

a

PROJEKTARBEIT / STUDENT PROJECT

THREE DIMENSIONAL DUAL CHANNEL SINGLE MOLECULE LOCALISATION MICROSCOPY

concluded at the

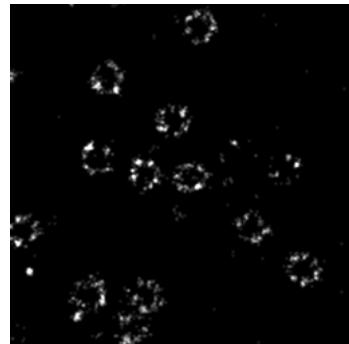
TECHNISCHE UNIVERSITÄT (TU) WIEN

advised by

LUKAS VELAS

of

MORITZ SIEGEL



Vienna
June 28, 2021

moritz.siegel@tuwien.ac.at
<https://github.com/imrahiliyas/biophysics>

Contents

1	Introduction	4
1.1	Fluoro	4
1.2	2d SMLM	4
1.3	3d SMLM	4
1.4	dSTORM	4
2	Methods	5
2.1	Maximum-likelihood estimation	5
2.2	PSF	5
2.3	Zernike	5
2.4	TIRF	5
2.5	dSTORM	5
2.5.1	Gloxy Buffer	5
2.5.2	OxEA Buffer	6
2.6	Dual Channel Optics	6
2.6.1	Correction Collar	6
2.6.2	Cramer Rao Lower Bound	7
2.7	Dual Channel SMLM	7
2.7.1	Rigid Transform	7
2.7.2	Affine Transform	7
2.7.3	Simulation	8
2.7.4	Projected 2d nuclear pore complex (NPC)	8
3	Results	9
3.1	Fluoro	9
3.1.1	dSTORM	9
3.1.2	Zernike Modes	9
3.1.3	Cramer Rao Lower Bound	15
3.2	3d SMLM Analysis	18
3.2.1	Import	18
3.2.2	Drift Correction	18
3.2.3	Photon Counts	18
3.2.4	Filter	18

3.2.5	Track Particles	19
3.3	3d single molecule localisation microscopy (SMLM) Analysis: NPC	19
3.3.1	Clustering	20
3.3.2	Cluster Analysis	20
3.3.3	Select Clusters	21
3.3.4	X,Y,Z Histograms	21
3.4	Dual Channel 3d SMLM	22
3.4.1	Simulation	22
3.4.2	Projected 2d NPC	25
4	Discussion	27
4.1	SMLM	27
4.2	Dual Channel Optics	27
4.2.1	Zernike Modes	27
4.2.2	Correction Collar	27
4.3	3d SMLM Analysis	27
4.4	3d SMLM Analysis: NPC	27
4.4.1	Automation	27
4.4.2	Secondary Filter	28
4.5	Dual Channel Projections	28
4.5.1	SMLM Simulations	28
5	Conclusion	30
5.1	Dual Channel Optics	30
5.1.1	Zernike Modes	30
5.1.2	Correction Collar	30
5.2	dSTORM Buffer	30
5.3	3d SMLM Analysis	30
5.4	3d SMLM Analysis: NPC	30
5.5	Dual Channel 3d SMLM	30
5.5.1	Simulation	30
5.5.2	Projected 2d NPC	30
A	Code	32
A.1	3d SMLM Analysis: NPC	33
A.1.1	Import	33
A.1.2	Drift Correction	33
A.1.3	Photon Counts	34
A.1.4	Filter	34
A.1.5	Track Particles	34
A.2	3d SMLM Analysis: NPC	36
A.2.1	Clustering	36
A.2.2	Cluster Analysis	36
A.2.3	Select Clusters	37

A.2.4	X,Y,Z Histograms	38
A.3	Dual Channel 3d SMLM	40
A.3.1	Cockpit: cockpit.m	40
A.3.2	Splice Channels: ampel.m	41
A.3.3	Rigid Transformation: rig.m	43
A.3.4	Affine Transformation: affine.m	45
A.3.5	Clustering: dbscan.m	47
A.3.6	Simulate Dual Chanel 3d SMLM Data: simulate.m	48
	List of Tables	51
	List of Figures	52
	Acronyms	54

Chapter 1

Introduction

we want two color dyed SMLM data in sync, how to get there?

Single Molecule Localisation Microscopy (SMLM) is a technique of fitting a full Point Spread Functions (PSF) to a stack of images containing reasonably spaced fluorescence signals.

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 de-focus, one can guess the defocus and thus the z coordinate of the fluorescence molecule.

1.1 Flouro

Flouro

1.2 2d SMLM

SMLM

1.3 3d SMLM

SMLM

1.4 dSTORM

stochastic optical reconstruction microscopy (dSTORM)

Chapter 2

Methods

2.1 Maximum-likelihood estimation

For the Analysis of 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 :

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

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

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

depending on the form of the point spread function (PSF) model h and on the parameter vector θ :

$$\theta = (\beta, s, x_0, y_0, z_0) \quad (2.3)$$

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 θ is estimated by finding a minimum of the total negative log-likelihood function using the MATLAB tool `fminunc`:

$$\theta = \arg \min_{\theta} \sum_k \mu_{\theta,k} - n_k \ln \mu_{\theta,k} \quad (2.4)$$

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} :

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

2.3 TIRF

[TIRF](#)

2.4 dSTORM

[dSTORM](#)

2.4.1 Gloxy Buffer

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 (β -mercaptoproethylamine hydrochloride (MEA), Sigma-Aldrich).
- 10 vol% of a 250 g L⁻¹ solution of glucose.
- 0.5 mg ml⁻¹ glucose oxidase.
- 40 mg ml⁻¹ catalase (Sigma-Aldrich).
- in phosphate-buffered saline (PBS), pH 7.6 .

