a

Projektarbeit / Student Project

Three Dimensional Dual Colour Single Molecule Localisation Microscopy

concluded at the

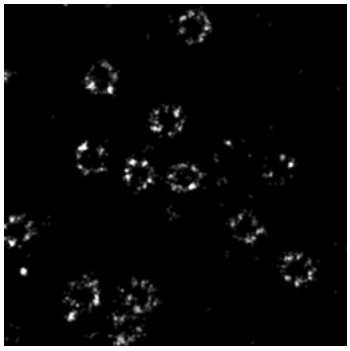
Technische Universität (Tu) Wien

advised by

Lukas Velas

of

Moritz Siegel



Vienna
June 29, 2021

moritz.siegel@tuwien.ac.at
https://github.com/imrahilias/biophysics

Contents

Chapter 1

Introduction

The ultimate goal of this thesis is to facilitate dual colour three dimensional single molecule localisation microscopy (SMLM), by alternating excitation with different lasers, but still measuring only one channel.

SMLM is a technique of circumvent Abbe's diffraction limit, enabling the study of structures smaller than half the wavelength of light; thus reaching super resolution fluorescence microscopy.

1.1 Fluorescence Microscopy

Fluorescence microscopy is a optical microscopy technique using fluorescence instead of reflection or transmission of light, foremost to study organic substances.

The specimen is illuminated with laser light, exciting the respective fluorophores; causing them to emit (much weaker) light on a slightly longer wavelength. After separation from excitation light using spectral filters, this image is captured by a sensitive scientific complementary metal-oxide-semiconductor (sCMOS) camera.

However, fluorophores that are bound to the specimen surface and those in the surrounding medium exist in an equilibrium state; When exciting with a conventional fluorescence microscope, the fluorophores bound to the surface are often overwhelmed by the background fluorescence.

1.2 Total Internal Reflection

To restrict the excitation to a thin layer of the specimen, and so only illuminating the bound fluorophores, total internal reflection fluorescence (TIRF) was invented. This is a type of microscope with which a thin region of a specimen, usually less than 200 nanometers can be observed. This is used notably to study constituents of and interaction with cellular surfaces, such as cell adhesion or binding of antibodies.

1.3 Single Molecule's Location

The basic principle of all SMLM techniques is to separate the signal of neighbouring individual emitters in time, which allows to determine their positions with nanometer precision.

This is done with stochastic optical reconstruction microscopy (STORM), which was invented simultaneously by different research groups, and has since the mid 2000s become the most widely used technique for SMLM. In order to prevent two fluorophores that are near by each other to overlap their signals, which prohibits distinction and so ultimately limits the obtainable resolution; STORM simply suppresses the emission of a photon 99% of the time.

During STORM a few of the fluorophores are randomly activated to a *on state*, emit light for some time τ, and then fall back into the *off (dark) state*, resulting in fluorophore *blinking*. This has to be repeated many times, until all the fluorophores have been *on* several times stochastically; in our case yielding 50k frames, each consisting of many, but mostly not neighbouring emitters.

1.4 Two Dimensional SMLM

For obtaining 2d localisations, it is convenient and most often precise enough to fit a Gaussian to each of the diffuse dots in the frame, in order to estimate their centre—thus, the position of the fluorophore. For this thesis we used the ImageJ plugin `ThunderSTORM` cit for this task. cit

In Figure 1.1, a summary of the principle of 2d SMLM is shown, including the common pitfalls: Due to localization errors, the localizations are slightly displaced from the true molecule positions (a). In addition, the structure is distorted or misrepresented by decreased labeling efficiency (b), label displacement (c) or overcounting (d), cit .

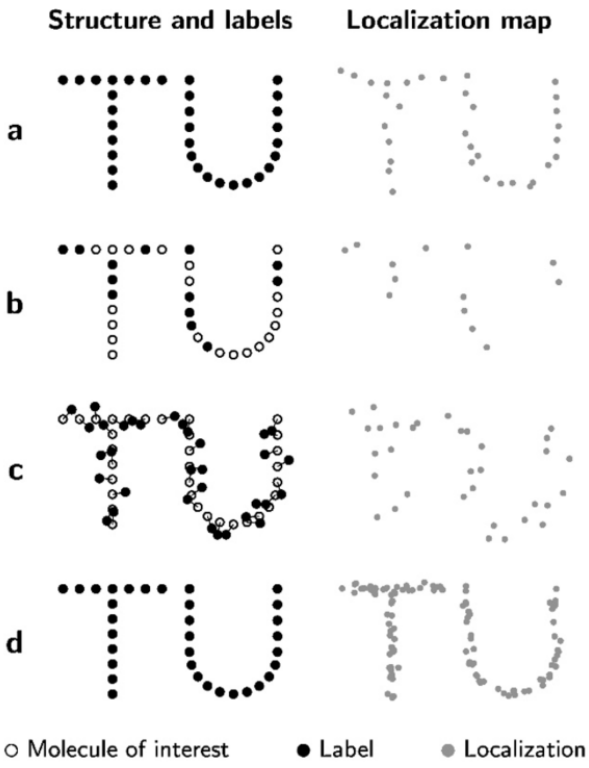


Figure 1.1: Influence of imaging parameters on the reconstructed SMLM image. The actual localization (left column) of the protein molecules (circles) yielding the obtained localization (right column).

1.5 Three Dimensional SMLM

To be able to accurately estimate the 3 dimensional location of the fluorescent molecule, the PSF must be known quite well. Simply put: If one knows the shape of a point source in varying degrees of defocus, one can guess the defocus and thus the z coordinate of the fluorescence molecule.

Fitting a full Point Spread Functions (PSF)—using prior calculated Zernike Modes—to such a stack of images containing reasonably spaced fluorescence signals even yields a list of 3d localisations.

Chapter 2

Methods

2.1 Maximum Likelihood

For the Analysis of direct stochastic optical reconstruction microscopy (dSTORM) fluorescence microscopy, maximum likelihood estimation (MLE) is widely accepted as a gold standard; We closely follow the method from cit here. First we assume a Poissonian probability amount of photons n_k detected in pixel k:

$$P_k(n_k|\mu_{\boldsymbol{\theta},k}) = \mathrm{e}^{-\mu_{\boldsymbol{\theta},k}} \frac{\mu_{\boldsymbol{\theta},k}^{n_k}}{n_k!} \tag{2.1}$$

where the expectation values $\mu_{\boldsymbol{\theta},k}$ is defined as:

$$\mu_{\boldsymbol{\theta},k} = \beta + s * h_k(x_0, y_0, z_0) \tag{2.2}$$

depending on the form of the point spread function (PSF) model h and on the parameter vector $\boldsymbol{\theta}$:

$$\boldsymbol{\theta} = (\beta, s, x_0, y_0, z_0) \tag{2.3}$$

with β and s denoting the mean photon numbers for background level respective signal; and x_0, y_0, z_0 denote the particle position in 3d.

For each recorded frame, the parameter vector $\boldsymbol{\theta}$ is estimated by finding a minimum of the total negative log-likelihood function using the MATLAB tool `fminunc`:

$$\boldsymbol{\theta} = \arg\min_{\boldsymbol{\theta}} \sum_k \mu_{\boldsymbol{\theta},k} - n_k \ln \mu_{\boldsymbol{\theta},k} \tag{2.4}$$

2.2 PSF Model & Zernike

In first order approximation the PSF can be aberration-free, but for SMLM this is not going to be precise enough. In this thesis we employ a PSF model expressed in a Zernike modes basis using Noll coefficients cit ranging from 2 to 37 (3rd order spherical aberration) plus the 4th order spherical mode Z_{56}:

$$\Phi = a_{56} Z_{56} + \sum_{i=1}^{37} a_i Z_i \tag{2.5}$$

2.3 STORM & dSTORM

We used dSTORM microscopy throughout this thesis, which is the *direct* variant of STORM, utilising very bright fluorophores, with a high rate of photoswitching and minimal photobleaching.

state pic

2.3.1 Gloxy Buffer

The principle of STORM is based on an appropriate *storm buffer*, which suppresses the activation of the *bright state*, and limits the amount of oxygen, the main source of photobleaching.

Single channel SMLM may employ the same buffer for both excitation wavelengths, in our case at 645 nm (red) respective at 488 nm (blue). After its central ingredients glucose

and oxidase, the buffer we used throughout this paper for all single channel measurements unless noted otherwise is called Gloxy buffer.

Gloxy buffer concentration

- 50 mmol β-mercaptoethylamine hydrochloride (MEA) (Sigma-Aldrich).
- 10 vol% of a $250\,\mathrm{g\,L^{-1}}$ solution of glucose.
- $0.5\,\mathrm{mg\,ml^{-1}}$ glucose oxidase.
- $40\,\mathrm{mg\,ml^{-1}}$ catalase (Sigma-Aldrich).
- with phosphate-buffered saline (PBS) adjust to pH 7.6 .

2.3.2 OxEA Buffer

Dual colour fluorescence microscopy poses novel challenges to find a proper dSTORM buffer, that works for both excitation wavelengths simultaneously—thus two distinct fluorophores AF647 and AF488—without interfering with each other. The buffer composition we used is based on cit, and called OxEA after its main ingredients OxyFlour and MEA:

OxEA buffer concentration

- 50 mmol MEA (Sigma-Aldrich).
- 3 vol% OxyFlourTM (Oxyrase Inc., Mansfield, Ohio, U.S.A.).
- 20 vol% of 60% sodium DL-lactate syrup (L1375, Sigma-Aldrich).
- with PBS adjust to pH 8–8.5 with NaOH.

OxEA buffer protocol

For about 1 mL of OxEA buffer we used the amounts shown in Table 2.1, to obtain above listed concentrations.

Table 2.1: Ingredients used for preparation of OxEA buffer for dual channel dSTORM buffer.

Order	Ingredient	Store	Vol / µL
1	Ultra pure H_2O		600
2	10 M NaOH		20
3	10× PBS		100
4	60% DL-lactate	fridge	200
5	1 M MEA	freezer	50
6	OxyFluor	freezer	30

pH

The pH of the OxyFlour β-mercaptoethylamine hydrochloride dSTORM buffer (OxEA) buffer is checked using both broad range pH testing strips, and a digital pH meter, to be between pH 7 and pH 8.

2.4 Dual Colour Optics

It is possible to use different excitations to simultaneously measure different fluorescence markers, but for that the point spread function has to known for each wavelength. So the PSFs are estimated both for red and blue laser light, with the Phase Retrieval program by [Jesacher et al 21?]; each on a stack of 50000 dSTORM images of 100 nm 100nm? beads stained with fluorescence dyes for both red (645 nm) and blue (488 nm) laser light.

2.4.1 Aberration Correction Collar

To further complicate things, the new Olympus 1.5 NA objective comes with a Correction Collar to compensate for aberrations—which naturally vary slightly for both color channels. In order to find the best Correction Collar setting of the Olympus 1.5 NA objective for SMLM, the Point Spread Functions (PSF)

is computed for each of the three settings of the correction collar $\{0.13, 0.17, 0.19\}$; using the program by Jesacher et al, described in Section 2.2.

2.4.2 Cramér Rao Lower Bound

In order to find the best correction collar setting of the Olympus 1.5 NA objective for SMLM, the Cramér Rao lower bound (CRLB) is computed using the MATLAB program `crlb.m`, with prior estimated PSF via phase retrieval Jesacher et al.

All CRLBs are computed at a defocus position of -500 nm, in steps of 5 nm from 0 to 500 nm. For all Estimations we assume identical arbitrary, but consistent signal strength (2500) respective background (100).

2.5 Dual Colour Projection

The ultimate goal of this thesis is to facilitate dual colour SMLM, by alternating excitation with different lasers, but still measuring only one channel. The alternation ensures the comparability of the two sets of images; so that they are affected by drift over time, bleaching, etc. in the same way.

The generated image stacks are then split and fed separately through the SMLM algorithm, since the PSF model depends on the emitted wavelength—and thus on the excitation. The two obtained data sets with localisations for both channels are then analysed together.

Since the methods used to obtain 3d SMLM data are wavelength dependent, the two channels will most likely be *off*, yet we ultimately want to overlay them. We hereby propose a calibration step, based on evaluating evenly dyed beads in both channels, analysing the localisations, and finally figuring out their offset.

2.5.1 Rigid Transform

In first order approximation, this offset may be thought as linear, so the two sets are related via a *rigid* transform; thus only by rotation and translation. Such a transformation can be solved easily via linear algebra; the solution used here is one of the many known ways, using least squares and singular value decomposition (SVD) based on cit.

In this short overview, the Lines refer to the the function `rig()` shown in Section A.3.3, where the full code is listed. Basically, one first calculates the centroids of the two sets of points `p`,`q`, and shifts both sets to their respective centres, thus eliminating translation (block 1 at Lines 24–28). As a second step the *best* (least squares) transform that *rotates* the set p to the set q is found via SVD (block 2 at Lines 30–51). As a last step the translation between set p and q is computed using the known rotation (block 3 at line 53).

2.5.2 Affine Transform

The next more complex approach includes other—nonlinear—transformations like *shearing* or *scaling*, called an *affine* transform. Our approach is based on cit, and can be viewed as a more complex variant of the rigid transform shown in Section 2.5.1, still using least squares and singular value decomposition (SVD) to find the optimal affine transform in 3-dimensional euclidean space.

Between centering and SVD, which follow the rigid transform, a *orthogonal reduction* is performed (block 2 at Lines 37–60). Here the covariance matrices of both sets `p` and `q` are computed, to form their inverse via a Choleski decomposition (Lines 44–49). The already centred sets `p` and `q` are subsequently orthogo-

nalised (Lines 51–56); so that `p` and `q` are now solely related by a rotational matrix.

2.5.3 Simulation

As a proof of concept,we performed both a rigid and an affine recovery on simulated data. The code for this simulations is listed in Section A.3.6, as well as in my Git repository at git. First, each simulation creates a cylinder-shaped pseudo-random point cloud `p`, and deducts some projection to get a second set `q`, related to `p` by either a rigid or an affine transformation.

2.5.4 Projected 2d NPC

As a second proof of work, we tested both rigid and affine recovery on several real dual colour SMLM data sets of nuclear pore complex (NPC)s of two-color in-focus measurements of different cells successively recorded in the same sample. The fluorescence images are analysed with *Thunderstorm*, a dSTORM plugin for ImageJ, to obtain 2d localisations based on Gaussian fits.

Since these data sets are two dimensional, we added low-magnitude noise ($\mathcal{O}(10^{-3})$) as a virtual z component, so both sets are projected almost onto the z plane for analysis. This is necessary, because the singular-value-decomposition can faults on 2d data.

The resulting data sets are then analysed either a rigid or an affine transformation.

This process is centrally governed by the octave program `cockpit.m`, which internally uses the octave function `ampel.m` to sort the imported data file into the two alternating color channels. The code is listed in Section A.3.1, and Section A.3.2, as well as in my Git repository at git.

Chapter 3

Results

3.1 Dual Colour Buffer

In order to enable similarly precise dual colour three dimensional SMLM, by using alternating excitation with red and blue laser light; the buffer constituents play a central role.

