trap constants were then plotted for each laser power output setting. The predicted direct proportionality between the laser power and the trap constant could be observed. However, given the distance from the data to the best fit, the values for the slope coefficient is not of much use.

Due to the current Corona-virus outbreak we were unable to do the optical trap measurements ourselves so to make up for this we were tasked with rewriting part of a MATLAB script in Python. The bead tacking algorithm could not be finished due to complications for non-grid interpolation in Python. Th sub-pixel interpolation function and the symmetry centre finding function, were successfully transformed to python script. The presented Python file is capable of roughly following the symmetry centre of the bead in an optical trap, however with low accuracy. There appeared to be a small offset between the true centre and the program estimate of the centre. The estimate also seemed to always lag behind a few frames. Therefore, an alternative Trackpy function is proposed as an alternative for the original tracking code.

Using the covariance matrix of the data, the values for the semi-major and semi-minor axis of the covariance ellipse were found for both data sets. The values match the theoretical inverse proportionality with respect to the laser power. The difference between the values for the major and minor axis compared to the MATLAB script shows the advantage of calculation of the trap constant using the covariance matrix. For this method incorporates the shape and direction of the optical trap.

Contents

1 Abstract **i**

Table of contents **ii**

2 Introduction **1**

3 Theory **2**

3.1 Optical trapping 2

3.2 Trap constant derivation 3

3.3 Error calculation 4

4 Experimental Method **5**

4.1 Experimental set-up 5

4.2 Computation 5

5 Results **6**

5.1 Results of the datasets 6

5.2 Python program results 7

6 Discussion **8**

7 Conclusion **9**

List of references **10**

8 Appendix **11**

8.1 Particle Spread 11

8.2 Python 12

2 Introduction

An optical trap, or optical tweezers, is a technique that is frequently used in molecular biology to study particles at the micro- and nanometre scale. By trapping a particle in a focussed laser beam, the particle is limited in movement. This allows the user to study microscopic manipulations and measurements on microscopic particles and therefore proves to be very useful in the field of biophysics. Examples of which are sorting of cells, unzipping of DNA and enzyme interactions [1] [2].

In order to perform quantitative measurements using optical tweezers, it is vital to know what force it exerts on the particle. For small movements of the particle around its equilibrium, this force is directly proportional with it's deviation. This proportionality constant is defined as the trap constant which is dependent on various parameters such as the particle size and laser power.

The aim of this report is to find the relation between the laser power and the trap constant. Secondly this report serves to get familiarised with the optical tweezers technique and to investigate its limitations and possibilities.

Due to the COVID-19 virus no experiments were carried out for this report and data of previous experiments by other students is used. This data contains images of a trapped bead for laser beams with different powers. For each laser intensity, there is a set of images at fixed time intervals such that the movement of the bead can be studied. For this report, the images are processed by a MATLAB algorithm which calculates the trap constant in two perpendicular directions. The relation between each trap constant and the laser power is found using least square fitting for the theoretical linear dependence.

Another part of this practicum involves designing a PYTHON algorithm that can perform the same calculation as the provided MATLAB code. In this report, the result of our programming is described and suggestions are made for improvement of the PYTHON code and further analysis of the trap constant is different directions.

In section 2 the theory regarding the report will be described followed by the experimental method in section 3. The results and discussion can be found in section 4. Lastly the conclusions in section 5.

3 Theory

The basic principles of optical tweezers and the theory for further calculations are described in this section.

3.1 Optical trapping

To understand the working principle of optical tweezers we consider a spherical dielectric particle, a bead, in a coherent light beam with a symmetrical intensity gradient such as in figure 1. The light beam will exert a force on the bead in the direction of the highest light intensity of the gradient. To understand this, we need to consider two situations.

For the situation in which the dimensions of the bead are much greater than the wavelength of the light we can apply straight forward ray optics. In the situation where the size of the bead is much smaller than the wavelength we can approximate the bead as a dipole that feels Lorentz force due to a gradient in the electric field.

For the situation where the dimensions of the bead are much larger than the wavelength of the light beam, we consider that photons can exert a radiation force on the bead. This force is a result of the momentum that photons carry and will be directly proportional to the light intensity. We now consider two rays of light that reach the bead symmetrical with respect to its centre. Due to the bead's spherical symmetric shape, the light rays will be refracted by the dielectric particle at the same rate, but in opposite directions (see figure 1). Both light rays will, given the change in direction of the light and the third law of Newton, exert a force on the bead. The light ray with the higher intensity will, however, exert a larger force. If the intensity gradient is larger in the centre of the light beam, such as in figure 1, this would lead to a net force pointing in the direction of the symmetry axis of the light beam. This force would trap the bead to the optical axis. In the case of a beam of light being focussed such as in figure 2, the bead would not only be trapped in the direction perpendicular to the beam axis, but also in the direction of the axis. This is also a result of the change of the refraction of light exerting a force on the bead (see figure 2). However, to light scattering, the bead is in the axial direction trapped slight behind the waist of the light beam. [1]

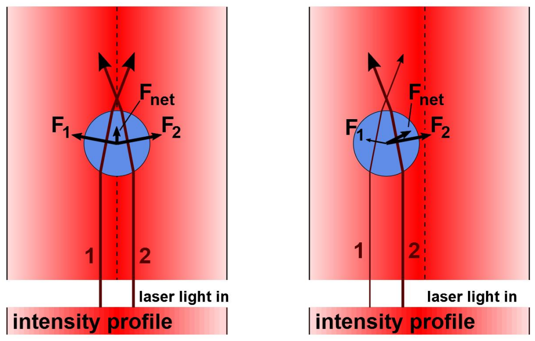

Figure 1: Schematic diagram of the ray optics explanation for optical trapping (unfocused laser). The intensity profile of the light beam is symmetric around the centre. When the bead is displaced from the beam center (right image), the net force is towards the centre due to a larger momentum change of more intense light in closer to the centre. The net force is zero in the horizontal direction when the bead is placed in the centre of the beam.