2.4.2 OxEA Buffer

Dual channel Fluorescence microscopy poses novel challenges to find a proper dSTORM buffer, that works for both excitation wavelengths—thus two distinct fluorophores AF647 and AF488. 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 solution (L1375, Sigma-Aldrich).
- in PBS, pH adjusted to 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.

pH

The pH of the OxEA buffer is checked using both broad range pH testing strips, and a digital pH meter, to be between pH 7 and pH 8.

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

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

2.5 Dual Channel Optics

It is possible to use different excitations to simultaneously measure different fluorescence markers, but for that the Point Spread Functions (PSF) has to be 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.

2.5.1 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 21?].

2.5.2 Cramer Rao Lower Bound

In order to find the best Correction Collar setting of the Olympus 1.5 NA objective for SMLM, the Cramer Rao Lower Bound (cramer rao lower bound (CRLB)) is computed using the program by [Jesacher et al 21?], with prior estimated Point Spread Functions (PSF) via phase retrieval [Jesacher et al 21?].

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 signal strength (2500) respective background (100).

2.6 Dual Channel SMLM

The ultimate goal of this thesis is to facilitate dual channel SMLM, by alternating excitation with different lasers. 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.

2.6.1 Rigid Transform

In first order approximation, this offset may be thought as linear, so the two sets are re-

lated 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 single-value-decomposition (SVD) based on [cit](#). In this quick 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).

2.6.2 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 ??, still using least squares and single-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 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 orthogonalised (Lines 51–56); so that p and q are now solely related by a rotational matrix.

2.6.3 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 Sec-

tion 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.

2.6.4 Projected 2d NPC

As a second proof of work, we tested both rigid and affine recovery on several real dual channel SMLM data sets of NPCs of two-color in focus measurements of different cells successively recorded in the same sample. The fluorescence images are analysed with *Thunder-storm*, 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 Flouro

sample image

3.1.1 dSTORM

2d npc images

3.1.2 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.

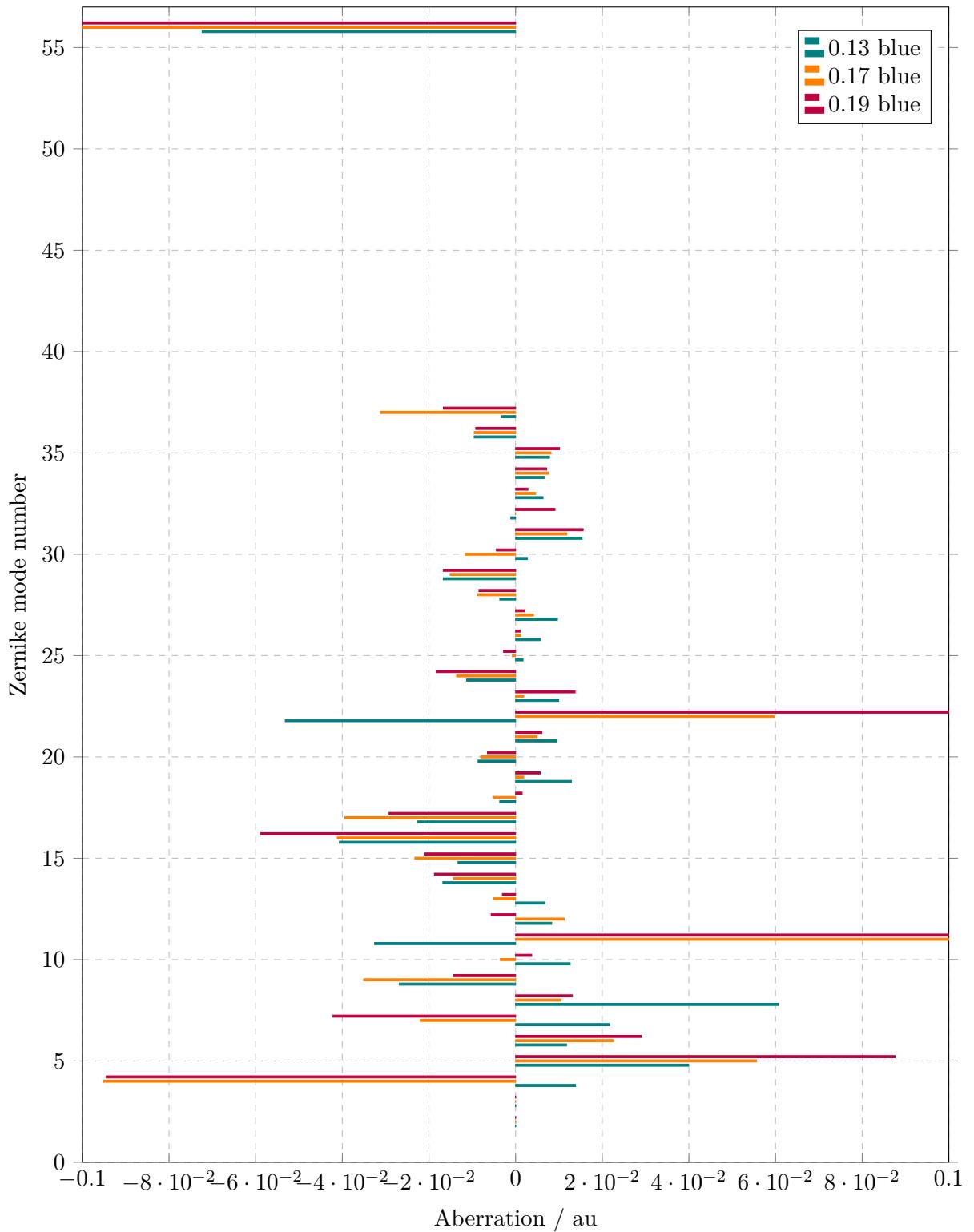


Figure 3.1: Blue channel Zernike modes {1 ... 37, 56} versus aberrations of PSF model via phase retrieval, for all three correction collar settings (0.13, 0.17, 0.19).