The prepared samples, both beads and NPCs have been evaluated using glucose oxidase dSTORM buffer (Gloxy) buffer, described in detail in Section 2.3.1, which works well with either red or blue excitation.

Additionally the samples, both beads and NPCs have been evaluated using OxEA buffer, described in detail in Section 2.3.2, which is needed if one wants to use both red and blue excitation.

The OxEA buffer did work, but offered notably less blinking, as well as was much lower signal-to-noise ratio (SNR) For the red channel compared to using Gloxy buffer; with the blue channel being worse still.

3.2 Dual Colour Optics

3.2.1 Zernike Modes

The Zernike modes of the PSF are shown for each of the three correction collar settings $\{0.13, 0.17, 0.19\}$, each for blue channel respective red channel in Figure 3.1 respective Figure 3.2.

For convenience the same results are additionally shown grouped by the the three correction collar settings $\{0.13, 0.17, 0.19\}$, now for both red and blue channel in Figure 3.3, Figure 3.4 and Figure 3.5.

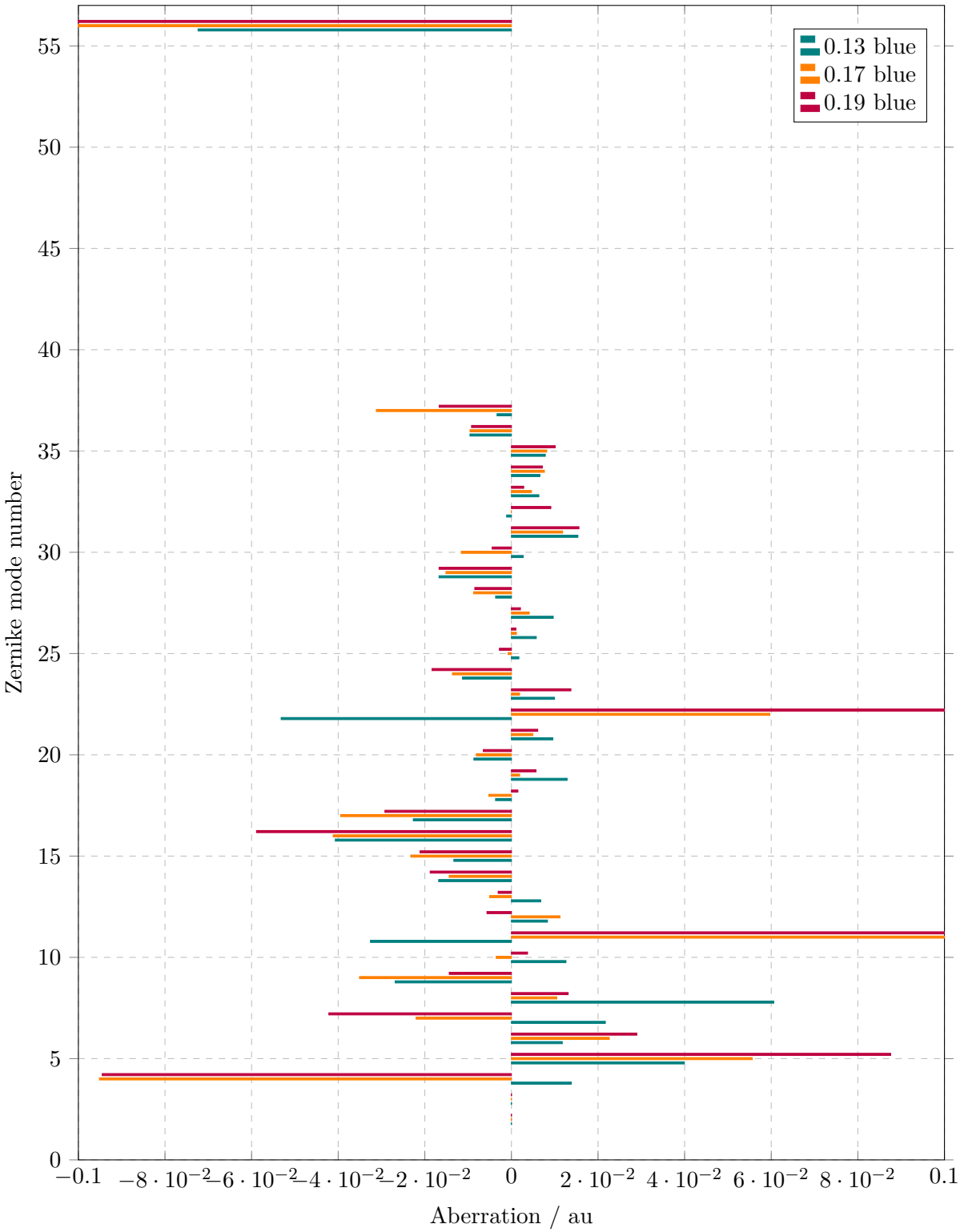


Figure 3.1: Blue channel ZERNIKE *modes* $\{1 \dots 37, 56\}$ *versus aberrations of PSF model via phase retrieval, for all three correction collar settings (0.13, 0.17, 0.19).*

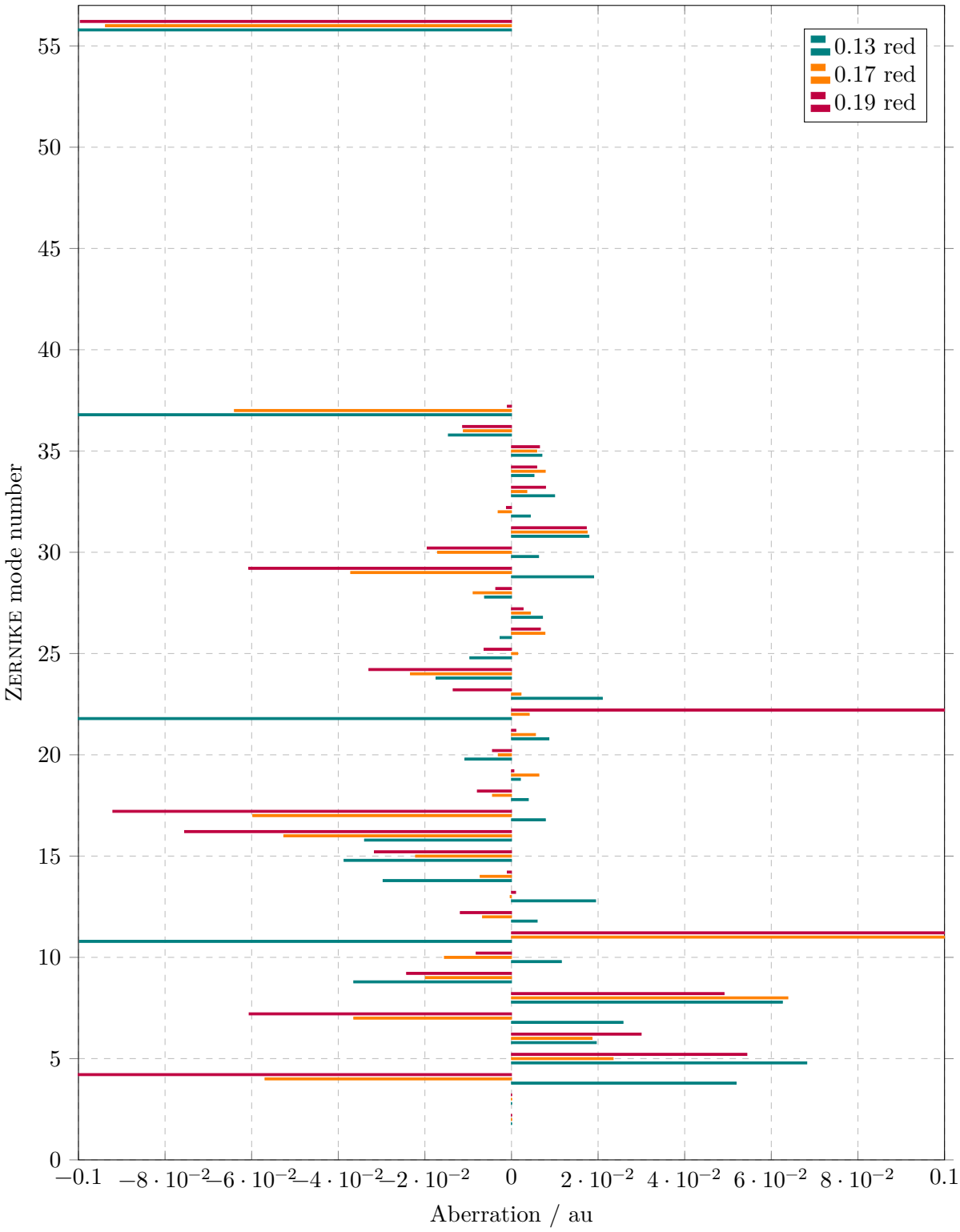


Figure 3.2: Red channel Zernike modes $\{1 \dots 37, 56\}$ versus aberrations of PSF model via phase retrieval, for all three correction collar settings (0.13, 0.17, 0.19).

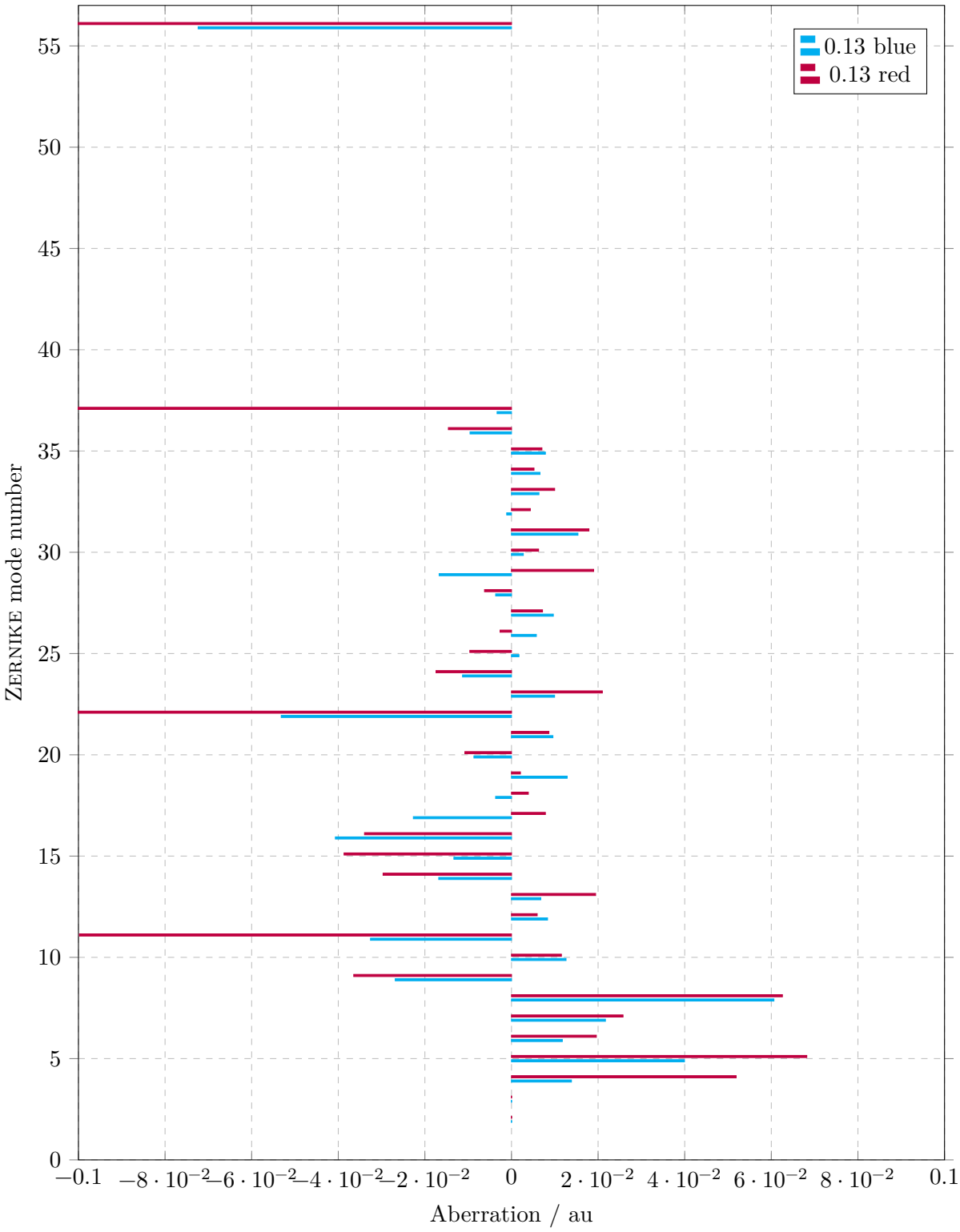


Figure 3.3: Red and blue channel Zernike modes $\{1 \dots 37, 56\}$ versus aberrations of PSF model via phase retrieval, for correction collar setting of 0.13.

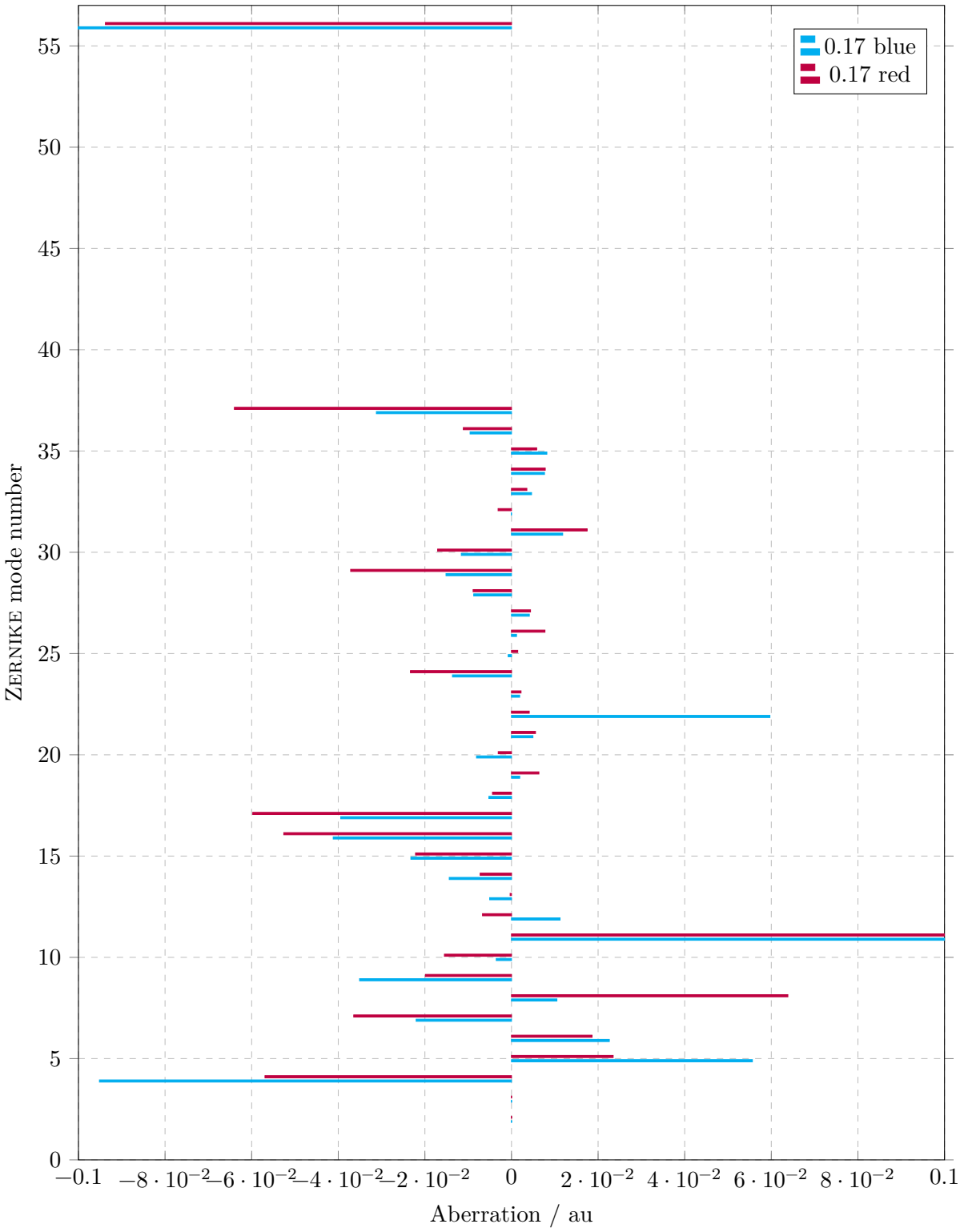


Figure 3.4: Red and blue channel Zernike modes $\{1 \dots 37, 56\}$ versus aberrations of PSF model via phase retrieval, for correction collar setting of 0.17.