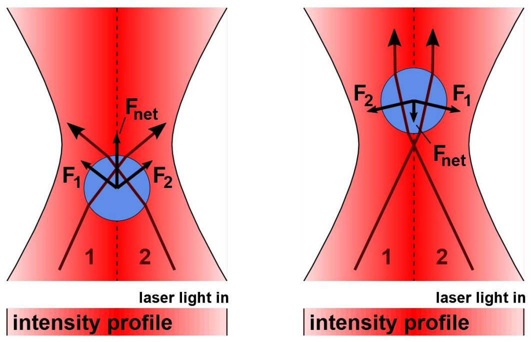

Figure 2: Schematic diagram of the ray optics explanation for optical trapping (focused laser). The light beam is now focussed. Due to the angle of incoming light, the momentum change of the light causes a net force points towards the focus. The equilibrium position is slightly behind the focus to compensate for the light scattering force.

Figure 3: Both figures were taken from Wikipedia [3].

In the situation where the dimensions of the bead are much smaller of than the wavelength of the light beam, we approximate the bead as a perfect dipole. According to Shaevitz (2006) , if we also consider the laser to have a Gaussian intensity profile in the plane perpendicular to propagation, the Lorentz Force is given by:

$$F = (p \cdot \nabla)E + \frac{1}{c}\frac{d\,p}{dt} \times B \tag{1}$$

Where $p = \alpha E$ is the dipole field, α is the polarizability, E the electric field induced by the light and B the magnetic field induced by the light. Optical traps are typically used with a continuous wave (CW) laser such that $\frac{\partial}{\partial t}(E \times B) = 0$. In this case the magnitude of the time-averaged force in the direction of the optical axis becomes [1]:

$$\langle F \rangle = \frac{\alpha}{2}\nabla\langle E^2 \rangle \tag{2}$$

Most optical trapping and also the experiment in this report includes beads with the same order of magnitude dimensions as the wavelength of the light beam. The physics of such a system is complicated and somewhat in between the cases explained above. This comprehensive theory will not be discussed in this report. However, from the latter derivations we can conclude that the trapping force is directly proportional with the laser intensity.

3.2 Trap constant derivation

According to Shaevitz (2006), 'for small motions of a bead near the centre of an optical trap, the forces acting on the bead approximate a zero rest–length, linear spring at the trapping centre.' Therefore, for small motions of the bead, the stored energy in the optical 'spring' is $1/2k_{trap}\langle x^2 \rangle$ with k_{trap} a constant defining the strength of the optical trap and $\langle x^2 \rangle$ the variance in the motion. According to the equipartition theory, the energy of the Brownian motion of a particle is given by $\frac{1}{2}k_b T$ with k_b the Boltzmann constant and T the temperature. [1] Equating the two energies yields:

$$k_{trap} = \frac{k_B T}{\langle x^2 \rangle} \tag{3}$$

From this we can conclude that by following the position of the bead over time, it is possible to find the the value of k_{trap}.

In the latter definition of k_{trap}, the 3 dimensions of real life are not taken into account. For the experiments in this report we only consider 2-dimensional images in the plane perpendicular to the propagation direction of the laser beam. This plane will in this report be addressed as the plane of interest, POI. For the POI we can consider two definitions for k_{trap}. We define $k_{trap,r}$ as the 'average' trap constant and is calculated using only the motion of the bead in the radial direction. $k_{trap,r}$ gives a good indication of the force in any arbitrary direction in the POI. It does, however, ignore the shape of the probability distribution of the bead. In the case where the potential well would be elongated such as in figure 4, the values for $\langle x^2 \rangle$ and therefore also k_{trap} can differ depending on the direction. We define the trap constant in an arbitrary direction as $k_{trap,i}$. Note that this is defined as a line in the POI which cuts the expectation value of the bead. To find the value for $k_{trap,i}$ we first realize that for realistic laser beams that are used for optical trapping, we expect a 2-dimensional gaussian intensity profile in the POI [1]. The shape of this gaussian profile can be described by a 2 dimensional covariance matrix. The iso-contours and therefore also the variance for such 2-dimensional gaussians are ellipses with their centre at the expectation value [4] (see figure 4. According to Rojas (2009), the

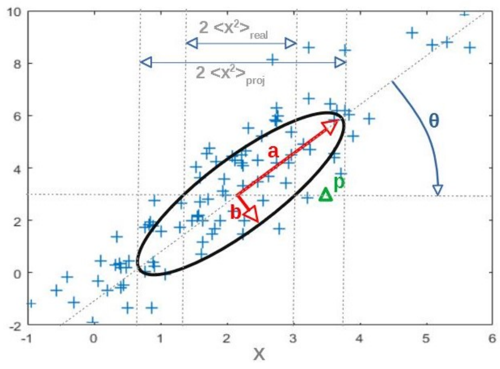

Figure 4: Schematic diagram of a data set with a bivariate gaussian distribution. The ellipse represents the variance of the distribution with it's semi-major and semi-minor axis a and b. θ represents the angle between a and line from the centre of the ellipse to an arbitrary point p. $\langle x^2 \rangle_{real}$ and $\langle x^2 \rangle_{proj}$ represent respectively the real variance in the x-direction and the projection of the variance ellipse on the x-axis.