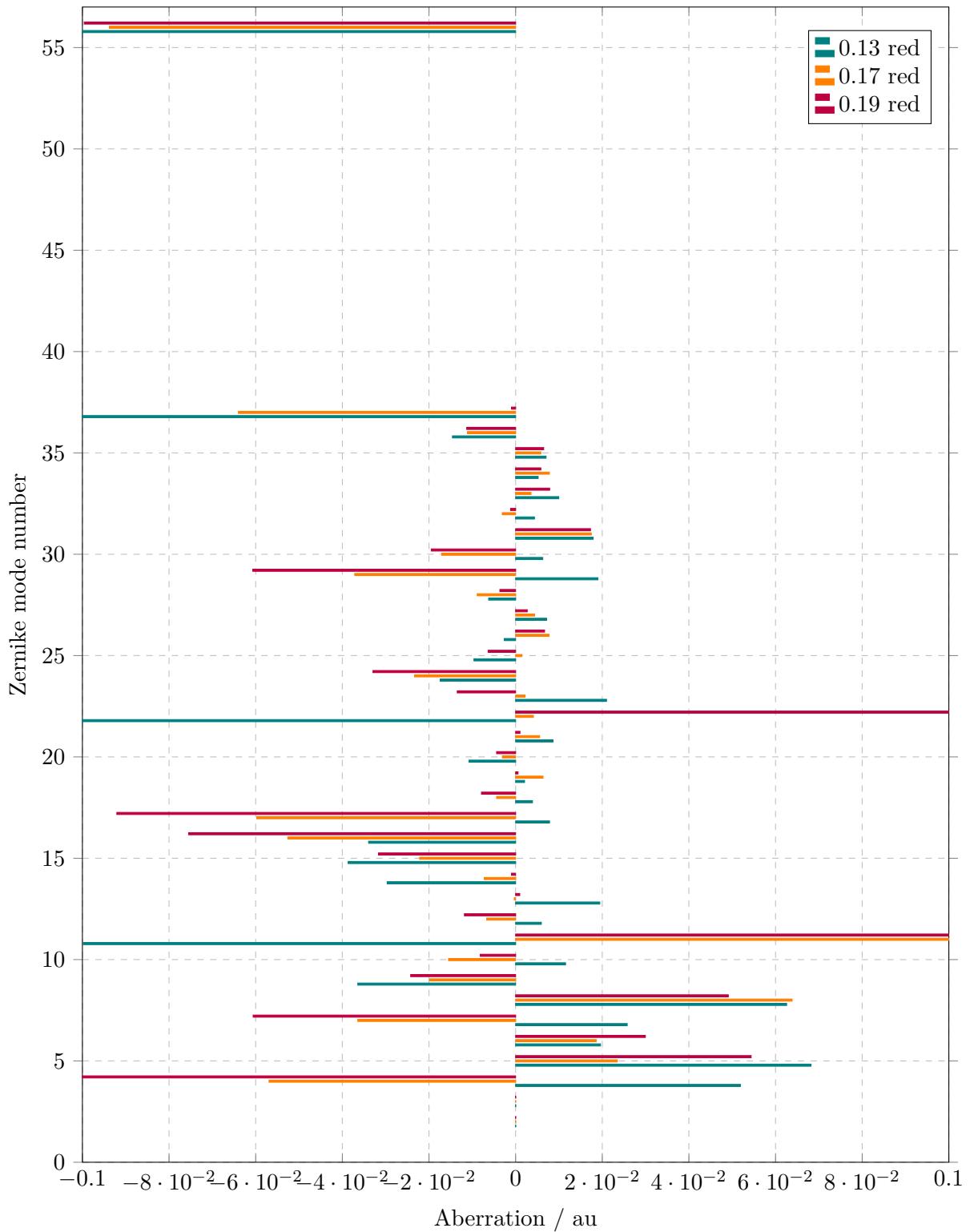


Figure 3.2: Red channel Zernike modes {1 ... 37, 56} versus aberrations of PSF model via phase retrieval, for all three correction collar settings (0.13, 0.17, 0.19).