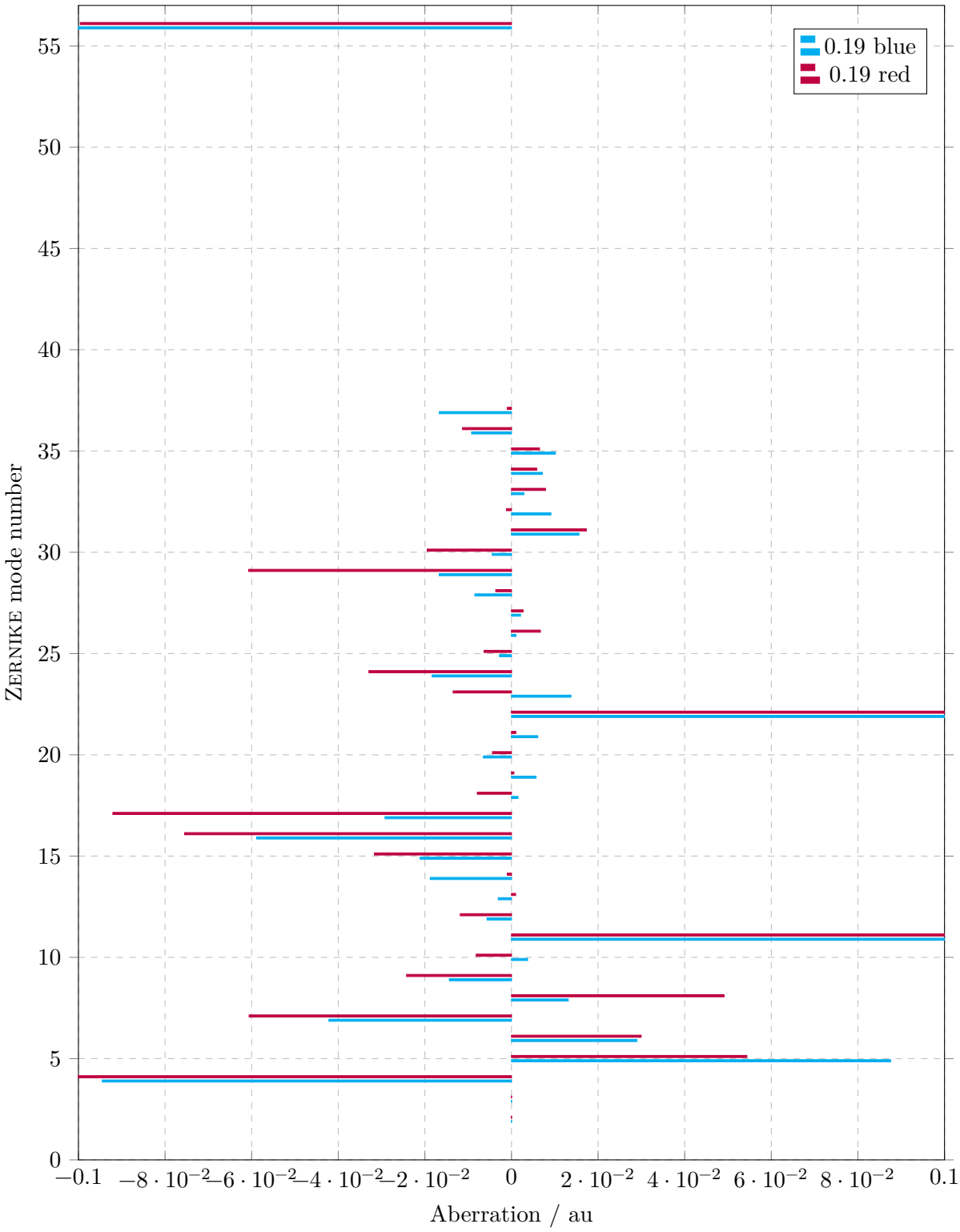


Figure 3.5: Red and blue channel Zernike modes $\{1 \ldots 37, 56\}$ versus aberrations of PSF model via phase retrieval, for correction collar setting of 0.19.

3.2.2 Cramér Rao Lower Bound

The Estimated Cramér Rao lower bound (CRLB) for different correction Collar settings (variing linestyles for 0.13, 0.17, 0.19) of the Olympus 1.5 NA objective; for X, Y and Z axis (top, middle, and bottom); for both red and blue channel (colors) is shown in Figure 3.6.

Additionally plots grouped by X, Y and Z axis are shown in Figure 3.7, as well as grouped by different correction Collar settings in Figure 3.8.

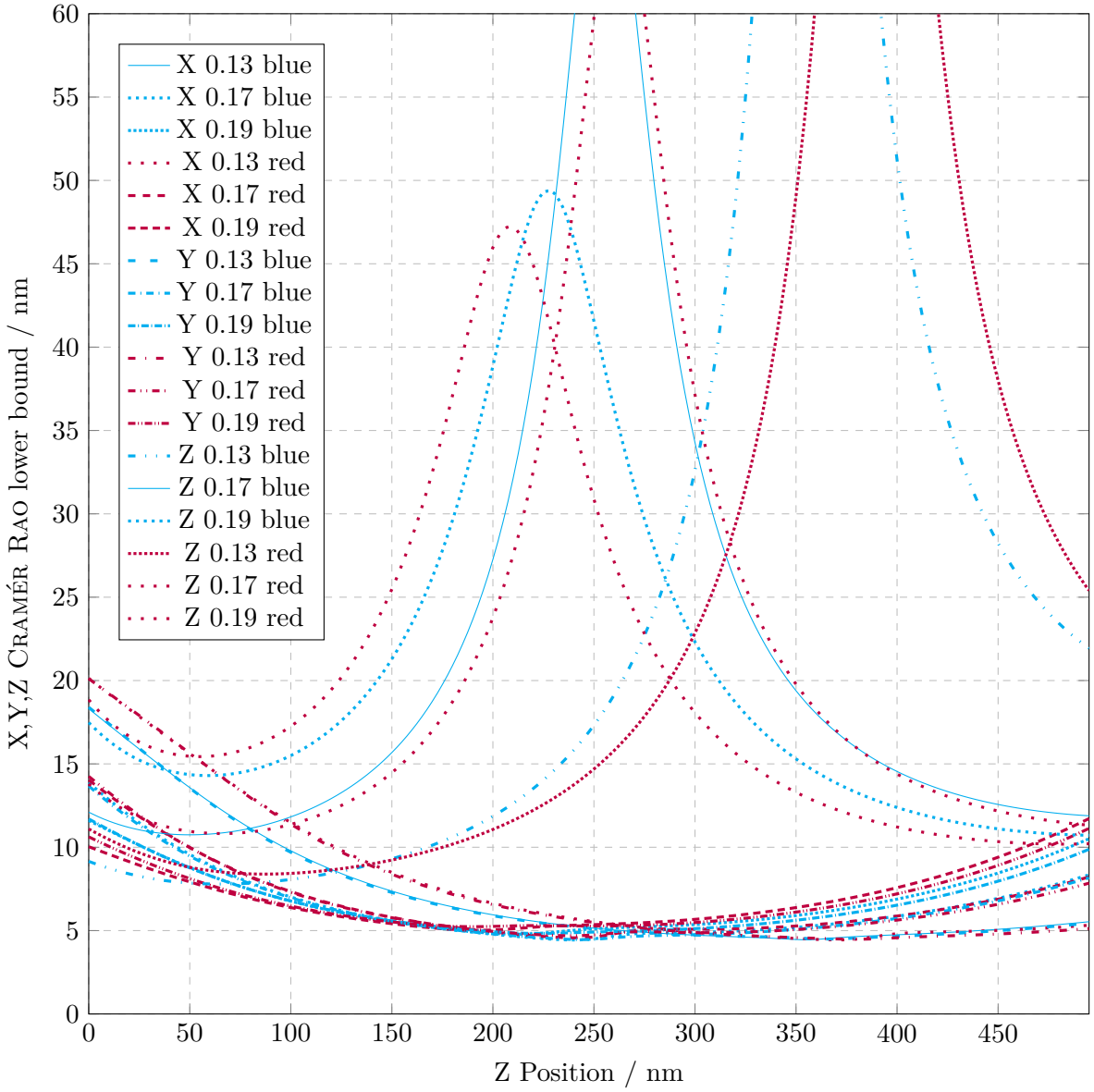


Figure 3.6: Estimated Cramér Rao *lower bound for different correction Collar settings (variing linestyles for 0.13, 0.17, 0.19) of the Olympus* 1.5 NA *objective; for X, Y and Z axis (top, middle, and bottom); for both red and blue channel (colors).*

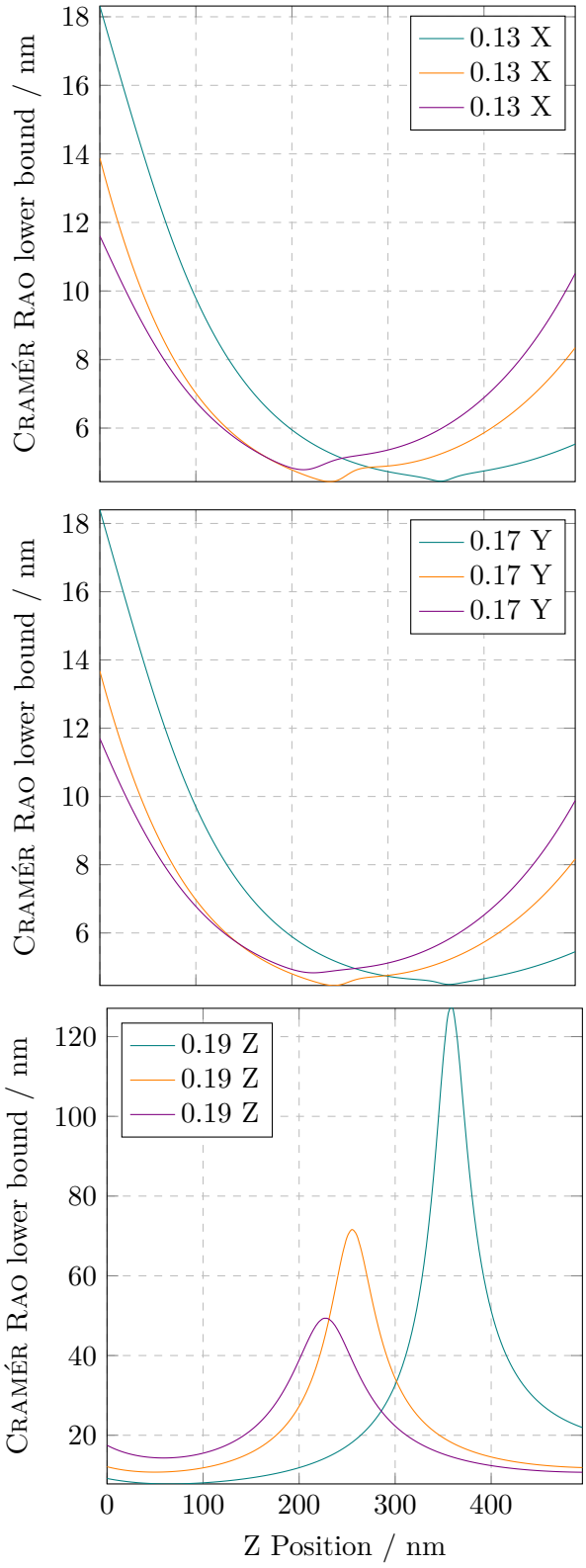


Figure 3.7: Estimated Cramér Rao lower bound for different correction Collar settings (teal: 0.13, orange: 0.17, purple: 0.19) of the Olympus 1.5 NA objective; grouped by X, Y and Z axis (top, middle, and bottom).

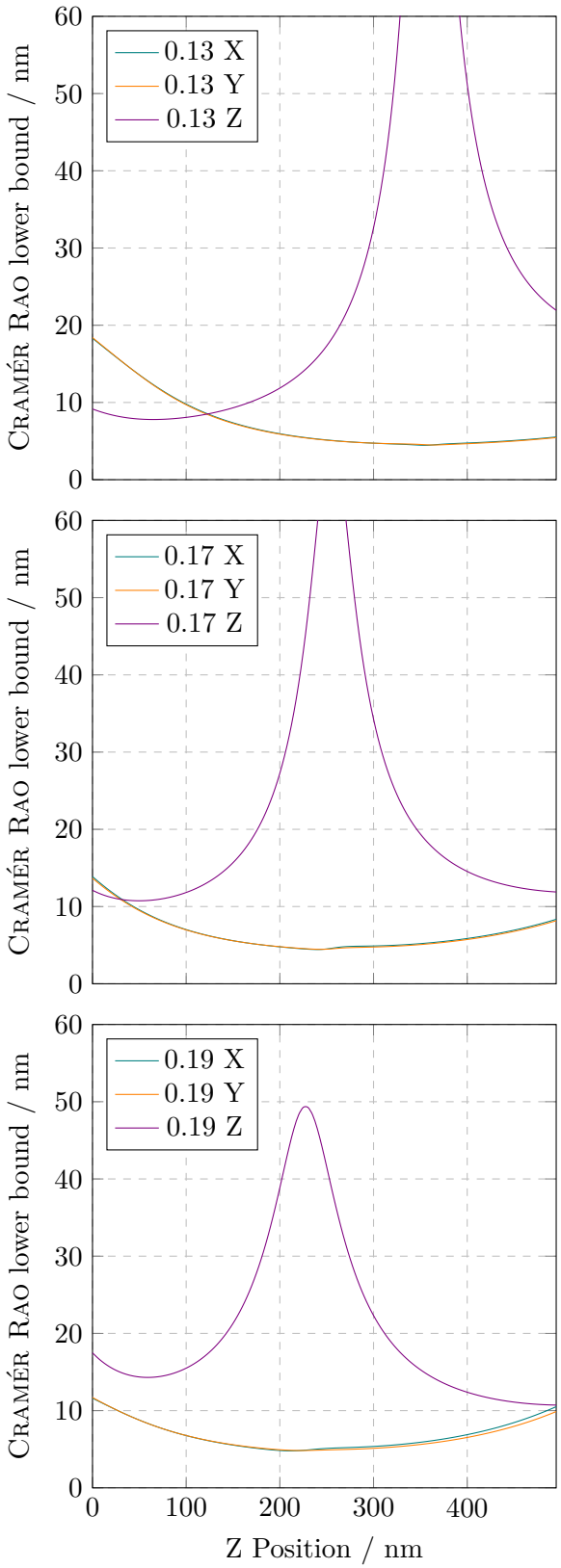


Figure 3.8: Estimated Cramér Rao lower bound for X, Y and Z axis (teal, orange and purple lines); grouped by different correction Collar settings (top: 0.13, middle: 0.17, bottom: 0.19) of the Olympus 1.5 NA objective.