eigenvectors of the covariance matrix point in the direction of the axis of an ellipse describing the variance in the POI. The largest of the two eigenvectors, $\vec{v}_1$, will point in the direction of the major axis and the smallest eigenvector, $\vec{v}_2$, in the direction of the minor axis. The magnitude of the semi-major axis and semi-minor axis, a and b respectively, are given by the eigenvalues corresponding to the eigenvectors. [5]

In order to find the variance in any arbitrary direction in the POI, $\langle x_i^2 \rangle$, we use the formula for an ellipse in polar coordinates:

$$\langle x_i^2 \rangle(\theta) = \frac{a\,b}{\sqrt{(b\,cos\theta)^2 + (a\,sin\theta)^2}} \tag{4}$$

Where θ corresponds to the angle of the direction of interest with respect to the major axis and the ellipse centre as origin. (see figure 4) Note that the latter definition of the variance is different than just projecting each point on an axis and subsequently calculating the variance. The difference between the two methods is clearly visible in figure 4 where $\langle x^2 \rangle_{real}$ is much smaller than $\langle x^2 \rangle_{proj}$. We can conclude from this, that when the position distribution is not symmetrical in the x- and y-direction, calculation of $k_{trap,x}$ or $k_{trap,y}$ without taking into account the covariance could give inaccurate results. Using equation 3 with the value for $\langle x_i^2 \rangle$ should yield better values for $k_{trap,i}$.

3.3 Error calculation

For this report, since we are not fully acquainted with the set-up and the corresponding error, when no error is specified the error is estimated to be half of the finest scale. For example for a size of 1.34 meter, the error would be 0.005 meter.

If Y is a variable which is a function of A,B,C, ... Then the error of Y, u_Y, is given by equation 5.

$$u_Y = \sqrt{\left(u_A \frac{\partial Y}{\partial A}\right)^2 + \left(u_B \frac{\partial Y}{\partial B}\right)^2 + \left(u_C \frac{\partial Y}{\partial C}\right)^2 + ...} \tag{5}$$

Using the latter equation we find that the error in the average position, $u_{\bar{x}}$ is given by:

$$u_{\bar{x}} = \frac{\sqrt{\sum_{i=1}^{n} u_{x_i}^2}}{n} \tag{6}$$

Since $\langle x^2 \rangle$ is given by the average of the squared of the distance to the average position we find that the error in the variance, $u_{\langle x^2 \rangle}$ is given by:

$$u_{\langle x^2 \rangle} = \sqrt{\frac{4\,\sum_{i=1}^{n} \left(u_{\bar{x}}^2 + u_{x_i}^2\right)(\bar{x} - x_i)^2}{n^2}} \tag{7}$$

For large values of n we can neglect the $u_{\bar{x}}^2$ term. Therefore we find:

$$u_{\langle x^2 \rangle} \approx \sqrt{\frac{4\,\sum_{i=1}^{n} u_{x_i}^2\,(\bar{x} - x_i)^2}{n^2}} \tag{8}$$

If u_{x_i} is constant we can rewrite this as:

$$u_{\langle x^2 \rangle} \approx \sqrt{4\,u_{x_i}^2 \langle x^2 \rangle} \tag{9}$$

Using equation 3 in equation 5 we find that the error in the trap constant, $u_{k_{trap}}$, is given by:

$$u_{k_{trap}} = \frac{u_{\langle x^2 \rangle} k_B T}{\langle x^2 \rangle^2} \tag{10}$$

In the case of a constant value of u_{x_i} we can use equation 9 to find that:

$$u_{k_{trap}} = 2\,u_{x_i} k_B T \langle x^2 \rangle^{-\frac{3}{2}} \tag{11}$$

Using equation 3 we can rewrite this as:

$$u_{k_{trap}} = 2\,u_{x_i} (k_B T)^{-\frac{1}{2}} k_{trap}^{\frac{3}{2}} \tag{12}$$

4 Experimental Method

No experiments were carried out for this report, but measurements from previous students are used. The experimental set-up that they used and the computer algorithms that were used for this report will be discussed in this section.

4.1 Experimental set-up

The experimental set-up that was used for this report can be seen in figure 5. According to the practicum manual, the red light from the laser has a wavelength of $\lambda = 658$ nm and passes through a beam expander in order to completely fill the back (back focal plane) of the objective. The beads that are used in the experiment have a diameter of approximately $2\ \mu m$. The mirrors M1 and M2 are used for compacting the beam path and aligning the beam to the optical axis of the objective. The used objective is meant to be used in an infinity corrected microscope.