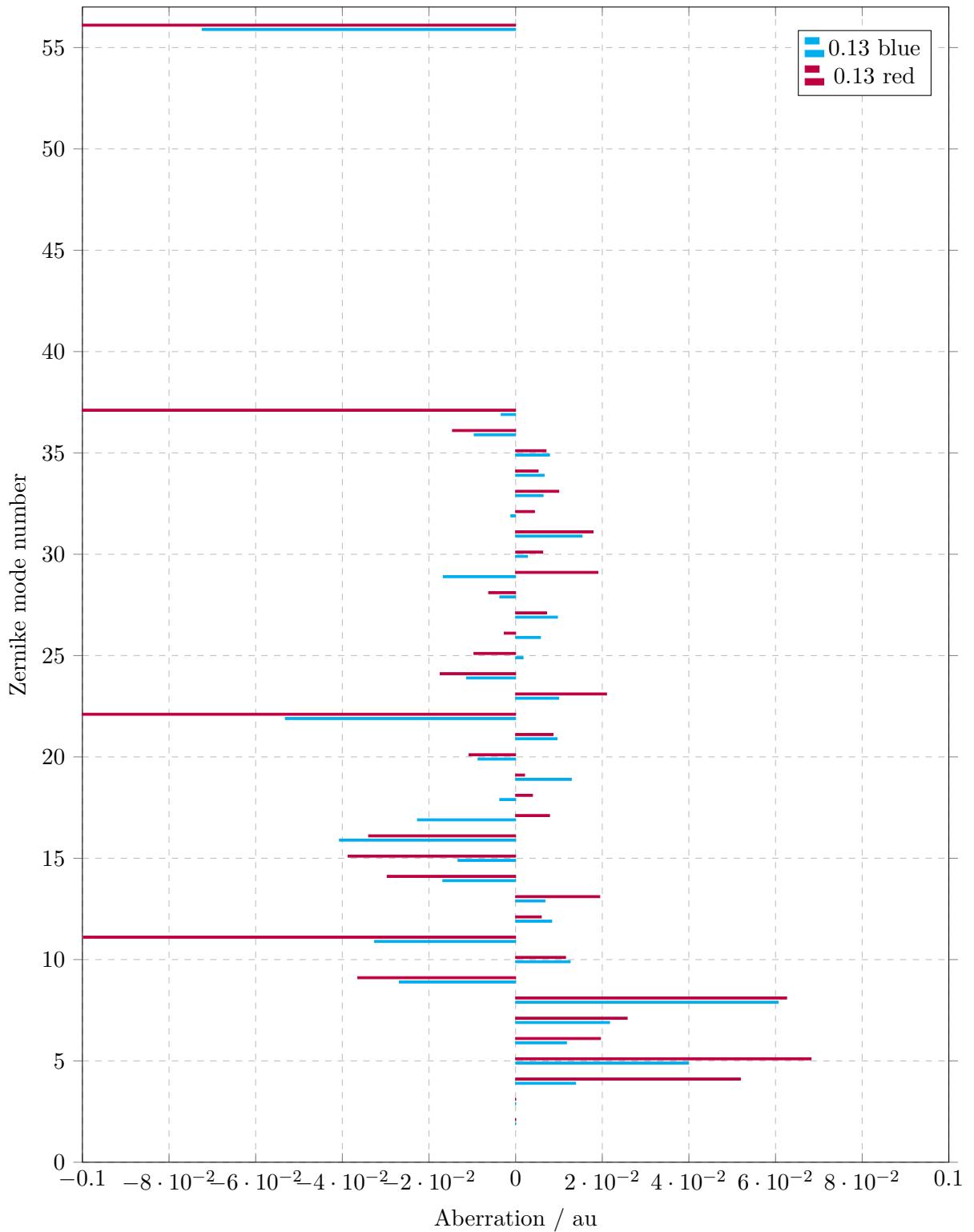


Figure 3.3: Red and blue channel Zernike modes {1 ... 37, 56} versus aberrations of PSF model via phase retrieval, for correction collar setting of 0.13.