3.3 Two Dimensional SMLM

In-focus, one colour, two dimensional SMLM is performed on all the samples, and evaluated by `Thunderstorm` via ImageJ. This gives a good x,y precision, an ideal reference for the later three dimensional localisation, whose x,y precision will most likely be much lower.

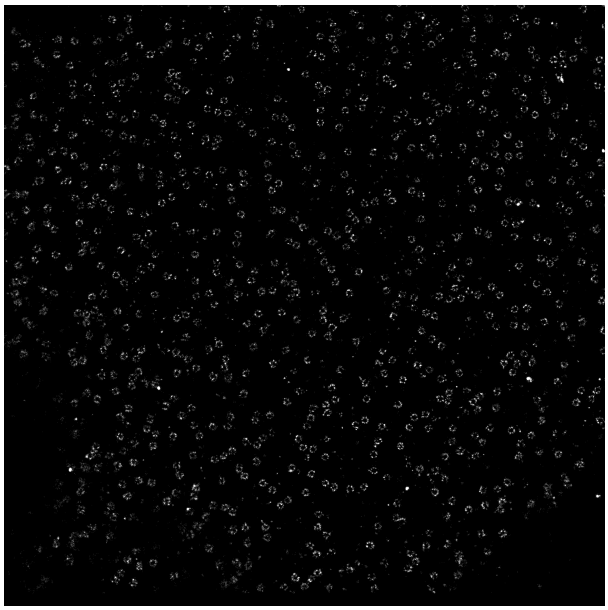


Figure 3.9: Example overview of a set of two dimensional SMLM data of NPCs, the labeled NPC tori are clearly visible.

Figure 3.9 and Figure 3.10 show an example overview (respective zoom) of a set of two dimensional SMLM data of NPCs using dSTORM with Gloxy buffer. The labeled NPC tori are clearly visible, even the more or less eight-fold symmetry of the nucleoporins is resolved.

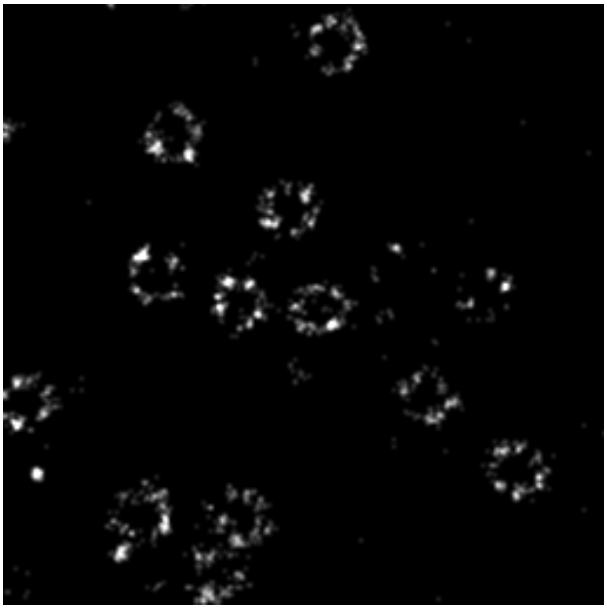


Figure 3.10: Zoom of Figure 3.9, Example of a set of two dimensional SMLM data of NPCs; The labeled NPC tori are clearly visible, even the more or less eight-fold symmetry of the nucleoporins is resolved.

3.4 Three Dimensional SMLM

Defocused, one colour, three dimensional SMLM is performed on some of the samples, and evaluated as described in Section 2.1. From the stack of 50k microscopy images, one obtains a list of localisations comprised of position (`x,y,z`), photon count (`n`), and a fit parameter (`fit`). Where the `sans serif` typeset letters refer to the variables in the code listed in this chapter.

In this section I want to describe our data analysis pipeline, in order to cut down the massive amount of initial localisations—in our example case over 130k—to distill it to the most meaningful conclusions.

As a proof of work we used NPCs as sort of a well defined biological test target. As these are used frequently for the purpose of validating a new method, it would be nice to quickly evaluate of *how good are the NPCs resolved*. The Section 3.5 is thus dedicated to find some metric for *best resolved* NPCs.

This analysis is performed in *Python*, and is freely available in my Git repository git; both as a plain python file (.py) as well as in the format of a *Jupyter Notebook* (.ipynb).

A slightly shortened version of the code is also shown in Appendix A for each Subsection. For the sake of readability we omit the code sections generating the shown plots, as they are mostly redundant. Of course the full code is available online.

3.4.1 Import

Here we import the used packages and the data file we obtain from running the SMLM algorithm.

3.4.2 Drift Correction

The *drift* of the setup over time can be estimated using *ImageJ*, and is then imported as `drift` drift. In the following block the drift correction is applied to all the 130k localisations, shown in Figure 3.11 as 3d plot of all the initial 370k drift corrected localisations.

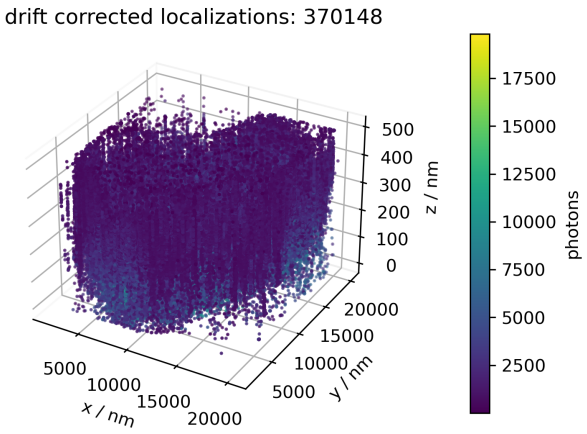


Figure 3.11: 3d plot of all 370148 drift corrected localisations, color coded by photon count.

3.4.3 Photon Counts

As a preliminary step it might prove useful to look at the photon count statistics, shown in Figure 3.12. Here we can easily see what the supposed intensity of a single molecule is; the peak, in our case about 2000. Those localisations below may be considered noise, those far above are probably overlapping or stacked molecules, so their intensities add up.

3.4.4 Filter

In this first filter we limit the dataset to the more meaningful points; like those with intensities between 2k and 7k. Also since we defocused for 500 m, only those `z` values between 0 and 499 can be considered realistic. Here 0 means directly attached to the glass substrate, so negative values would be *inside* the glass substrate; and thus need to be discarded as

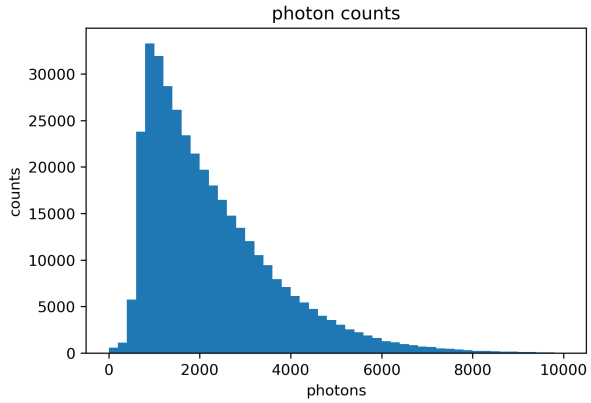


Figure 3.12: histogram of the photon count for the localisations.

unplausible. The dataset could be filtered by `min_fit`, but to our findings this does not contribute much.

Figure 3.13 shows a 3d plot of the remaining 154k localisations after we apply the filter, thus effectively shrinking down our exemplary data set by about 40%.

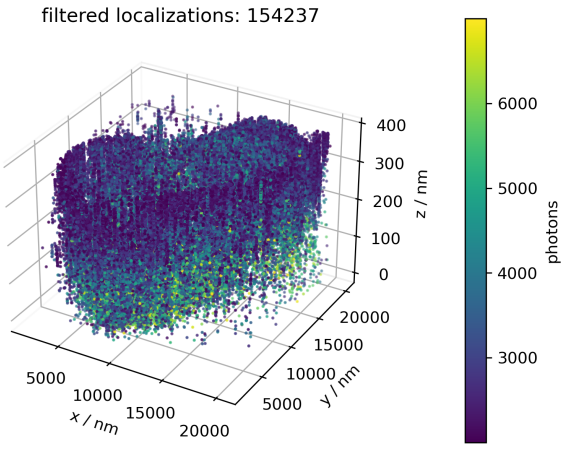


Figure 3.13: 3d plot of all 154237 filtered localisations, color coded by photon count.

3.4.5 Track Particles

The 50k frames we analyse here are taken with exposure of 20 m s, and 10 m s between consecutive exposures. Depending on the laser intensity and the buffer composition the bright state has a specific half-life. This leads to one exemplary molecule being *on* for some 30 m s (one frame) while some other is on for 60 m s; and so appears in two consecutive frames. Since the molecule in those two frames essentially is the same, we can *track* it over time: effectively averaging the location if present in multiple frames, thus reducing the amount of localisations while at the same time increasing their precision.

The parameters `sr` denotes the *search range*, how far apart two consecutive localisations are still considered *one* particle, this has to be adjusted based on the physical setup considering vibrations and the like.

Figure 3.14 shows a 3d plot of the remaining 98k localisations, after we track the particles. So the tracking step further shrinks down our exemplary data set by about 60%.

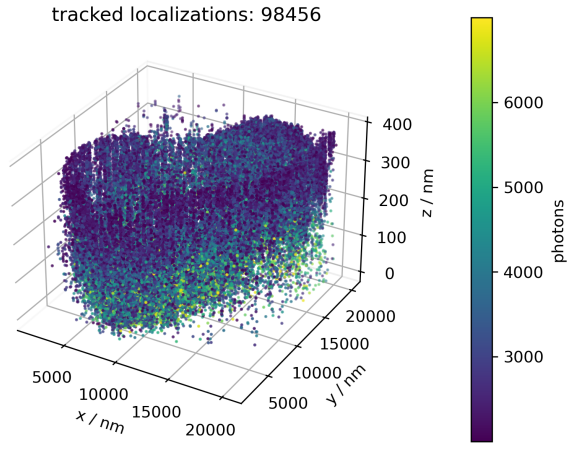


Figure 3.14: 3d plot of all 98456 filtered localisations, color coded by photon count.

3.5 NPC Analysis

We use NPCs as a test target for our method, thus we have some additional knowledge, like the geometry of each individual NPC: they comprise two stacked tori, each about 150 m diameter (in x,y direction), and 150 m apart (in z direction).

Now we can group our dataset with close to 100 thousand localisations to clusters of roughly this size in x,y (set `dim=2`). Note that for the sake of completeness, we include the possibility of clustering in 3d to spheres in x,y,z (set `dim=3`), but mind that x,y and z precision most often greatly varies (see Section sec).

The parameter `min_samples` denotes the minimum amount of constituents a cluster must have to be considered such.

Figure 3.14 shows a 3d plot of the localisations of 658 identified clusters, omitting all the other 35489 localisations as noise. So the clustering step further shrinks down our exemplary data set by about 30%.

3.5.1 Clustering

Figure 3.15 shows the localisations of the 658 identified clusters (omitting 35489 localisations as noise, due to not belonging to a cluster), color coded by photon count.

3.5.2 Cluster Analysis

For all the localisations within each of these identified clusters, we now derive the centroid position (`xmean,ymean,zmean`), with standard derivation (`xvar,yvar,zvar`). This enables the classification of the within-cluster distribution of localisations, alas how well they represent the anticipated NPC geometry.

To obtain this *quality*, we compose both the quantity `ringness`, denoting how well the cluster shapes a ring in x,y direction; as well as the quantity `twofold`, denoting how well the cluster shapes two stacked tori in z direction, basically forming a camel-curve in z direction. To break this down to one scalar each, we compute the root mean square (RMS) of the deviations of each localisations radius from the cluster centre (in x,y direction) from the known NPC radius (Lines 20–24). The quantity `twofold` is similarly comprised of a RMS deviation of each localisations z value from the cluster centre (in z direction) from the known NPC height. The mentioned parameters are thus called `npc_radius`, respective `npc_height`.

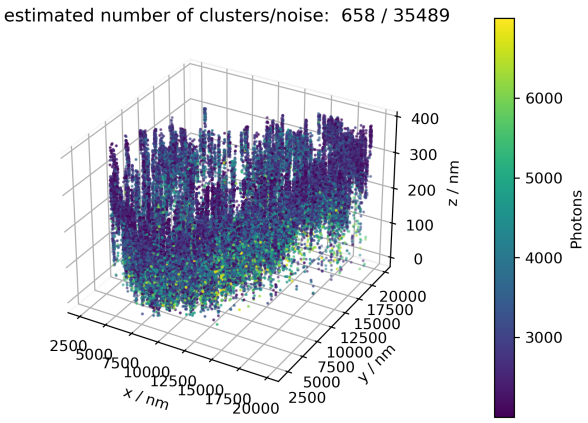


Figure 3.15: 3d plot of the localisations of the 658 identified clusters (omitting 35489 localisations as noise, due to not belonging to a cluster), color coded by photon count.

Figure 3.16 shows a broad overview of the location of the cluster centres (not the localisations within), color coded by photon count. This should be considered more of a short sanity check than a profound analysis.