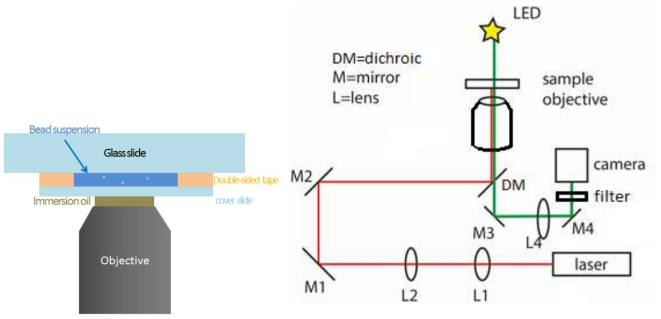

Figure 5: Schematic diagram of the experimental set-up that was used for the experiments corresponding with the treated data in this report. This diagram was taken from the practicum manual [6].

L4 focusses the light back to an image at focal distance. In the set-up, the L1 and L2 have focal length of respectively 50 and 350 mm. The front focal distance of the objective lens is 1.8 mm and the second objective lens has a focal length of 200 mm. Using simple division and error calculation with equation 5 we find that the magnification is $M = \frac{200}{1.8} = 1.1\ \pm 0.3 \cdot 10^2$. Given the pixel size of the camera of $5.2\ \mu$ and equation 5 we find that the conversion factor for pixels to length is given by $l_{pixel} = \frac{5.2}{M} \cdot 10^{-6} \approx 4.7\ \pm 0.1\ \cdot 10^{-8} m/pixel$.

Using the discussed set-up, a bead was to be trapped by a laser beam and 1000 images were taken at fixed time intervals. This was performed for laser beams with different powers. The involved powers were 0, 5, 10, 20, 30 and 40 mW. For this report, two data sets from different students are analysed.

4.2 Computation

A MATLAB algorithm was provided for this practicum in order to calculate the trap constant in a x- and y-direction, $k_{trap,x}$ and $k_{trap,y}$. This algorithm involves noise removal of the images and tracking the bead using a 'quadrant-interpolation (QI) algorithm which makes use of the circular geometry of the diffraction pattern to resample the image on a circular grid.' [7] Subsequently, power spectrum analysis is performed to calculate the values for $k_{trap,x}$ and $k_{trap,y}$. Since the trapping force is directly proportional to the light intensity the correlation between the laser power P and the trap constant k_{trap} is given by: $k = \alpha \cdot P$. Using the MATLAB code and a linear regression method, the value for α is determined for the two data sets.

Secondly, a PYTHON algorithm is used with the same noise cancelling technique, but with a trackpy function to follow the bead. This algorithm subsequently calculates the covariance matrix and the vectors describing the covariance ellipse. Using this PYTHON code, the change of this covariance ellipse is analysed.

5 Results

5.1 Results of the datasets

The results of the MATLAB code calculating $k_{trap,x}$ and $k_{trap,y}$ for the two data sets are presented in table 1. Since the MATLAB code failed to track the bead for some of the measurements in the first data set, not all values are used for fitting. The second dataset was much better and did not have any faulty measurements. The corresponding values are shown in table 2.

Laser Power $[mW]$	0*	10*	20	30*	40	40
$k_{trap,x}$ $[pN/nm]$	$2.36 \cdot 10^{-5}$	$2.45 \cdot 10^{-7}$	$1.60 \cdot 10^{-5}$	$8.35 \cdot 10^{-7}$	$1.99 \cdot 10^{-5}$	$9.60 \cdot 10^{-5}$
$k_{trap,y}$ $[pN/nm]$	$8.29 \cdot 10^{-5}$	$2.94 \cdot 10^{-7}$	$1.65 \cdot 10^{-5}$	$9.71 \cdot 10^{-7}$	$1.72 \cdot 10^{-5}$	$8.96 \cdot 10^{-5}$

Table 1: Results of the first dataset. The values are truncated to two decimal places. * denotes a faulty measurement

Laser Power $[mW]$	0	5	10	20	30	40
k_x $[pN/nm]$	$3.64 \cdot 10^{-7}$	$8.24 \cdot 10^{-5}$	$1.08 \cdot 10^{-4}$	$3.03 \cdot 10^{-4}$	$6.14 \cdot 10^{-4}$	$7.55 \cdot 10^{-4}$
k_y $[pN/nm]$	$5.55 \cdot 10^{-7}$	$4.18 \cdot 10^{-5}$	$2.53 \cdot 10^{-5}$	$1.10 \cdot 10^{-4}$	$1.69 \cdot 10^{-4}$	$2.54 \cdot 10^{-4}$

Table 2: Trap results, values are truncated to two decimal places for formatting reasons

We cannot calculate the exact value of the error since the MATLAB algorithm does not provide us with the locations and its error. However, if we estimate the error of the bead location to be $u_{xi} = l_{pixel}/2$ and we know that 1000 frames were used for the calculation of the trap constants we find from equation 11 that $u_{k_{trap}} = l_{pixel}(k_B T)^{\frac{1}{2}} k_{trap}^{\frac{3}{2}}$. However when these values are calculated we see an error between 0.1% and 0.00001% of that of the k_i value, these values are lower then we believe they should be. We suspect that this is the case since the MATLAB algorithm confidently tracks a incorrect stationary artefact on the trap surface.

[h!]

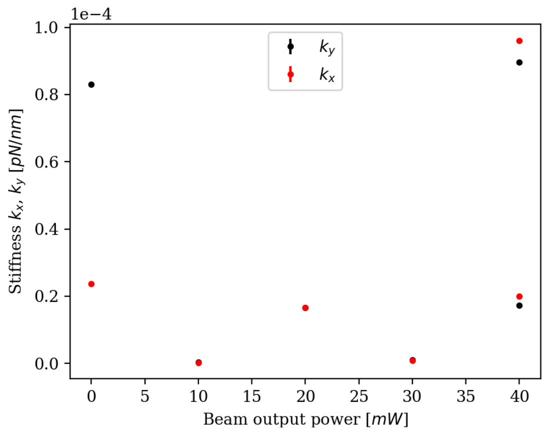

Figure 6: Results of the first dataset plotted and fitted.