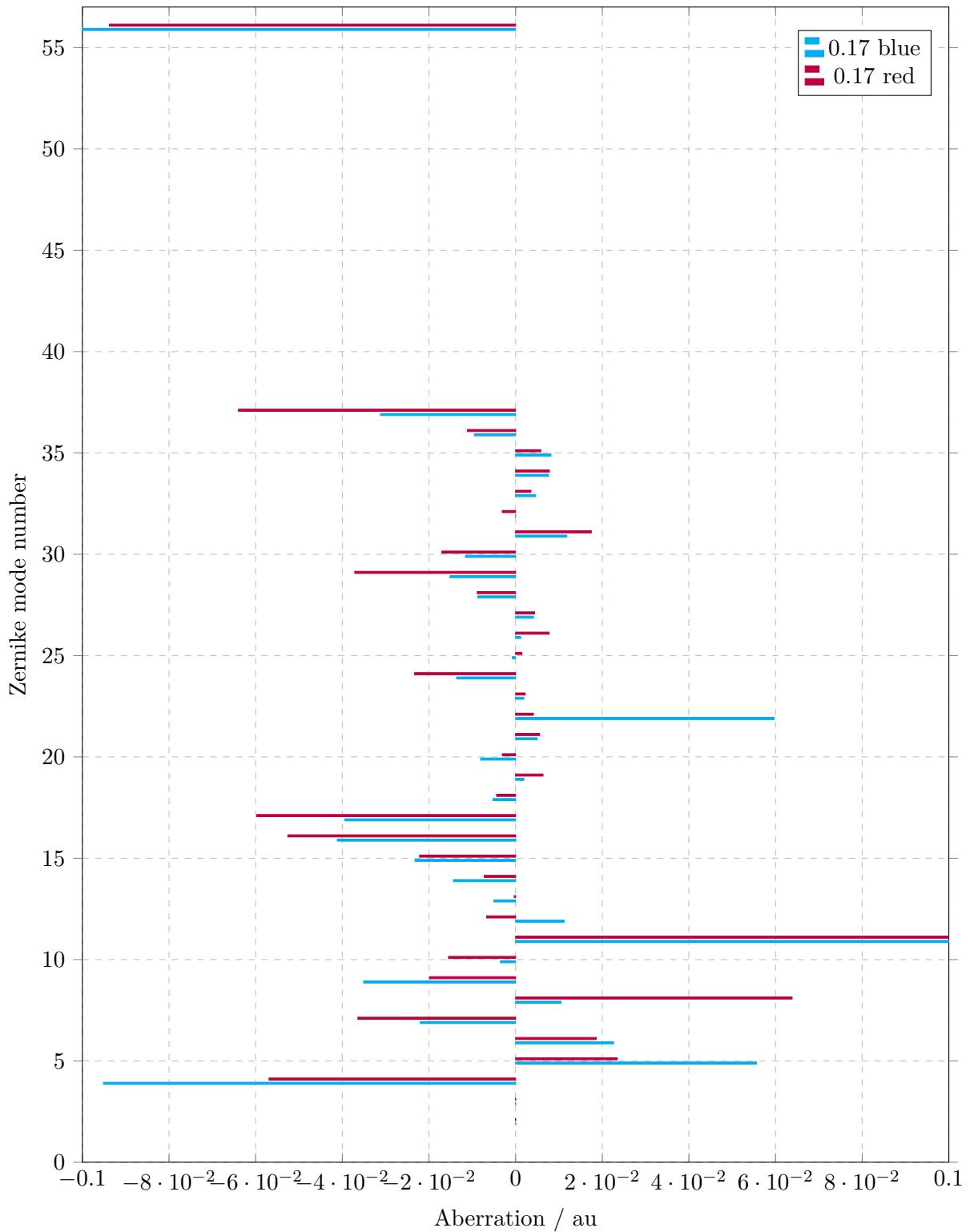


Figure 3.4: Red and blue channel Zernike modes {1 ... 37, 56} versus aberrations of PSF model via phase retrieval, for correction collar setting of 0.17.

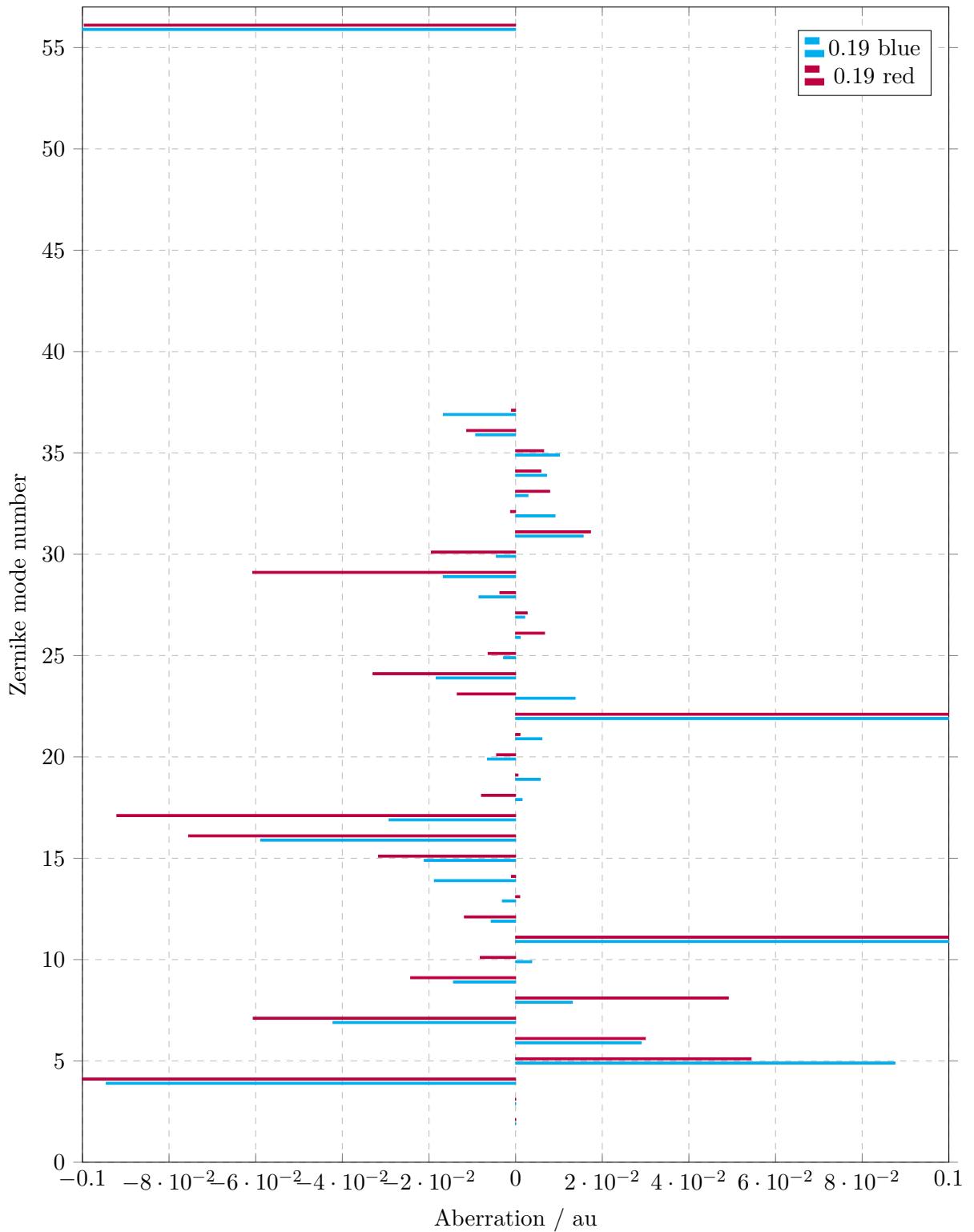


Figure 3.5: Red and blue channel Zernike modes $\{1 \dots 37, 56\}$ versus aberrations of PSF model via phase retrieval, for correction collar setting of 0.19.