3.5.3 Select Clusters

Now we possess all the information needed to evaluate the list of clusters; for example to sort for the lets say 10 most *ringlike* clusters. Or, by setting `sort = 'ringness + twofold'`, we may find the 10 *best* clusters in terms of both `ringness` and `twofold`—weighted equally—

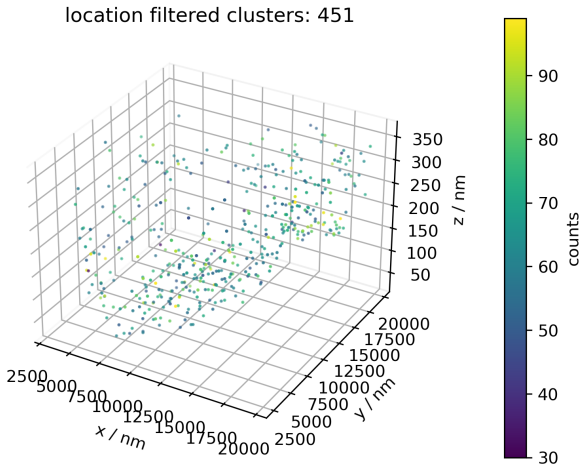


Figure 3.16: 3d plot of the positions of the 451 filtered clusters, color coded by photon count.

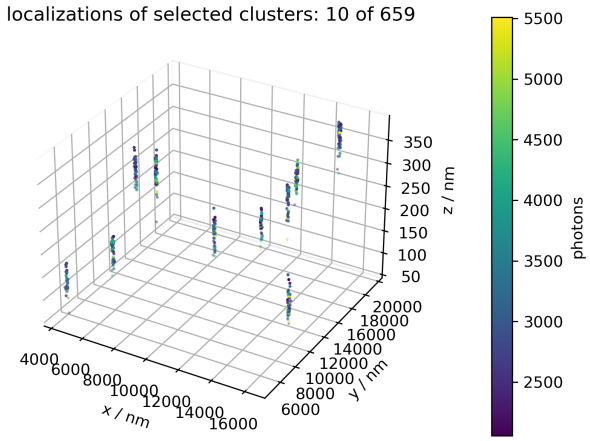


Figure 3.17: 3d plot of the localisations within the 10 best *clusters, color coded by photon count.*

which would comprise the 10 *overall best*, thus *most NPC like* clusters.

For sake of completeness we also include the x,y and z variances, even if they do not comprise a very meaningful parameter in the particular case due to the NPCs geometry.

Figure and Figure shows a 3d plot of the localisations within the 10 *best* clusters, color coded by photon count respective by cluster assignment.

3.5.4 X,Y,Z Histograms

As a further sanity check, we plot the histograms for selected clusters (`plot_cluster`), in x,y and z direction; to figure out if the sorting did work effectively—thus the higher sorted clusters are indeed *better* examples of NPCs.

Figure 3.19 shows exemplary histograms of the x,y (right) respective z distribution (left) of the localisations within the *best* cluster (top), the *worst* (bottom) and one in between.

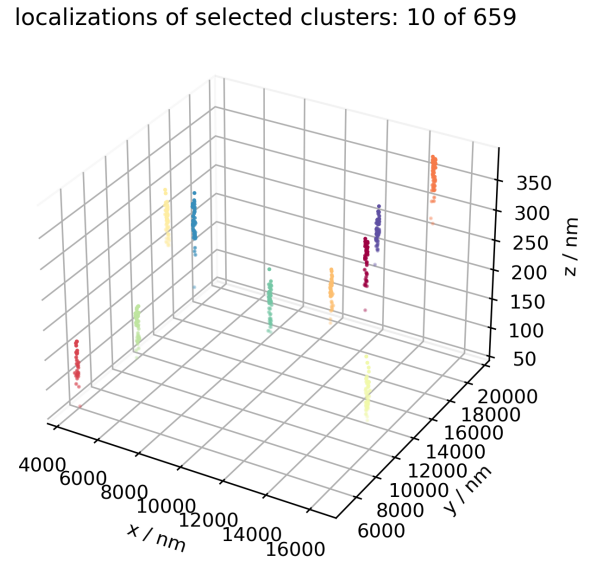


Figure 3.18: 3d plot of the localisations within the 10 best *clusters, color coded by cluster assignment.*

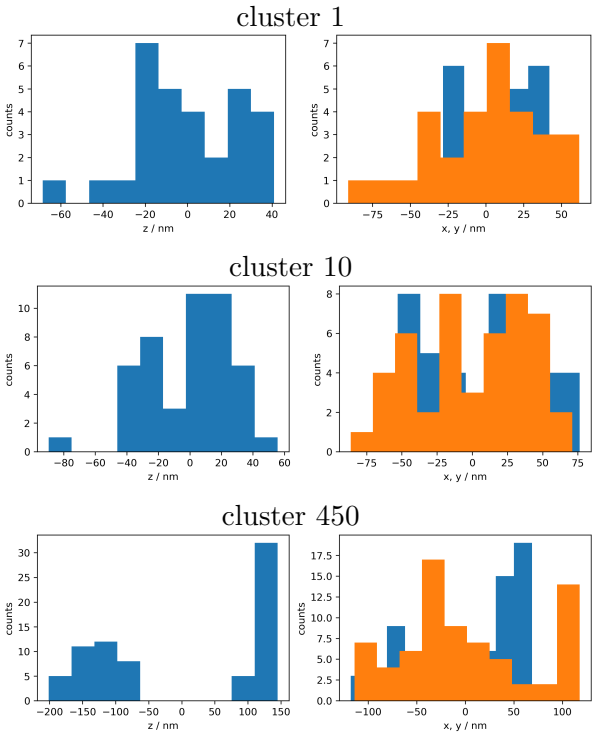


Figure 3.19: histogram of the x,y (right) respective z distribution (left) of the localisations within the best *cluster (top), the* worst *(bottom) and one in between.*

3.6 Dual Colour 3d SMLM

In order to perform two-colour SMLM we marked NPC and bead samples with an equal amount of both AF488 and AF647 fluorescence dyes.

A first measurement using the Gloxy buffer worked well for the red channel (for which the Gloxy buffer is designed) but did not lead to a good SNR in the blue channel.

Using the OxEA buffer described in Section sec, we got both channels working ok, but the overall SNR proved to be not good enough for drift analysis via ImageJ, or even particle tracking. There is no way we could try to recover the 3d transformation relating those data sets under this noisy conditions.

Thus we will analyse simulated data instead: based on three dimensional random data sets in Section 3.6.1, as well as based on two dimensional SMLM data sets in Section 3.6.2.

3.6.1 Simulation

In the first and second simulation the transformation relating both sets simply is a translation, respective a rotation and translation (rigid). Examples of sets related in such a way are shown in Figure 3.20 as purple and cyan dots on the top row for solely shifted and in the middle row for shifted and rotated sets (rigid). In a third simulation, the two sets are related by an affine transform adding shearing, reflection and scaling, shown in Figure 3.20 as purple and cyan dots on the bottom row.

All three simulations are now evaluated by both the algorithms `rig()` and `affine()` assuming a rigid respective an affine transformation. The resulting transformation, thus the projection from one set to the other is shown as magenta lines in all six sub figures of Figure 3.20.

The rigid recovery correctly estimates the rigid transforms of the first two simulations (top and middle row), but fails completely if the point clouds are actually *not* related by a rigid transform (bottom row)!

To test the algorithms stability to noise, the simulations are additionally salted slightly (additive noise to `q`). Due to the salt; and the fact, that the positions are pseudo-random points, the resulting transformations are slightly different for every simulation. The below shown comparison should be understood as an example—yet well suited to highlighting common problems.

Shift Projection

For the first simulation (top row), the initially (true) translation vector $\mathbf{t}_s^t$ and rotation matrix $\mathbf{R}_s^t$ is:

$$\mathbf{t}_s^t = \begin{pmatrix} 0 \\ 0.005 \\ 0.003 \end{pmatrix}, \mathbf{R}_s^t = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.1}$$

The under rigid assumption (left) recovered translation vector $\mathbf{t}_s^r$ and rotation matrix $\mathbf{R}_s^r$ is:

$$\mathbf{t}_s^r = \begin{pmatrix} 0.0047 \\ 0.0104 \\ 0.0030 \end{pmatrix}, \mathbf{R}_s^r = \begin{pmatrix} 0.999 & 0.000 & 0.000 \\ 0.000 & 0.999 & 0.000 \\ 0.000 & 0.000 & 0.999 \end{pmatrix} \tag{3.2}$$

t vec?

The under affine assumption (right) recovered translation vector $\mathbf{t}_s^a$ and rotation matrix $\mathbf{R}_s^a$ is:

$$\mathbf{t}_s^a = \begin{pmatrix} 0.0050 \\ 0.0108 \\ 0.0030 \end{pmatrix}, \mathbf{R}_s^a = \begin{pmatrix} 1.000 & 0.000 & 0.301 \\ 0.000 & 0.999 & -0.148 \\ 0.000 & 0.000 & 1.001 \end{pmatrix} \tag{3.3}$$

Rigid Projection

For the first simulation (top row), the initially (true) translation vector $\mathbf{t}_r^t$ and rotation ma-

trix $\mathbf{R}_r^t$ is:

$$\mathbf{t}_r^t = \begin{pmatrix} 0 \\ 0.005 \\ 0.003 \end{pmatrix}, \mathbf{R}_r^t = \begin{pmatrix} 0.342 & 0 & -0.940 \\ 0 & 1 & 0 \\ 0.940 & 0 & 0.342 \end{pmatrix} \tag{3.4}$$

The under rigid assumption (left) recovered translation vector $\mathbf{t}_r^r$ and rotation matrix $\mathbf{R}_r^r$ is:

$$\mathbf{t}_r^r = \begin{pmatrix} 0.0017 \\ 0.0103 \\ 0.0077 \end{pmatrix}, \mathbf{R}_r^r = \begin{pmatrix} 0.342 & 0.000 & -0.940 \\ 0.000 & 9.999 & 0.000 \\ 0.940 & 0.000 & 0.342 \end{pmatrix} \tag{3.5}$$

t vec?

The under affine assumption (right) recovered translation vector $\mathbf{t}_r^a$ and rotation matrix $\mathbf{R}_r^a$ is:

$$\mathbf{t}_r^a = \begin{pmatrix} 0.0014 \\ 0.0093 \\ 0.0069 \end{pmatrix}, \mathbf{R}_r^a = \begin{pmatrix} 0.3420 & 0.000 & -0.614 \\ 0.000 & 9.999 & 1.014 \\ 0.940 & 0.000 & 1.235 \end{pmatrix} \tag{3.6}$$

Affine Projection

For the first simulation (top row), the initially (true) translation vector $\mathbf{t}_a^t$ and rotation matrix $\mathbf{R}_a^t$ is:

$$\mathbf{t}_a^t = \begin{pmatrix} 0 \\ 0.005 \\ 0.003 \end{pmatrix}, \mathbf{R}_a^t = \begin{pmatrix} -0.684 & -0.342 & -0.940 \\ 0.000 & 1.000 & 0.000 \\ -1.879 & -0.940 & 0.342 \end{pmatrix} \tag{3.7}$$

The under rigid assumption (left) recovered translation vector $\mathbf{t}_a^r$ and rotation matrix $\mathbf{R}_a^r$ is:

$$\mathbf{t}_a^r = \begin{pmatrix} 0.0113 \\ 0.0059 \\ 0.0341 \end{pmatrix}, \mathbf{R}_a^r = \begin{pmatrix} -0.325 & -0.106 & -0.940 \\ -0.309 & 0.951 & 0.000 \\ -0.894 & -0.290 & 0.342 \end{pmatrix} \tag{3.8}$$

t vec?

The under affine assumption (right) recovered translation vector $\mathbf{t}_a^a$ and rotation matrix $\mathbf{R}_a^a$ is:

$$\mathbf{t}_a^a = \begin{pmatrix} -0.0058 \\ 0.0110 \\ -0.0129 \end{pmatrix}, \mathbf{R}_a^a = \begin{pmatrix} -0.684 & -0.342 & -0.566 \\ 0.000 & 1.000 & -0.319 \\ -1.879 & -0.940 & 1.364 \end{pmatrix} \tag{3.9}$$

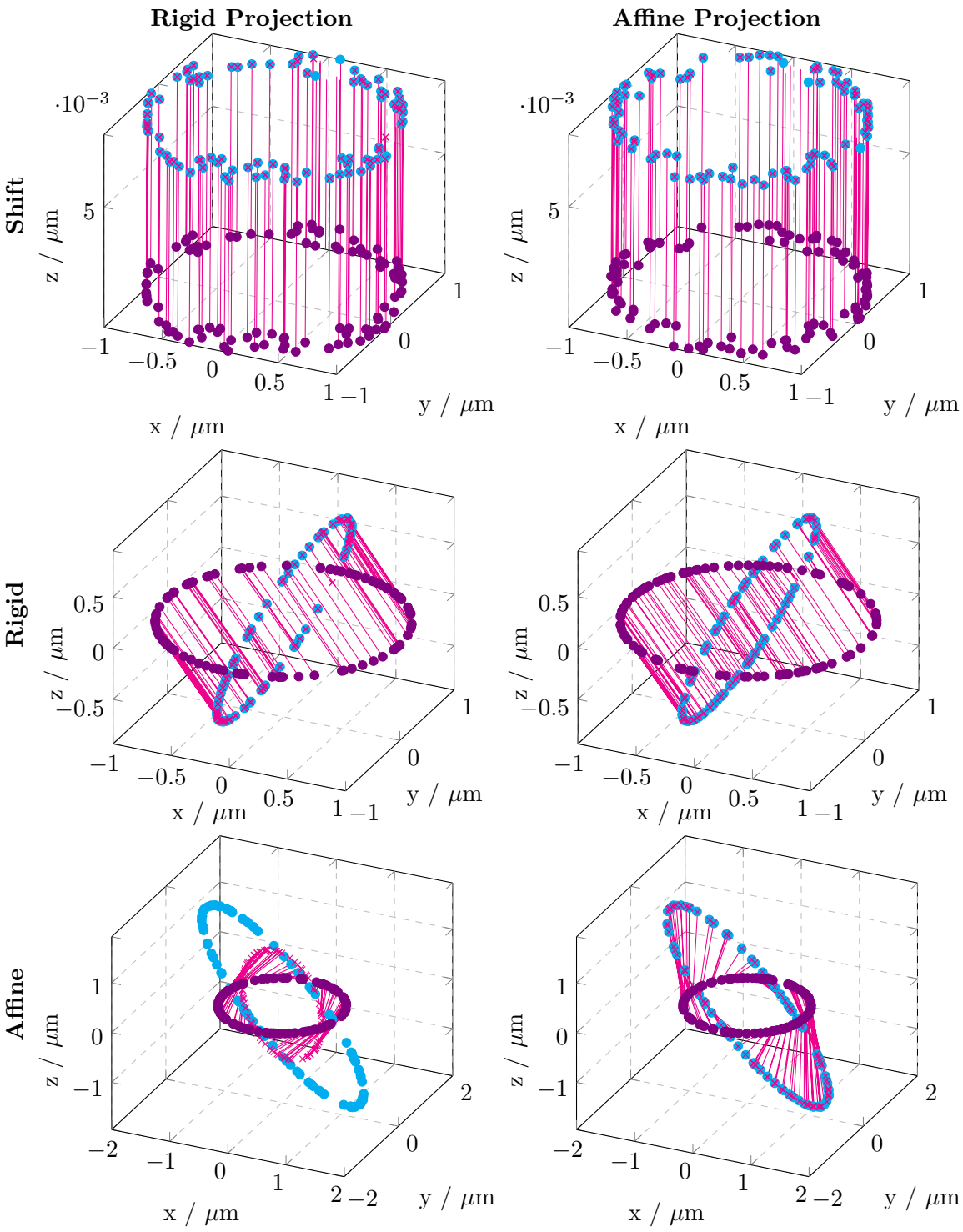


Figure 3.20: Demonstration of the recovery of rigid (left) respective affine (right) transformations, via recovering localisations of simulated two channel SMLM data; transformation (magenta) from red channel (violet) to blue channel (cyan); for the use cases of translation (top), rigid transformation (middle) and affine transformation (bottom). Obviously a rigid transformation may not correctly reconstruct an affine transformed data set (bottom left).