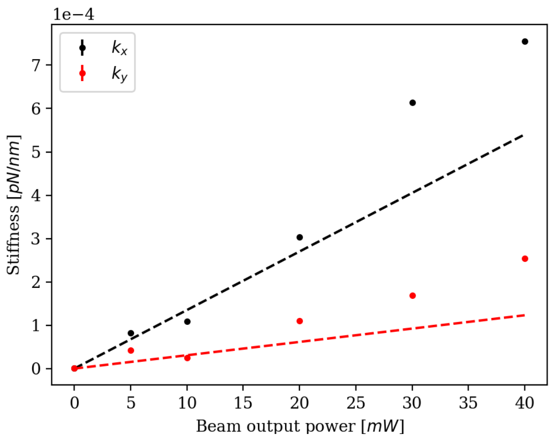

Figure 7: Results of the second dataset plotted and fitted.

The data shown above in tables 1 and 2 are plotted in figures 6 and 7 respectively. In the plots the individual data points are plotted. For the first dataset no reasonable fit could be made corresponding with the expected linear relation. For the second dataset, a least squares fit for k_x and k_y to the linear relation to the laser power is plotted.

The corresponding values for the correlation constants are $\alpha_x = 1.5 \pm 0.5 \cdot 10^{-5}$ and $\alpha_y = 4 \pm 1 \cdot 10^{-6}$

Table 3: Resulting Axis lengths of dataset 1

Dataset 1						
Laser power output $[mW]$	0	10	20	30	40	40
Semi-major-axes $[nm]$	$8.69 \cdot 10^{-4}$	$3.88 \cdot 10^{-4}$	$2.41 \cdot 10^{-5}$	$2.50 \cdot 10^{-5}$	$3.36 \cdot 10^{-6}$	$4.73 \cdot 10^{-6}$
Semi-minor-axis $[nm]$	$5.66 \cdot 10^{-5}$	$9.94 \cdot 10^{-5}$	$1.19 \cdot 10^{-5}$	$7.16 \cdot 10^{-6}$	$2.93 \cdot 10^{-6}$	$3.97 \cdot 10^{-6}$

Table 4: Resulting Axis lengths of dataset 1

Dataset 2						
Laser power output $[mW]$	0	5	10	20	30	40
Semi-major-axes $[nm]$	$9.67 \cdot 10^{-4}$	$1.22 \cdot 10^{-5}$	$3.02 \cdot 10^{-6}$	$1.86 \cdot 10^{-6}$	$1.32 \cdot 10^{-6}$	$4.73 \cdot 10^{-6}$
Semi-minor-axis $[nm]$	$1.24 \cdot 10^{-5}$	$1.90 \cdot 10^{-6}$	$4.48 \cdot 10^{-7}$	$2.71 \cdot 10^{-7}$	$1.45 \cdot 10^{-7}$	$2.89 \cdot 10^{-6}$

5.2 Python program results

During this practical we were also tasked with rewriting a MATLAB script in to Python. The piece of script we needed to rewrite was the function that tracks the centre of the bead in the image, this function was comprised of interpolation method and a method that found the symmetry centre of the bead using Fourier transforms. The second part was easily implemented in Python as it was mostly just finding the right Python functions that were equivalent to their MATLAB counterparts. The interpolation function was a lot harder to implement in Python since most of the existing Python interpolation functions did not have the same functionality. The documentation of this part of the practical can be found in the appendix

As outlined in the section 3.2 a method for deriving the trap constants was proposed that differs from the one from the MATLAB script. This method was implemented in a second different python script to try and show its usefulness. This python script can be found in the appendix.

The result speaks for itself, it shows a clear connection between the laser output power and the length of the semi-major and semi-minor axis of the ellipses that encircle the points describing the symmetry centres of the beads. The points were fitted with a function with shape $L_{axis} = \frac{\beta}{P} + \gamma$, this seems to fit well as can be seen in figure 8 & 9. This suggests that the theoretical inverse proportionality between the $\langle x^2 \rangle$ and the laser power is correct. The data corresponding to these plots are shown in table 3 and in table 4. In the appendix three more plots showing the spread of the centres of mass can be seen.

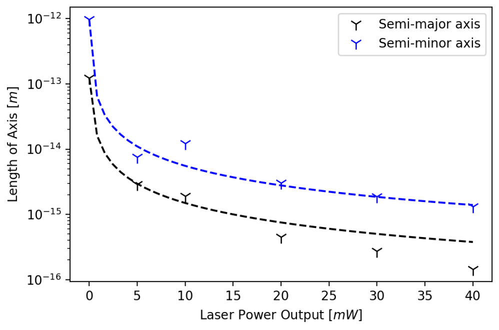

Figure 8: Semi-major and minor axis of the first dataset plotted and fitted.