3.1.3 Cramer Rao Lower Bound

The Estimated Cramer Rao Lower Bound for different correction Collar settings (varying 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.

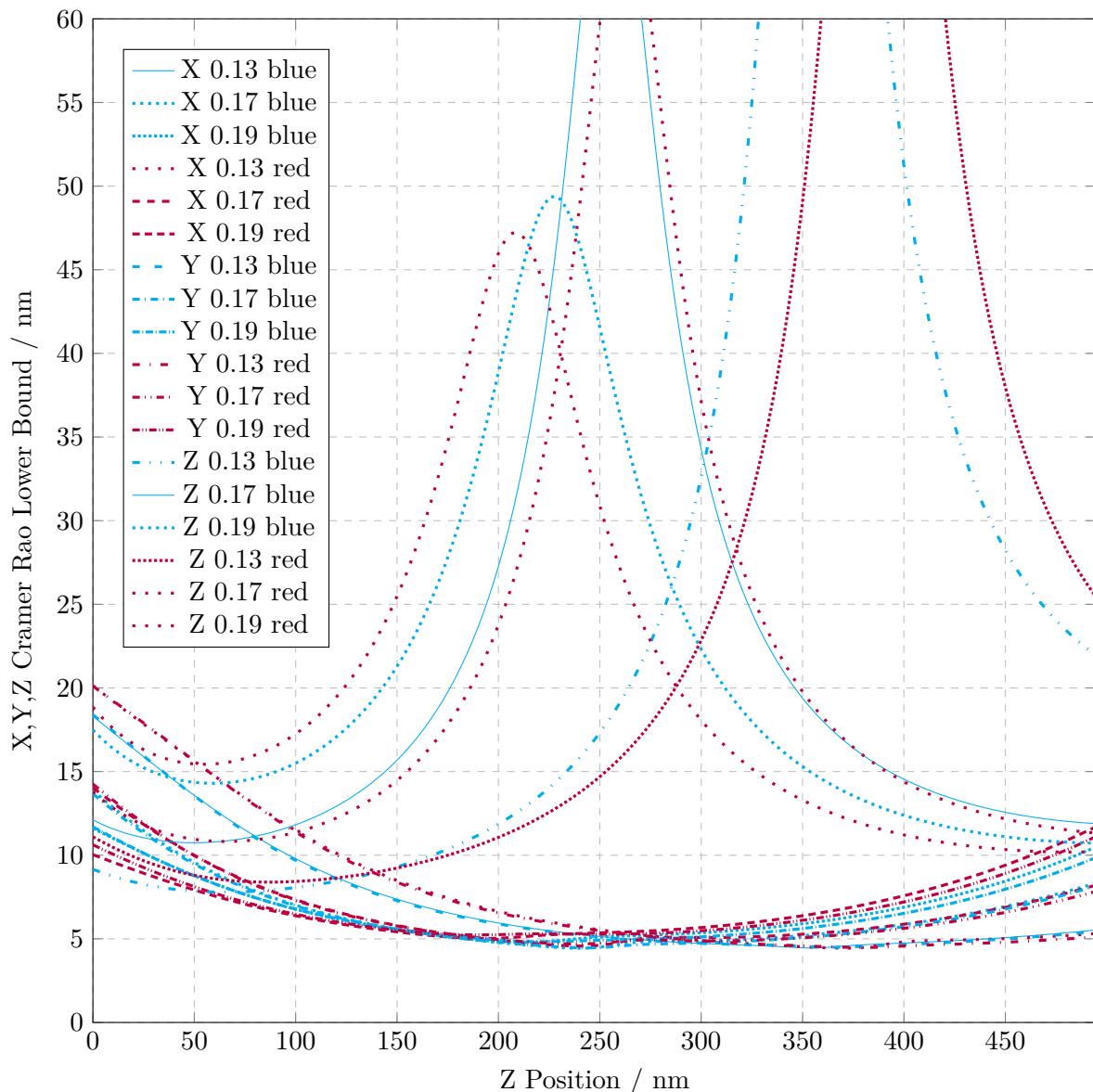


Figure 3.6: Estimated Cramer Rao Lower Bound for different correction Collar settings (varying linestyles for 0.13, 0.17, 0.19) of the Olympus 1.5 NA objective; for X, Y and Z axis (top, middle, and bottom); for both red and blue channel (colors).