3.6.2 Projected 2d NPC

Lacking proper 3d dual colour SMLM data, we choose to use two-dimensional Thunderstorm SMLM data for a second proof of work.

A close look on such generated sets already shows interesting behaviour: the two sets are *shifted*. Obviously by utilising different wavelength regimes in a highly wavelength dependent process (optical path, refraction, objective, PSF, etc), one will eventually face some offset.

Rigid Projection

The rigid recovery worked quite flawlessly, the mean rotation matrix $\overline{\mathbf{R}}_r$ is almost unity within uncertainty. The mean translation vector $\overline{\mathbf{t}}_r$ shows a good variance over all the twelve analysed sets, with mean and standard derivation of:

$$\overline{\mathbf{t}}_r = \begin{pmatrix} 39 \\ 30 \\ 0 \end{pmatrix} \pm \begin{pmatrix} 10 \\ 4 \\ 0 \end{pmatrix} \mu\text{m} \tag{3.10}$$

$$\overline{\mathbf{R}}_r = \begin{pmatrix} 1.0000 & 0.0008 & 0 \\ -0.0008 & 1.000 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.11}$$

$$\sigma(\mathbf{R}_r) = \begin{pmatrix} 4 \cdot 10^{-7} & 4 \cdot 10^{-4} & 0 \\ 4 \cdot 10^{-4} & 4 \cdot 10^{-7} & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{3.12}$$

As an example of this rigid transformation, the localisations of the blue channel (blue $\times$), the red channel (violet $+$) and the projection from blue channel to red channel channel (magenta $\circ$) is shown in Figure 3.21, for one exemplary data set. The transformed blue channel localisations mostly align well with the red channel localisations.

Affine Projection

A similar analysis assuming an affine transformation though fails completely. This is due to the fact that the data sets are projected onto the z plane for analysis, so are now composing strictly parallel planes. This leads to the covariance matrices being *singular*, so the affine transform routine fails (or at least heavily crashes trying) to invert the covariance matrices. At least the results are so horribly wrong—if at all—that it is highly unlikely to *not* get suspicious immediately. In our case the translation vector is in the order of millimeters, which is not very plausible for two sets within the same cell. Thus we omitted the results.

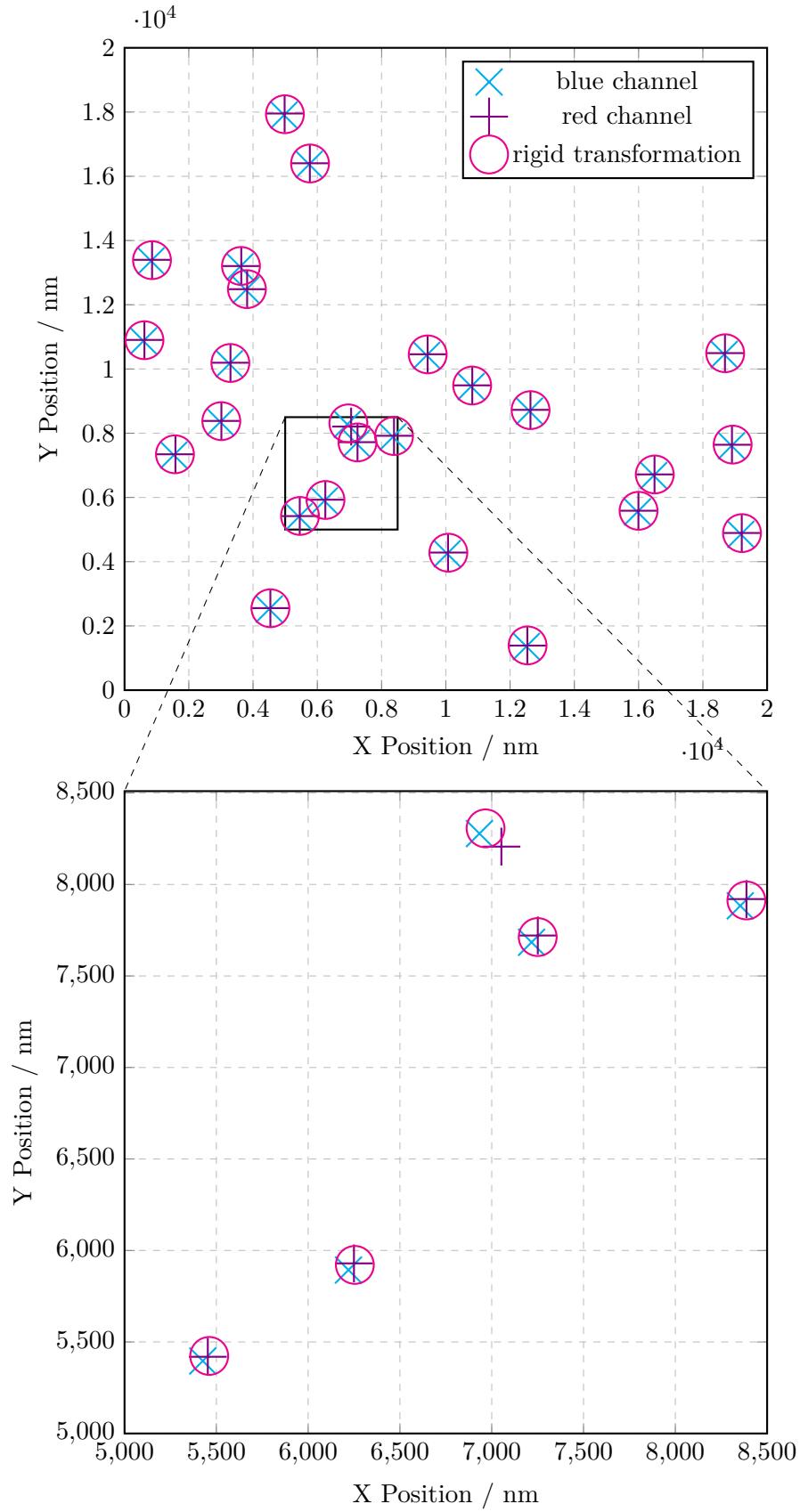


Figure 3.21: Demonstration of a rigid transformation of the localisations (magenta ∘) from blue channel (blue ×) to red channel (violet +); the transformed blue channel localisations mostly align well with the red channel localisations.

Chapter 4

Discussion

4.1 Dual Colour Optics

4.1.1 Zernike Modes

The image stacks look ok, even if not all are perfectly symmetrical–some errors are to be expected.

The estimated Zernike modes of the PSF are mostly plausible; based on the modes and the order of magnitude.

This is further backed by the comparison of the results grouped by the three correction collar settings $\{0.13, 0.17, 0.19\}$, in Figure 3.3, Figure 3.4 and Figure 3.5. It is quite obvious, that the results for both red and blue channel are in the same order of magnitude—if not quite similar—for most of the Zernike modes.

Yet the phase Retrieval program gave the error *high residual error* for both the red and the blue channel when using the correction collar settings $\{0.17, 0.19\}$.

4.1.2 Correction Collar

Based on the Figures 3.7 and Figure 3.8, and considering the fact—that we are interested in moderately defocusing (up to say 250 nm)—one might conclude, that the most preferable setting for the correction collar is 0.13.

Yet the recommended setting for our microscope setup is 0.17! Which is backed by Lukas, based on the PSF stacks: One would have guessed the 0.17 is more sensitive to z since it changes much more with different z positions than 0.13.

4.2 3d SMLM Analysis: NPC

4.2.1 Define Scalar Quality

The fully automatic evaluation of the NPC *quality*, shown in Section 3.5 surfaced mixed results. The key problems being for one, that there are quite many parameters involved; and secondly that the analysis proved very susceptible to changes of most of these parameters.

A quick glance at the histograms in Section 3.5.4 shows, that the cluster considered to be the *best* does not look better in fact, than most of the other clusters. We may well consider this approach failed.

Nevertheless we might have gained some valuable insight, into why this approach doesn't work. Reasons for this may be any and all of the following, for some of which we may suggest possible improvements.

numbers The clusters consist of very few localisations each, statistical analysis of so few elements have to be considered shaky. More constituents within each cluster would probably make this analysis more reliable.

quality The definitions of the two quality entities `ringness` and `twofold`, might not be derived well using RMS. A more sophisticated approach to quantify the clusters deviations to our known NPC geometry might work better, such as fitting the histogram in z direction to a Gaussian.

weighting The equal weighting of `ringness` and `twofold` quantities are canceling each other; Some clusters might look very *ring-like*, but are not at all stacked on top of each other, and vica versa. Unequal weighting based on empirical fine tuning might lesser this problem, but will most likely not solve it.

4.2.2 Secondary Filter

Some of the results of the NPC analysis might still prove themselves useful as a secondary filters. In reducing the clusters until only a few remain, effectively also finds the *best*.

high variance One might want to exclude clusters with an overly high variance, as these might be in fact two NPCs too close to each other to be accounted separately by the clustering algorithm.

low variance Quite similarly we might exclude those clusters showing extremely low variance, under suspicion of being well concentrated—yet noise, not representing a NPC at all.

spread Likewise we may exclude clusters spreading far in z, due to our knowledge of the NPC heigth, as well as in x,z due to the NPCs well defined radius.

In the end, possessing a list of the clusters position allows for a much easier way to plot some of them manually, than to zoom in on a plot of thousands of localisations repeatedly.

4.3 Dual Colour Projection

Simply put, the algorithm `rig()` finds the *best* rigid transformation, whatever the real relation is. This poses a serious caveat for experimental data, since one is usually not in possession of any form of ground truth to check whether the result is plausible or not.

4.3.1 SMLM Simulations

At first glance at the plots in Figure 3.20, the rigid as well as the shift problem (top and middle row) are similarly well solved either by rigid or affine assumption; their projection of the set `q` neatly correspond with set `p`. Also quite obviously the recovery of a assumed rigid transform of sets *not* related in such a way (bottom row) does not yield a significant solution.

A closer inspection of the respective recovered translation vectors $\mathbf{t}$ and rotation matrices $\mathbf{R}$ in Section 3.6 enables a deeper understanding of the involved misconceptions—that ultimately question this approach.

Shift

Comparison of the respective recovered translation vectors $\mathbf{t}_s^{r,a}$ and rotation matrices $\mathbf{R}_s^{r,a}$ to the ground truth $\mathbf{t}_s^t$ and $\mathbf{R}_s^t$ show, that both sort of agree—for solely shifted data sets. The affine projection is a little farther off.

Rigid

Comparison of the respective recovered translation vectors $\mathbf{t}_r^{r,a}$ to the ground truth $\mathbf{t}_r^t$ shows, that even the rigid recovery does not represent the ground truth well, with the affine projection being even farther off still.

Yet a comparison of the respective recovered rotation matrices $\mathbf{R}_r^{r,a}$ to the ground truth $\mathbf{R}_r^t$ show good agreement for the rigid recovery. The affine recovery is slightly off, but somewhat acceptable.

Rigid

Comparison of the respective recovered translation vectors $\mathbf{t}_a^{r,a}$ to the ground truth $\mathbf{t}_a^t$ show very little agreement for both the rigid and the affine recovery.

A comparison of the respective recovered rotation matrices $\mathbf{R}_a^{r,a}$ to the ground truth $\mathbf{R}_a^t$ shows show good agreement for the affine recovery, with the rigid recovery completely failing.

Chapter 5

Conclusion

5.1 Dual Colour Optics

5.1.1 PSF Model & Zernike

The Zernike modes for both red and blue channel are estimated via phase retrieval of in-focus measurements of beads at various depths. This yields a full PSF model to be used for defocus 3d SMLM.

5.1.2 Correction Collar

The best setting of the Correction Collar for the Olympus 1.5 NA objective is shown to be 0.13. Here we find the preferable compromise between x,y precision and z precision, in the regime between about 0 and 250 µm.

5.2 Three Dimensional SMLM

The analysis performed in Section 3.4 is well suited to greatly reduce the localisations, just by omitting unplausible data and noise; in our example from initially 370k to below 100k, or to about 26%.

5.3 NPC Analysis

Our approach of fully automatic evaluation of the NPC *quality*, may well be considered failed.

Nevertheless we might have gained some valuable insight, as some of the results of the NPC analysis might still prove themselves useful as a secondary filters.

5.4 Dual Colour 3d SMLM

5.5 Dual Colour Buffer

The prepared samples, both beads and NPCs have been evaluated using Gloxy buffer and OxEA buffer.

The two channels (red and blue) showed gross differences in, blinking speed, *dark state* and *bright state* lifetime, Signal and background magnitude as well as SNR.

The data obtained from this dual colour three dimensional localisation was not of high enough precision to be analysed meaningfully. We thus propose further investigation in buffers used for dual colour SMLM.

5.5.1 Simulation

Concluding, one best uses an rigid recovery for a rigid problem, or an affine recovery for an affine problem, as shown in Table 5.1 and Table 5.2.

Due to the convexity of the problem (there are infinitely many transformations), the recovery of a translation vector under the assumption of at least dome rotation involved is not really possible, as shown in Table 5.1 and Table 5.2. This poses a dangerous caveat

Table 5.1: Quality of the recovered rotation translation vectors (transformation) matrices $\boldsymbol{t}^{r,a}_{s,r,a}$.

Problem	rig()	affine()
Shift	ok	ok
Rigid	bad	bad
Affine	worse	bad

Table 5.2: Quality of the recovered rotation (transformation) matrices $\boldsymbol{R}^{r,a}_{s,r,a}$

Problem	rig()	affine()
shift	ok	ok
rigid	good	bad
affine	worse	good

for the dual colour SMLM analysis, where we cannot rule out rotations.

5.5.2 Projected 2d NPC

As a second proof of work, we tested both rigid and affine recovery on several real dual colour SMLM data sets of two color in focus measurements of different cells successively recorded in the same sample.

Mind that this is a simulation, the actual transformation between both sets using the SMLM analysis might look completely different.

Rigid

The rigid recovery worked quite flawlessly, the mean rotation matrix $\overline{\mathbf{R}}_r$ is almost unity within uncertainty. Yet the mean translation vector $\overline{\mathbf{t}}_r$ shows a high variance (at least on x axis) over all the twelve analysed sets, with mean and standard derivation of:

$$\overline{\mathbf{t}}_r = \begin{pmatrix} 39 \\ 30 \\ 0 \end{pmatrix} \pm \begin{pmatrix} 10 \\ 4 \\ 0 \end{pmatrix} \text{µm} \tag{5.1}$$

$$\overline{\mathbf{R}}_r = \begin{pmatrix} 1.0000 & 0.0008 & 0 \\ -0.0008 & 1.000 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{5.2}$$

$$\sigma(\mathbf{R}_r) = \begin{pmatrix} 4 \cdot 10^{-7} & 4 \cdot 10^{-4} & 0 \\ 4 \cdot 10^{-4} & 4 \cdot 10^{-7} & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{5.3}$$

Affine

A similar analysis assuming an affine transformation though fails completely, this method simply cannot work on two dimensional data.