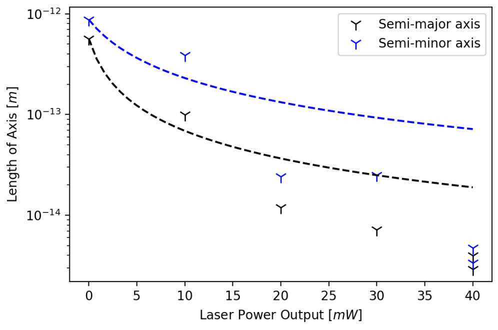

Figure 9: Semi-major and minor axis of the second dataset plotted and fitted.

The values for semi-major- and semi-minor-axes were acquired using the Trackpy python function. Given the values and the accuracy of the fitted lines indicates that the Trackpy function performs well for particle tracking. The difference between figure 6, not showing any correlation, and figure 8, showing the theoretical correlation, can be explained by better performance of the Trackpy function compared to the MATLAB script for the first dataset.

6 Discussion

As expected the values for $k_{trap,x}$ and $k_{trap,y}$ increase with an increasing laser power. For the second data set, a reasonable fit can be made using the theoretical relation between P and k_{trap}. The fit does however not fit the errors. Therefore one could doubt the accuracy of the result for the corresponding α. For the first data set we find that the data does not match the theoretical direct proportionality between P and k_{trap}. This seems to be the result of a failing MATLAB code which does not follow the bead. Studying the images shows other particles or some sort of unwanted artefacts in the image which were not the trapped bead(see figure 10). The code sometimes uses the position of the noise as the bead position, therefore giving inaccurate results.

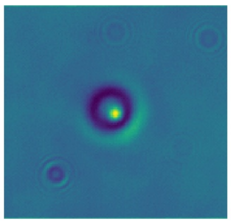
Figure 10: An image of the first dataset for the 10 mW laser showing other particles than the bead that was trapped by the optical tweezers.

The practical manual asks for a calculation of the average trap constant. It was proposed to calculate this as follows: $k_{trap,tot} = \sqrt{k_{trap,x}^2 + k_{Trap,y}^2}$. This would however give inaccurate results since the orientation and shape of the variance is not taken into account. For the calculation of the trap constant we take the inverse of the variance. As explained in the theory we expect the variance to be ellips shaped. However, if we take the inverse of the radius of an ellipse we get a complicated shape such as in figure 11. Comparing the shape to a circle with radius $\sqrt{k_{trap,x}^2 + k_{Trap,y}^2}$ shows that the proposed method for the calculation is not appropriate. The method described in section 3.2 could provide more accurate results.

The results for the semi-major and semi-minor axis of the covariance ellipse indicates an inverse correlation of the axis length to the laser power. This is expected given equation 3 and the direct proportionality between the trap constant and laser power. It is interesting that the values for a and b (see section 5.2) for the first data set seem to have more than a factor 5 difference which would indicate an elongated covariance ellipse. Comparing this to the MATLAB results, there seems to be virtually no difference between $k_{trap,x}$ and $k_{trap,y}$ (only taking into account correct measurements). This points out that the MATLAB algorithm, only projecting the positions on two axis, fails to incorporate the shape of the variance (see figure 4). Therefore, the method as outlined in section 3.2 seems to be promising for calculation of trap constants in any direction independent of the orientation and shape of the trap.

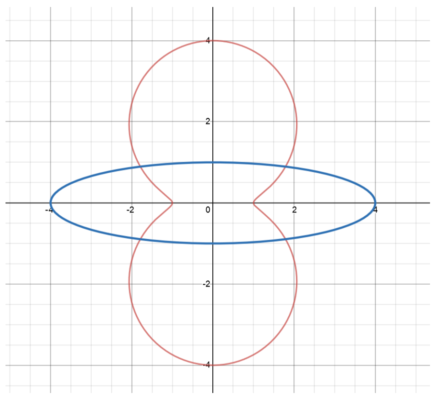

Figure 11: An ellipse (blue) and the inverse of its radius (red).

The task of recreating a MATLAB script in Python was partially successful. The symmetry centre finding function was successfully implemented as well as the subpixel interpolation function. Due to some dissimilarities in the way MATLAB and Python functions interpolate an unstructured set of data we were unable to get the main tracking function to work. The resulting estimates of the symmetry centre location were not far of but were not dead-on either.

Although the MATLAB code works fine in many cases, it showed that it does not always work well. This can visually be seen when running the first dataset but is also illustrated by the difference between figure 8 compared to figure 6. Moreover, for students, the MATLAB code is long and involves some complicated steps. For these reasons we would suggest using the Trackpy for particle tracking. This function shows promising results in speed and accuracy. By visual inspection when running the code, it is clear that this function had no trouble whatsoever with the data covered in this report. To investigate the accuracy of the Trackpy function, a numerical analysis should be be done comparing it's values to the values acquired by the MATLAB script. A downside of the Trackpy function is, although there is elaborated documentation of the function, the maths used for the tracking is not explicitly noted. Therefore students would perhaps have less insight in the tracking mechanism. More information about the Trackpy function can be found in the appendix.

7 Conclusion

The predicted direct proportionality between the laser power and the trap constant could be observed. However, given the distance from the data to the best fit, the values for the slope coefficient is not of much use.
Rewriting the MATLAB script using Python was a partial success, the symmetry centre finding function and the subpixel interpolation function were successfully implemented in Python but the tracking function currently uses a slow and not very useful interpolation method which results in an undesirable offset and delay in the predicted symmetry centre. With a properly functioning interpolation function the python script would probably yield s similar result as the MATLAB script.
Using the method described in the theory, the values for the semi-major and semi-minor axis of the covariance ellipse were found for both data sets. The values match the theoretical inverse proportionality with respect to the laser power. The difference between the values for the major and minor axis compared to the MATLAB script shows the advantage of calculation of the trap constant using the covariance matrix. For this method incorporates the shape and direction of the optical trap.