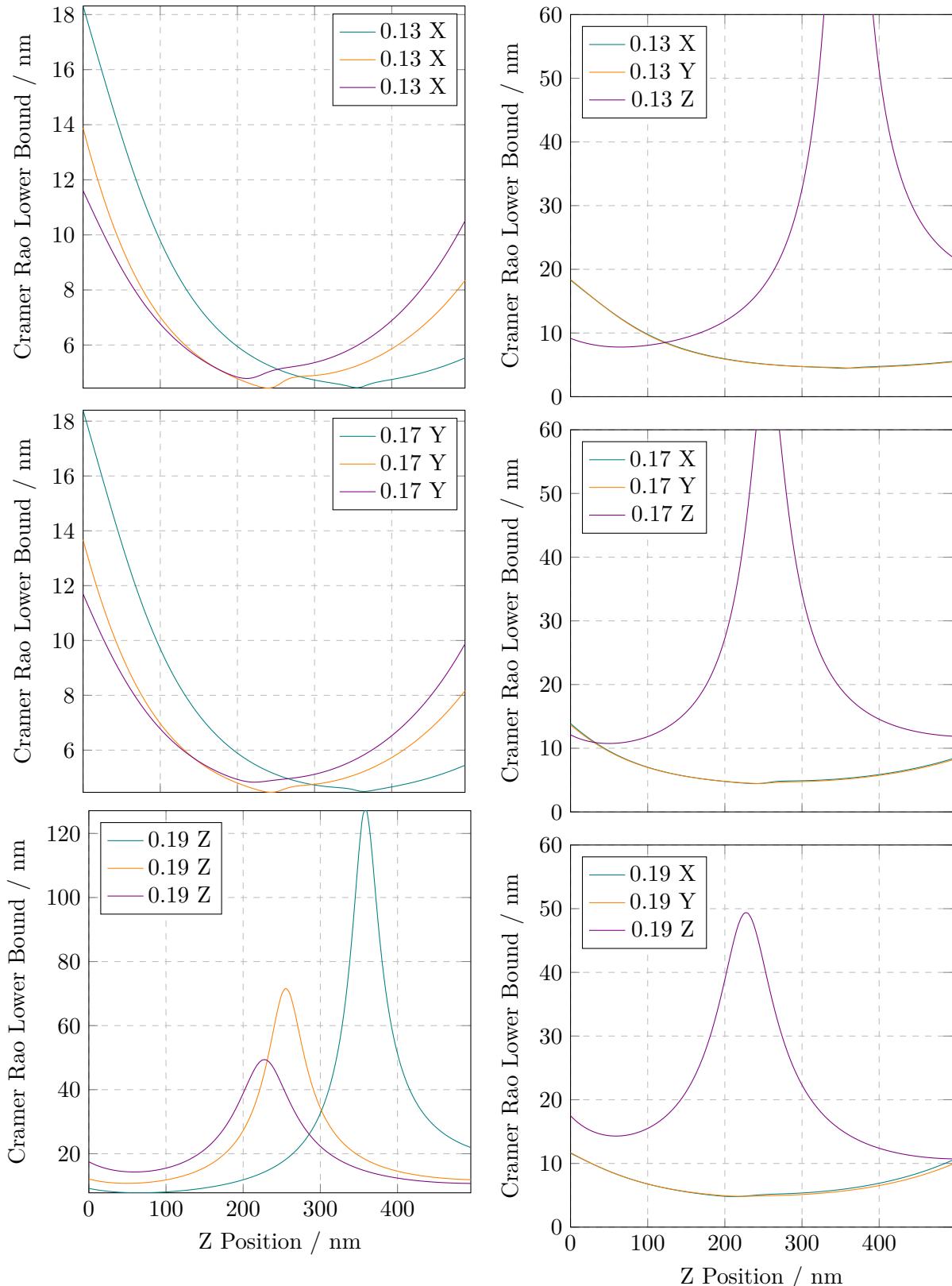


Figure 3.7: Estimated Cramer Rao Lower Bound for different correction Collar settings¹⁷ (teal: 0.13, orange: 0.17, purple: 0.19) of the Olympus 1.5 NA objective; grouped by X, Y and Z axis (top, middle, and bottom).

Figure 3.8: Estimated Cramer 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.

3.2 3d SMLM Analysis

Applying the SMLM described in Section [sec](#) for the stack of 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 [sec](#) 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.

[npc image](#)

3.2.1 Import

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

3.2.2 Drift Correction

The *drift* of the setup over time can be estimated using *ImageJ*, and is then imported as `drift`. In the following block the drift

correction is applied to all the 130k localisations, shown in Figure 3.9 as 3d plot of all the initial 370k drift corrected localisations.

drift corrected localizations: 370148

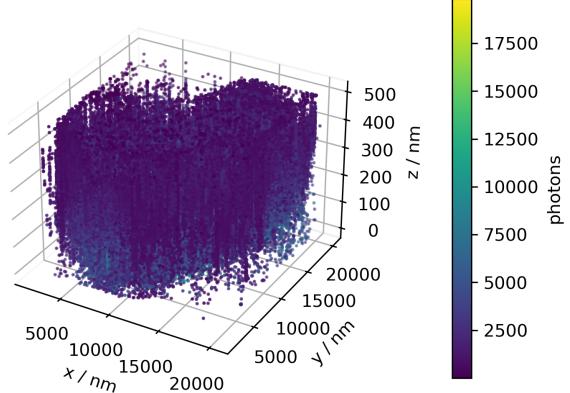


Figure 3.9: 3d plot of all 370148 drift corrected localisations, color coded by photon count.

3.2.3 Photon Counts

As a preliminary step it might prove useful to look at the photon count statistics, shown in Figure 3.10. 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.

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

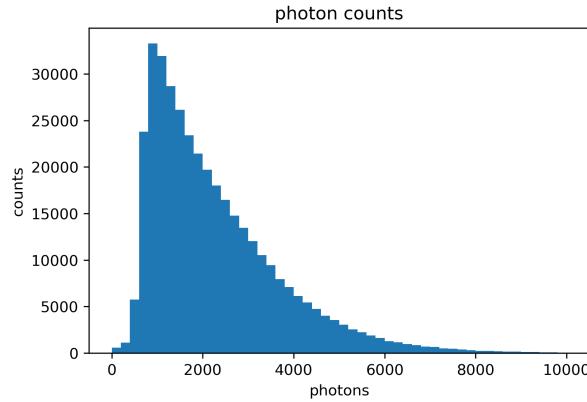


Figure 3.10: histogram of the photon count for the localisations.