Appendix A

Code

A.1 3d SMLM Analysis: NPC

A.1.1 Import

```
#@markdown ##import core
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import matplotlib as mpl
from mpl_toolkits.mplot3d import Axes3D
import scipy.io
from sklearn.cluster import DBSCAN
from sklearn import metrics
from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler

#@markdown ##import jupyter
#%matplotlib ipympl
#%matplotlib widget
#%matplotlib interactive
% matplotlib inline
import trackpy
#import sdt
#from sdt import io, chromatic, multicolor, brightness

## local
wd = 'data/210422_npc_red_defocus/'
data = pd.read_csv( wd + 'cell1_tr1000_def500.csv',
header = None,
names=["x", "y", "z", "n", "bg","fit","id","frame"] )
```

A.1.2 Drift Correction

```
#@markdown ## import & scale drift
#@markdown > set **magnification** via factor in drift.

drift = pd.read_csv( wd + 'day2_cell1_driftValues.csv')

drift['Y2']=drift['Y2']*146.6
drift['Y3']=drift['Y3']*146.6
drift['X2']=np.round(drift['X2'])
drift['X3']=np.round(drift['X3'])

#@markdown ## apply drift correction
```

```

for i in range(len(drift)-1):
fr=data[(data['frame']>=drift['X2'].iloc[i]) &
(data['frame']<drift['X2'].iloc[i+1])]
fr['y']=fr['y']-drift['Y2'].iloc[i]
fr['x']=fr['x']-drift['Y3'].iloc[i]
data[(data['frame']>=drift['X2'].iloc[i]) &
(data['frame']<drift['X2'].iloc[i+1])]=fr
```

A.1.3 Photon Counts

```
#@markdown ## Check: photon counts
#@markdown > set `max_photons` accordingly (default: 10000).
max_photons = 10000 #@param {type:"slider", min:0, max:40000, step:1000}

fig = plt.figure()
plt.hist( data['n'], bins=50, range=( 0, max_photons ) )
```

A.1.4 Filter

```
#@markdown ## filter
#@markdown > set `min_photons` and `max_photons` accordingly (default: 2000 < photons < 7
#@markdown > set `min_z` and `max_z` accordingly (default: 0 < z < 499).\
#@markdown > set `min_fit` accordingly (default: 6e6).
min_photons = 2000 #@param {type:"slider", min:0, max:40000, step:1000}
max_photons = 7000 #@param {type:"slider", min:0, max:40000, step:1000}
min_z = 0 #@param {type:"slider", min:0, max:500, step:1}
max_z = 384 #@param {type:"slider", min:0, max:500, step:1}
min_fit = 7192000 #@param {type:"slider", min:0, max:1e7, step:1000}

fdata = data[ ( data['n'] > min_photons ) &
( data['n'] < max_photons ) &
( data['z'] > min_z ) &
( data['z'] < max_z ) &
( data['fit'] < min_fit ) ]
```

A.1.5 Track Particles

```
#@markdown ## track all in x,y
#@markdown > set `sr` to wanted search range (default: 50).\
#@markdown > set `mem` to wanted memory (default: 10).
sr = 50 #@param {type:"slider", min:0, max:100, step:1}
mem = 10 #@param {type:"slider", min:0, max:100, step:1}
```

```
6
7   linkedxy = trackpy.link_df( fdata,
8   pos_columns = ["x","y","z"],
9   search_range = sr,
10  memory = mem )
11
12  particles = linkedxy.groupby( "particle" ).aggregate( np.mean )
13  std_pos = linkedxy.groupby( "particle" ).aggregate( 'std' )
14  particles["length"] = linkedxy.groupby( "particle" ).apply( len )
15  particles["z_std"] = std_pos['z'].copy()
16  particles["x_std"] = std_pos['x'].copy()
17  particles["y_std"] = std_pos['y'].copy()
```

A.2 3d SMLM Analysis: NPC

A.2.1 Clustering

```
#@markdown ## compute dbscan
#@markdown > set `dim = 2` for clustering in x and y (default:).\
#@markdown > set `dim = 3` for experimental clustering in 3d; be aware that x,z and y pre
#@markdown > set `eps` (default: 200).\
#@markdown > set `min_samples` (default: 10).
dim = "2" #@param [2, 3]
eps = 100 #@param {type:"slider", min:0, max:500, step:10}
min_samples = 50 #@param {type:"slider", min:1, max:100, step:1}

alocalisations = localisations.to_numpy()
alocalisations[ :, 0:2 ]

db = DBSCAN( eps, min_samples ).fit( alocalisations[ :, 0:int( dim ) ] )
core_samples_mask = np.zeros_like( db.labels_, dtype=bool )
core_samples_mask[ db.core_sample_indices_ ] = True
labels = db.labels_
localisations[ "cluster" ] = labels

# count clusters (ignore noise if present)
n_clusters_ = len( set( labels ) ) - ( 1 if -1 in labels else 0 )
n_noise_ = list( labels ).count( -1 )

#print('Estimated number of clusters: %d' % n_clusters_)
#print('Estimated number of noise points: %d' % n_noise_)

nlocalisations = localisations.loc[ localisations['cluster'] == -1 ]
clocalisations = localisations.loc[ localisations['cluster'] != -1 ]
```

A.2.2 Cluster Analysis

```
#@markdown ## analyse clusters
#@markdown > set `npc_radius` to NPC radius /nm (default: 50).\
#@markdown > set `npc_radius` to NPC haight /nm (default: 150).
npc_radius = 50 #@param {type:"slider", min:0, max:500, step:1}
npc_height = 25 #@param {type:"slider", min:0, max:500, step:1}

clabels = set(labels)
cnames = [ "counts", "xmean", "ymean", "zmean", "nmean", "xvar", "yvar", "zvar",
"nvar", "label", "ringness", "twofold" ]
clusters = pd.DataFrame( index = clabels, columns = cnames, dtype="float64" )
```

```
clusters[ "label" ] = clabels

for k in clabels:
tmp = localisations.loc[ localisations['cluster'] == k ]
clusters.loc[ k, "counts" ] = len( tmp )
for label in [ "x", "y", "z", "n" ]:
clusters.loc[ k, label+"mean"  ] = np.mean( tmp.loc[ :, label ] )
clusters.loc[ k, label+"var"  ] = np.var( tmp.loc[ :, label ] )

## xy: radius (distance to centroid)
rad = np.sqrt( ( tmp.loc[ :, "x" ] - clusters.loc[ k, "xmean" ] )**2 +
( tmp.loc[ :, "y" ] - clusters.loc[ k, "ymean" ] )**2 )
## xy: radius rms deviation from NPC radius
clusters.loc[ k, "ringness"  ] = np.sqrt( sum( ( rad - npc_radius )**2 ) )

## z: radius (distance to centroid)
rad = abs( tmp.loc[ :, "z" ] - clusters.loc[ k, "zmean" ] )
## z: radius rms deviation from NPC radius
clusters.loc[ k, "twofold"  ] = np.sqrt( sum( ( rad - npc_height )**2 ) )

#@markdown ## filter clusters
#@markdown > set `count_threshold` to min elements (counts) in cluster (default: 30)\
#@markdown > set `diameter_threshold` to max x,y (radius) deviation of cluster from NPC d
#@markdown > set `twofold_threshold` to max z (radius) deviation of cluster from NPC heig
#@markdown > set `xyvar_threshold` to wanted x,y variance (default: 1e4)\
#@markdown > set `zvar_threshold` to wanted z variance (default: 1e4)
count_threshold = 100 #@param {type:"slider", min:0, max:200, step:1}
diameter_threshold = 500 #@param {type:"slider", min:0, max:500, step:10}
twofold_threshold = 1000 #@param {type:"slider", min:0, max:1000, step:10}
xyvar_threshold = 100000 #@param {type:"slider", min:0, max:1e5, step:1e3}
zvar_threshold = 100000 #@param {type:"slider", min:0, max:1e5, step:1e3}

fclusters = clusters[ ( clusters['counts'] < count_threshold ) &
( clusters['ringness'] < diameter_threshold ) &
( clusters['twofold'] < twofold_threshold ) &
( clusters['xvar'] < xyvar_threshold ) &
( clusters['yvar'] < xyvar_threshold ) &
( clusters['zvar'] < zvar_threshold ) ]
```

A.2.3 Select Clusters

```
#@markdown ## select best clusters & plot localisations
#@markdown > set `show_clusters` to wanted number of best clusters (default: 100).\
```

```
#@markdown > set `sort` to sort the best clusters (default: ringness + twofold).
show_clusters = 10 #@param {type:"slider", min:0, max:1000, step:1}
sort = 'ringness + twofold' #@param [ "ringness + twofold", "twofold", "ringness", "xyvar
if sort == "xvar": # sort by variance
sclusters = fclusters.sort_values( "xvar" )
elif sort == "xyvar": # sort by x and y variance using least squares
sclusters = fclusters.loc[
( fclusters.xvar ** 2 + fclusters.yvar ** 2 ).sort_values().index ]
elif sort == "ringness": # sort by ringness (deviation to ringness)
sclusters = fclusters.sort_values( "ringness" )
elif sort == "twofold": # sort by ringness (deviation to ringness)
sclusters = fclusters.sort_values( "twofold" )
elif sort == "ringness + twofold": # sort by ringness (deviation to ringness)
sclusters = fclusters.loc[
( fclusters.twofold ** 2 + fclusters.ringness ** 2 ).sort_values().index ]

show_clusters = min( show_clusters, len( sclusters ) )
selected_clusters = sclusters["label"].iloc[ 0:show_clusters ]

fig = plt.figure()
ax = fig.add_subplot( projection = '3d' )
for clus in selected_clusters:
flocalisations = localisations[ ( localisations['cluster'] == clus ) ]
ff = ax.scatter( flocalisations['x'],
flocalisations['y'],
flocalisations['z'],
s=1 ,c = flocalisations['n'] )
```

A.2.4 X,Y,Z Histograms

```
#@markdown ## Check: z distribution
#@markdown > set `plot_cluster` to wanted cluster (default: 0).
plot_cluster = 3  #@param {type:"slider", min:0, max:100, step:1}
plot_cluster = min( plot_cluster, len( sclusters ) -1 )

tmp = localisations.loc[ localisations['cluster'] ==
sclusters.loc[ sclusters.index[ plot_cluster ],
"label" ] ]

z = tmp.z - np.mean( tmp.z )

fig, axes = plt.subplots(2, 1, figsize=(4, 7) ) # figsize=(4, 12)
```

```
13
14    axes[0].hist( tmp.z - np.mean( tmp.z ) )
15    axes[0].set_xlabel('z / nm')
16    axes[0].set_ylabel('counts')
17    #axes[0].set_title( 'z' )
18
19    axes[1].hist( tmp.x - np.mean( tmp.x ) )
20    axes[1].hist( tmp.y - np.mean( tmp.y ) )
```

A.3 Dual Colour 3d SMLM

A.3.1 Cockpit: cockpit.m

```
#!/bin/octave
## images fluorescence microscopy / 3d position / offsets.
## splice red/green/blank fluorescence microscopy images already located
## into red/green matrices for further analysis, find rigid transform,
## do statistics on all files.
## @ moritz siegel

## init
close all
clear all
clc
graphics_toolkit('gnuplot')
#graphics_toolkit('qt')
#graphics_toolkit('fltk')

## \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
## soft settings \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
global hd = "~/biophysics" # parent directory
global wd = "data/210318_beads_2_colors_in_focus"
global fn = "position"
global nn = ""; # additional luminosity suffix  ( default: "" )
method = "rigid"; # /char, "rigid","affine", method to reconstruct
global minpts = 10; # minimum number of points needed in its neighbourhood to consider it a
global dist = 300; #/nm, distance on which neighbourhood is calculated.( default: 200 )
nmax = 15; #/num, maximum number of files analysed.( default: 10 )
global verbose = true; # /bool, plot for error checking? ( default: true )
exclude = [ 6, 7, 8, 11, 12 ]
channel_order0 = [ 0, 1, 2 ];
channel_order = repmat( channel_order0, nmax, 1 );
channel_order( 3, : ) = [ 2, 0, 1 ];
channel_order( 4, : ) = [ 1, 2, 0 ];
channel_order( 6, : ) = [ 2, 0, 1 ];
channel_order( 7, : ) = [ 1, 2, 0 ];
channel_order( 8, : ) = [ 2, 0, 1 ];
channel_order( 12, : ) = [ 2, 0, 1 ];
## end of settings: hands off /////////////////////////////////////////////////
## ///////////////////////////////////////////////////////////////////////////

disp( 'warmup \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\ ' );
## lets move it move it
```

```
addpath( hd ) # function & stuff needed
stamp = strftime("%Y_%m_%d_%H%M%S", localtime (time ())); # create timestamp for saving fil
global nwd = sprintf( "%s/%s/all_%s%s%s", hd, wd, method, stamp, nn ); # concat working dir
mkdir( nwd )
chdir( nwd )
global rf = fopen ( "kockpit.log", "w" );

disp( 'cycle \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\ ' );
rotations = nan( nmax, 3, 3 );
translations = nan( nmax, 3 );
for n = 1 : nmax
  if ( any( exclude == n ) )
    disp( sprintf( "    warning: skipping file positions%d.csv", n ) );
    continue
  endif
  disp( sprintf( "    importing positions%d.csv", n ) );
  disp( '    load \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\ ' );

  ## "id","frame","x [nm]","y [nm]","sigma [nm]","intensity [photon]","offset [photon]","bk
  try
    pos = csvread( sprintf( "%s/%s/%s%d.csv", hd, wd, fn, n ) );
  catch
    continue
  end_try_catch


  disp( '    analyse \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\ ' );

  [ centroid_red, centroid_blue_transformed, centroid_blue, red, blue, rotation, translatio

  ## save optimal rigid transform for comparison
  rotations( n, :, : ) = rotation;
  translations( n, : ) = translation;
endfor
```