References

[1] J. W. Shaevitz, “A practical guide to optical trapping,” *University of Washington*, vol. 138, 2006.

[2] A. J. Te Velthuis, J. W. Kerssemakers, J. Lipfert, and N. H. Dekker, “Quantitative guidelines for force calibration through spectral analysis of magnetic tweezers data,” *Biophysical journal*, vol. 99, no. 4, pp. 1292–1302, 2010.

[3] “Optical tweezers,” Jun 2020.

[4] C. B. Do, “The multivariate gaussian distribution,” *Section Notes, Lecture on Machine Learning, CS*, vol. 229, 2008.

[5] R. Rojas, “The secret life of the covariance matrix,” *Freie Universität Berlin [online], URL: http://www. inf. fu-berlin. de/inst/ag-ki/rojas_home/documents/tutorials/secretcovariance. pdf [cited 1 October 2012]*, 2009.

[6] M. W. Docter, “Optical tweezers lab manual,” Jun 2020.

[7] M. T. van Loenhout, J. W. Kerssemakers, I. De Vlaminck, and C. Dekker, “Non-bias-limited tracking of spherical particles, enabling nanometer resolution at low magnification,” *Biophysical journal*, vol. 102, no. 10, pp. 2362–2371, 2012.

8 Appendix

Below two plots showing the by the trackpy library found centre of masses of the beads are displayed. These plots indicate the tightening of the axes for increased laser output.

8.1 Particle Spread

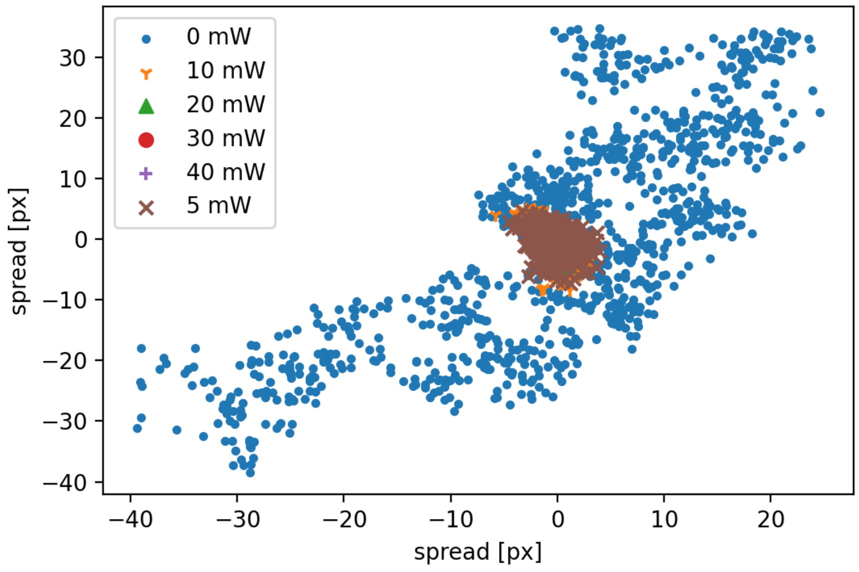

Figure 12: Spread of the first dataset shifted towards origin by average displacement from centre

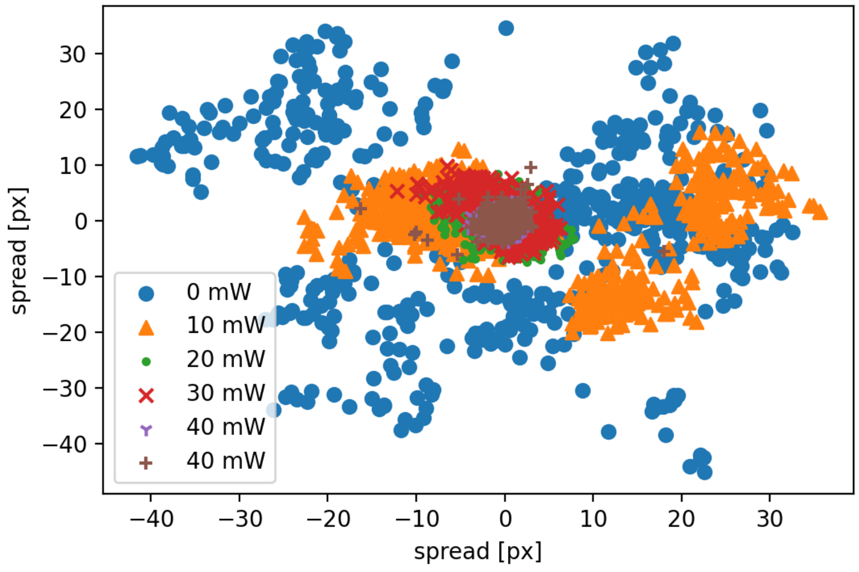

Figure 13: Spread of the second dataset shifted towards origin by average displacement from centre