A.3.2 Splice Channels: ampel.m

```
#!/bin/octave
function[ centroid_red, centroid_blue_transformed, centroid_blue, red, blue, rotation, tran
  ## This function "ampel" splices the given table including 3d positions
  ## etc from images of twocolor fluorescence microscopy into red & blue
  ## color chanels based on the frame number. It uses "dbscan" to cluster
  ## the images, and sverages the positions within each
```

```
## cluster. Depending on "method" either finds best rigid transform, or
## best affine transform correlating the red and blue sets via least
## squares (singular-value-decomposition).
## @ moritz siegel

global nwd
global rf
global minpts
global dist

## sort red/blue/empty of many frames.
red = blue = empty = zeros( size( pos ));
nb = nr = ne = 1;
for k = 1 : size( pos, 1 )

  ## exclude blank lines (text).
  if ( all( pos( k, : ) == 0 ) )
    disp( sprintf( 'warning: line %d is empty; skipping.', k ));
    fprintf( rf, 'warning: empty line; skipping.\n' );
    continue
  endif

  ## red pill / blue pill?
  channel = mod( pos( k, 2 ), 3 );
  switch ( channel )
    case channel_order(1)
      blue( nb, : ) = pos( k, : );
      nb = nb + 1;
    case channel_order(2)
      empty( ne, : ) = pos( k, : );
      ne = ne + 1;
    case channel_order(3)
      red( nr, : ) = pos( k, : );
      nr = nr + 1;
  endswitch
endfor
disp( sprintf( 'sorted localisations:\n %d red\n %d blue\n %d empty', nr, nb, ne ) );
fprintf( rf, 'sorted localisations:\n %d red\n %d blue\n %d empty\n', nr, nb, ne );

## analyse clusters & average centroids.
[ assignments_blue, c_blue ] = dbscan( blue( 1:300, 3:4 ), minpts, dist );
for k = 1 : c_blue
  idx = find( assignments_blue == k );
  centroid_blue( k, 1:2 ) = mean( blue( idx, 3:4 ), 1 );
```

```
  endfor
  [ assignments_red, c_red ] = dbscan( red( 1:300, 3:4 ), minpts, dist );
  for k = 1 : c_red
    idx = find( assignments_red == k );
    centroid_red( k, 1:2 ) = mean( red( idx, 3:4 ), 1 );
  endfor

  ## need to ditch clusters if the sets are not equally sized.
  assert( c_red == c_blue, "clusters are not equally sized" );

  ## recover transform.
  switch( method )
    case "rigid"
      ## we only have 2d data currently, introduce linear dependent z component.
      centroid_red = [ centroid_red, zeros( c_red, 1 ) ]
      centroid_blue = [ centroid_blue, zeros( c_blue, 1 ) ]
      [ rotation, translation ] = rig( centroid_blue, centroid_red )
    case "affine"
      ## we only have 2d data currently, introduce noise for z component,
      ## matrix cant be singular, else cholesky decomposition fails.
      z = 1 + 1e-3 * rand( c_red, 1 );
      centroid_red = [ centroid_red, z ];
      centroid_blue = [ centroid_blue, z ];
      [ rotation, translation ] = affine( centroid_blue, centroid_red );
  endswitch

  ## transform blue set to red set.
  centroid_blue_transformed = ( rotation * centroid_blue' )' + translation;

endfunction
```

A.3.3 Rigid Transformation: rig.m

```
  #!/bin/octave
  function [ rotation, translation, s ] = rig( p, q )
  ## this function "rig(p,q)" finds the optimal rigid transform in
  ## 3-dimensional euclidian space, using least squares and
  ## single-value-decomposition. Given a 3xn matrix (set of n 3d
  ## positions), it returns the rotation matrtix "rotation", and the
  ## translation vector "translation".
  ##
  ## K. S. Arun, T. S. Huang and S. D. Blostein, "Least-Squares Fitting of
  ## Two 3-D Point Sets," in IEEE Transactions on Pattern Analysis and
```

```
## Machine Intelligence, vol. PAMI-9, no. 5, pp. 698-700, Sept. 1987,
## doi: 10.1109/TPAMI.1987.4767965.
##
## moritz siegel @ 210322

## check stuff.
assert( nargin == 2 && size( p ) == size( q ), ...
"need 2 identical input matrices\n" );
assert( size( p, 2 ) == 3, "input matrix p must be nx3\n" );
assert( size( q, 2 ) == 3, "input matrix q must be nx3\n" );
n = size( p, 1 );
assert( n > 2, "need at least 3 points\n" );

## 1) center to get rid of translation.
centroid_p = mean( p, 1 );
centroid_q = mean( q, 1 );
p_shifted = p - repmat( centroid_p, n, 1 );
q_shifted = q - repmat( centroid_q, n, 1 );

## 2) solve least squares problem for best rotation.
## ------------------------------------------------

## covariance matrix.
h = p_shifted' * q_shifted;

## singular value decomposition.
[ u, s, v ] = svd( h );
rotation = v * u'

## reflection? theres more to that.
if ( det( rotation ) < 0 )
printf( "warning: det(r) < 0\n" );
if ( any( s( : ) ) < 0 )
printf( "found reflection, correcting.\n" );
v( :, 3 ) = -v( :, 3 );
rotation = v * u';
else
printf( "error: single-value-decomposition failed! \
provided data seems is too noisy for least-squares\n." );
endif
endif

## 3) compute translation.
translation = centroid_q - ( rotation * centroid_p' )';
```

```
55
56    endfunction
```

A.3.4 Affine Transformation: affine.m

```
1   #!/bin/octave
2   function [ affine_rotation, translation, s ] = affine( p, q )
3     ## this function "affine(p,q)" finds the optimal affine transform in
4     ## 3-dimensional euclidian space, using least squares and
5     ## single-value-decomposition. Given a 3xn matrix (set of n 3d
6     ## positions), it returns and the rotation matrtix "affine_rotation",
7     ## and the translation vector "translation".
8     ##
9     ## K. S. Arun, T. S. Huang and S. D. Blostein, "Least-Squares Fitting of
10    ## Two 3-D Point Sets," in IEEE Transactions on Pattern Analysis and
11    ## Machine Intelligence, vol. PAMI-9, no. 5, pp. 698-700, Sept. 1987,
12    ## doi: 10.1109/TPAMI.1987.4767965.
13    ##
14    ## Berthold K. P. Horn, "Closed-form solution of absolute orientation
15    ## using unit quaternions," J. Opt. Soc. Am. A 4, 629-642 (1987)
16    ##
17    ## @ moritz siegel
18
19    global hd
20    global wd
21    global nwd
22
23    ## check stuff.
24    assert( nargin == 2 && size( p ) == size( q ), ...
25            "need 2 identical input matrices\n" );
26    assert( size( p, 2 ) == 3, "input matrix p must be nx3\n" );
27    assert( size( q, 2 ) == 3, "input matrix q must be nx3\n" );
28    n = size( p, 1 );
29    assert( n > 2, "need at least 3 points\n" );
30
31    ## 1) center to get rid of translation.
32    centroid_p = mean( p, 1 );
33    centroid_q = mean( q, 1 );
34    p_shifted = p - repmat( centroid_p, n, 1 );
35    q_shifted = q - repmat( centroid_q, n, 1 );
36
37    ## 2) orthogonal reduction.
38    ## ------------------------
```

```
39
40    ## variance (?) matrices.
41    s_p = p_shifted' * p_shifted;
42    s_q = q_shifted' * q_shifted;
43
44    ## square-roots of the covariance matrices.
45    ## choleski decomposition: "A -> LL*" , any advantage over sqrtm()?
46    #s_sqrt_p = sqrtm( s_p );
47    #s_sqrt_q = sqrtm( s_q );
48    s_sqrt_p = chol( s_p, "lower" );
49    s_sqrt_q = chol( s_q, "lower" );
50
51    ## left division. "x\y" is conceptually equivalent to the expression
52    ## "inv(x) * y" but it is computed without forming the inverse of x.
53    ## if the system is not square, or if the coefficient matrix is
54    ## singular, a minimum norm solution is computed.
55    p_orthogonal = ( s_sqrt_p \ p_shifted' )';
56    q_orthogonal = ( s_sqrt_q \ q_shifted' )';
57
58    ## "p" and "q" are now solely related by a rotational matrix
59    ## "r = s_inv_sqrt_q * affine * s_sqrt_p" (no inverse!).
60
61    ## 3) solve least squares problem for best rotation.
62    ## ------------------------------------------------
63
64    ## covariance matrix again.
65    h = p_orthogonal' * q_orthogonal;
66
67    ## singular value decomposition.
68    [ u, s, v ] = svd( h );
69    rotation = v * u';
70
71    ## reflection? theres more to that.
72    if ( det( rotation ) < 0 )
73      printf( "warning: det(rotation) < 0\n" );
74      if ( any( s( : ) ) < 0 )
75        printf( "found reflection, correcting.\n" );
76        v( :, 3 ) = - v( :, 3 );
77        rotation = v * u';
78      else
79        printf( "error: single-value-decomposition might have failed! \
80  provided data seems is too noisy for least-squares.\n" );
81      endif
82    endif
```

```
83
84    ## 4)  reverse the rotation to the non-orthogonal affine transform using
85    ## right division: "x/y" is conceptually equivalent to the expression
86    ## "x * inv(y)" but it is computed without forming the inverse of x.
87    affine_rotation = ( s_sqrt_q * rotation ) / s_sqrt_p;
88
89    ## 5) compute translation.
90    translation = centroid_q - ( affine_rotation * centroid_p' )';
91
92    ## save stuff for plotting.
93    ## chdir( nwd )
94    ## save p_shifted.mat p_shifted;
95    ## save q_shifted.mat q_shifted;
96    ## save p_orthogonal.mat p_orthogonal;
97    ## save q_orthogonal.mat q_orthogonal;
98    ## save rotation.mat rotation
99    ## save affine_rotation.mat affine_rotation
100   ## save translation.mat translation
101
102 endfunction
```

A.3.5 Clustering: dbscan.m

```
1  #!/bin/octave
2  ## This reposetory contains a simple implementation of DBSCAN algorithm
3  ## using GNU OCTAVE.  DBSCAN is a popular clustering algorithm. It
4  ## creates clusters on a spatial data depending on two parameters:
5  ## MinPoints: minimum number of points needed in its neighbourhood to
6  ## consider it as a valid data(not noise).  EPS: A distance on which
7  ## neighbourhood is calculated.
8  ## For more info:
9  ## https://github.com/devil1993/DBSCAN
10 ## https://en.wikipedia.org/wiki/DBSCAN
11 ## http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=1A6A7A85AF3F43BCBF66D847FEC8F80
12 function [assignments,C] = dbscan(X,minpts,EPS)
13   C = 0;
14   assignments = zeros(size(X)(1),1);
15   clustered = zeros(size(X)(1),1);
16   for i=1: size(X)(1)
17     if(clustered(i)==1)
18       continue;
19     endif
20     clustered(i)=1;
```

```
    isneighbour = [];
    neighbourcount = 0;
    for j=1: size(X)(1)
      dist = sqrt(sum((X(i,:)-X(j,:)).^2));
      if(dist<EPS)
        neighbourcount++;
        isneighbour = [isneighbour j];
      endif
    endfor
    if(neighbourcount<minpts)
      continue;
    else
      C++;
      assignments(i) = C;
      for k=isneighbour
        if(clustered(k)==0)
          clustered(k) = 1;
          _isneighbour = [];
          _neighbourcount = 0;
          for j=1: size(X)(1)
            dist = sqrt(sum((X(k,:)-X(j,:)).^2));
            if(dist<EPS)
              _neighbourcount++;
              _isneighbour = [_isneighbour j];
            endif
          endfor
          if(_neighbourcount>=minpts)
            isneighbour = [isneighbour _isneighbour];
          endif
        endif
        assignments(k) = C;
      endfor
    endif
  endfor
endfunction
```

A.3.6 Simulate Dual Chanel 3d SMLM Data: simulate.m

```
#!/bin/octave
## @ moritz siegel

clear all
close all
```

```
clc
graphics_toolkit('gnuplot')
#graphics_toolkit('qt')
#graphics_toolkit('fltk')

## \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
## soft settings \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
global hd = "~/biophysics"; # parent directory
global wd = "data";
global nn = "AFFINE"; # additional luminosity suffix  ( default: "" )
method = "affine"; # /char, "rigid","affine", method to reconstruct
#method = "affine"; # /char, "rigid","affine", method to reconstruct
eps = 1e-8; # precision ( default: 1e-8 )
## end of settings: hands off ///////////////////////////////////////////////////
## ////////////////////////////////////////////////////////////////////////////


disp( 'init \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\ ' );

## lets move it move it.
addpath( hd ) # function & stuff needed
stamp = strftime("%Y_%m_%d_%H%M%S", localtime (time ())); # create timestamp for saving fil
global nwd = sprintf( "%s/%s/simulation_%s_%s_%s", hd, wd, method, stamp, nn ); # concat wo
mkdir( nwd )
chdir( nwd )

## init first point cloud. introduce noise for z component, matrix cant
## be singular (e.g. points on a plane), else cholesky decomposition fails.
n = 100;
theta = 100 * rand( n, 1 );
p = [ cos( theta ), sin( theta ), 1e-3 * rand( size( theta ) ) ];

## introduce noise
salt = [ 1e-2*rand( size( theta ) ), 1e-2*rand( size( theta ) ), 1e-5*rand( size( theta ) )
q = p + salt;

## define affine transforms & derive second point cloud.
simulated_translation = [ 0, 0.005, 0.003 ];
reflect = [ -1, 0, 0; 0, 1, 0; 0, 0, 1 ];
scale = [ 2, 0, 0; 0, 1, 0; 0, 0, 1 ];
theta = 70;
rot = [ cosd(theta), 0, -sind(theta); 0, 1, 0; sind(theta), 0, cosd(theta) ];
shear = [ 1, 0.5, 0; 0, 1, 0; 0, 0, 1 ];
simulated_transform = eye( 3 );
```

```
simulated_transform = simulated_transform * rot;
simulated_transform = simulated_transform * reflect;
simulated_transform = simulated_transform * scale;
simulated_transform = simulated_transform * shear;
q = ( simulated_transform * q' )';
q = q + simulated_translation;

## check determinant.
#assert( det( simulated_transform ) > 0, "det( simulated_transform ) <= 0, rerun" );


disp( 'analyse \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\ ' );

## recover transform
switch( method )
case "rigid"
    [ rekovered_transform, rekovered_translation ] = rig( p, q );
case "affine"
    [ rekovered_transform, rekovered_translation ] = affine( p, q );
endswitch

if ( any( any( ( simulated_transform - rekovered_transform ) > eps ) ) )
  disp( "warning: failed to recover affine transform" );
endif
```

bibliography

List of Tables

List of Figures

Acronyms

CRLB Cramér Rao lower bound. 8, 16

dSTORM direct stochastic optical reconstruction microscopy. 6, 7, 9, 19, 55

Gloxy glucose oxidase dSTORM buffer. 7, 10, 19, 33

MEA β-mercaptoethylamine hydrochloride. 7

MLE maximum likelihood estimation. 6

NPC nuclear pore complex. 9, 10, 20, 22, 23, 25, 30, 31, 33

OxEA OxyFlour β-mercaptoethylamine hydrochloride dSTORM buffer. 7, 10, 33

PBS phosphate-buffered saline. 7

PSF point spread function. 6, 8

RMS root mean square. 22, 31

sCMOS scientific complementary metal-oxide-semiconductor. 4

SMLM single molecule localisation microscopy. 4–10, 19, 20, 25, 27, 28, 33, 34, 57

SNR signal-to-noise ratio. 10, 33

STORM stochastic optical reconstruction microscopy. 4–6

SVD singular value decomposition. 8

TIRF total internal reflection fluorescence. 4

index