8.2 Python

The python code used to calculate the ellipse axes is displayed below.

```
def ellipse_calc(x,y):
    # replace nans by average
    x_ = np.where(np.isnan(x), np.nanmean(x), x)
    y_ = np.where(np.isnan(y), np.nanmean(y), y)

    # Calculate variance and covariance
    var_x = np.sum((x_-np.mean(x_))**2)/(len(x)+1)
    var_y = np.sum((y_-np.mean(y_))**2)/(len(y)+1)
    cov = np.sum((x_-np.mean(x_))*(y_-np.mean(y_))/(len(x_)+1))

    cov_matrix = np.asarray([[var_x, cov],[cov,var_y]])
    evals,evecs = linalg.eig(cov_matrix)
    evecs_ = evals*evecs
    #plt.plot(x,y, linestyle='none', marker='x', zorder=1)
    #plt.quiver(np.nanmean(x),np.nanmean(y), -evecs_[1,:],- evecs_[0,:],zorder=2, units=


    a = np.max(evals)
    b = np.min(evals)
    print(a)
    print(b)
    index_a = np.where(evals == a)[0]
    theta = np.arctan(evecs[0,index_a]/evecs[1,index_a])
    print('theta␣=', theta)
    return a,b,theta
```

Documentation MATLAB to Python, RP Optical Tweezers Round 15

Stijn van der Lippe, Stein Glastra, Jeroen Sangers and Reinaart van Loon

We elaborate upon the issues we encountered in converting the MATLAB code of step 8 in GUI7_MDTIF to Python. Furthermore we elaborate upon some changes we made in the implementation of the code in Python and explain what some of the functions we added do.

Documentation step 8: QI tracker

The part of the code we were stuck on implementing is the interpolation of a non-rectangular grid. Several functions were tested in Python. Below we discuss why they do not give the results we want.

scipy.interpolate.griddata:
https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.griddata.html
This function has to perform a triangulation for every data point in the input array to find the nearest neighbour if the input array is irregular (as is the case for us). This happens every iteration. This is extremely slow and therefore this function is unusable. More details can be found here.

scipy.interpolate.RectBivariateSpline:
https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.RectBivariateSpline.html
This creates a class that takes in two 1D arrays of coordinates and a single 2D array of values at the coordinates.
Calling this class using *.ev()* allows the interpolation at data points using 1D arrays of x and y coordinates. Calling the class normally can't work since it needs an array of points of all (x,y) coordinates we want to evaluate at, in ascending order which is something we can't do since we can't sort our points in such a way that the (x,y) coordinates stay linked.

scipy.interpolate.interp2d:
https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.interp2d.html
This function won't work since it only works with 1D arrays for x and y and outputs a 2D array of interpolated values which would be easy to use if the grid was rectangular, but it isn't so it would take multiple nested for loops which will be slow and error prone. And even then we get an array of outputs that assumes a rectangular grid and would therefore be distorted.

scipy.ndimage.map_coordinates:
https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html
This function is promising but currently yields some strange results which we do not yet fully understand and can't really be tested until the previous code works. In our implementation, the function only returns integer values and we are unsure what the cause of this is.

Documentation subpix_step
In MATLAB an array [~,a] was created to get the value and index of the largest entry in the d-array, in Python this had to be substituted with *np.argmax()* to get only the index.

Documentation SymCenter
In MATLAB, NaNs in the input array were first located using a find function. Subsequently, they were padded using the value *nanmean* of the input array. In Python, this can be done in a single step using *np.where*. This was done by using the truth statement *np.isnan*. If the statement is true for an array element, the value of that element was replaced with *np.nanmean* of the input array.

Documentation get_eval_coords
A function that takes in a 2D array of coordinates and returns a flattened 1D array of these coordinates. It is used in the function *track_xy*.

Documentation rebuild_grid
A function that takes in a 1D array of points and a shape. The function then builds a 2D array with the same dimensions as shape. It is used in the function *track_xy*.

Documentation QI_tracker
A small error was found in the *__init__* function of *QI_Tracker*. In the variable assignment of *self.X0samplinggrid* and *self.Y0samplinggrid* both arrays were not transposed, while they should be. This was fixed by adding *.T* at the end of both.

removeNoise
We suspect the long duration of the calculation can be attributed to the time the noise removal takes. We tried to speed this process up by replacing an iteration function (*next*) with a numpy *np.where* function, which is not iterative.
The following line was replaced:
xIhalf = next(x for x, val in enumerate(ilstflat_t)
with
xIhalf = np.where(ilstflat_t>Ihalf)[0][0].
This had little influence on the speed of the process.

The following line was also replaced:
xmindist = next(x for x, val in enumerate(dist) if val == min(dist))
with
xmindist = np.argmin(dist)
This had a major impact on the speed, removeNoise now takes less time to run.

Trackpy
http://soft-matter.github.io/trackpy/v0.4.2/tutorial/walkthrough.html

Trackpy is a python package which is used for particle tracking. It proves to be easy in use and promises accurate results (article about accuracy: https://www.cell.com/biophysj/fulltext/S0006-3495(05)73136-2).
Using this Python function instead of the complicated QI-tracking would potentially not only make a more compact and structured code, it also showed some promising results in comparison with the MATLAB code. For the ‘Bead’ data set that we were given, the MATLAB code fails to track the bead. This is a result of large jumps in space of the bead in the image which cannot be followed by the QI-tracker. The trackpy function, however, seems to have no trouble following the bead whatever the size of the ‘jumps’.
Another advantage is that the trackpy function provides the error in position for each independent frame. Therefore, instead of guessing the error for the QI-tracker, exact error calculation can be performed.