Figure 3.11 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%.

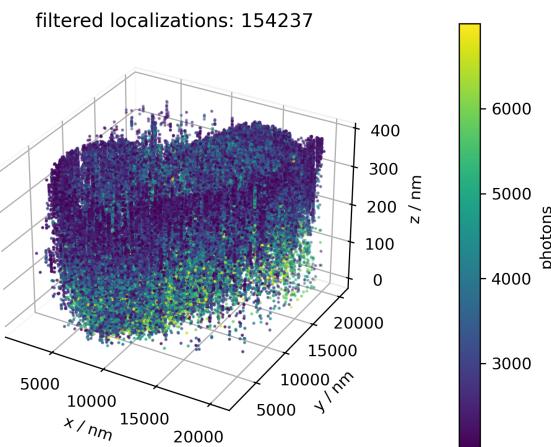


Figure 3.11: 3d plot of all 154237 filtered localisations, color coded by photon count.

3.2.5 Track Particles

The 50k frames we analyse here are taken with exposure of 20 ms, and 10 ms 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 ms (one frame) while some other is on for 60 ms; 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 3.12 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%.

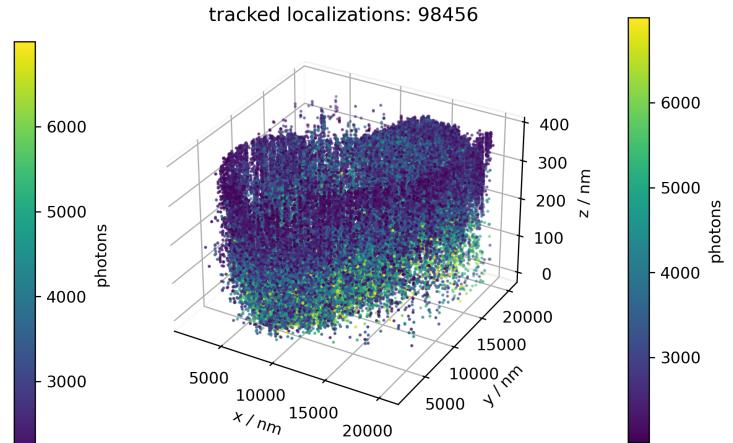


Figure 3.12: 3d plot of all 98456 filtered localisations, color coded by photon count.

3.3 3d SMLM Analysis: NPC

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 3.12 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%.

3.3.1 Clustering

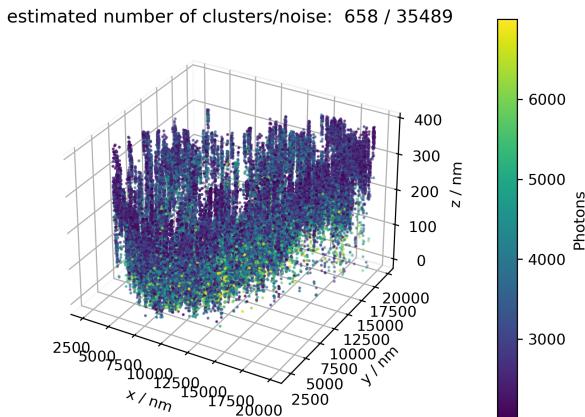


Figure 3.13: 3d plot of the localisations of the 658 identified clusters (omitting 35489 localisations as noise, due to not belonging to a cluster), color coded by photon count.

3.3.2 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 3.14 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.

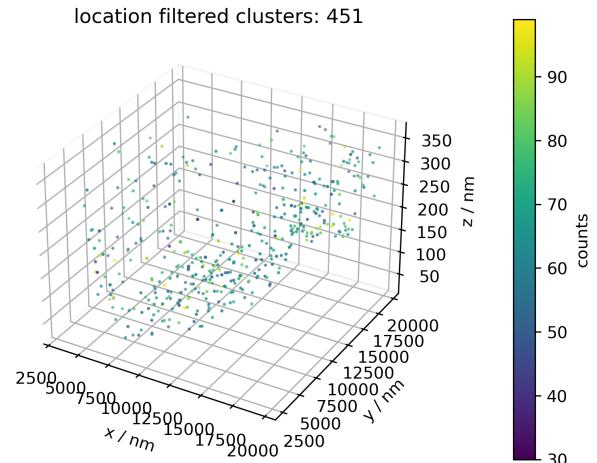


Figure 3.14: 3d plot of the positions of the 451 filtered clusters, color coded by photon count.

3.3.3 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—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 NPC's geometry.

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

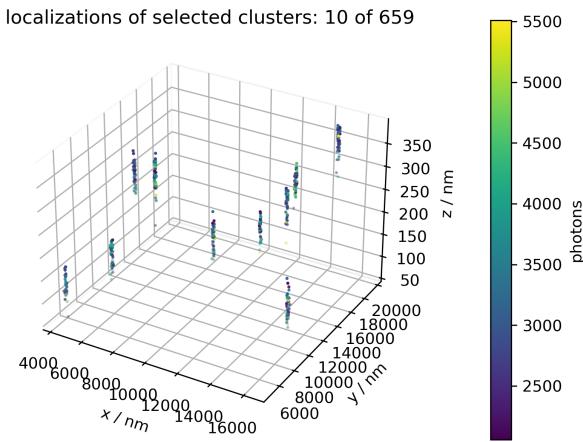


Figure 3.15: 3d plot of the localisations within the 10 best clusters, color coded by photon count.

3.3.4 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 3.16 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.

the localisations within the *best* cluster (top), the *worst* (bottom) and one in between.

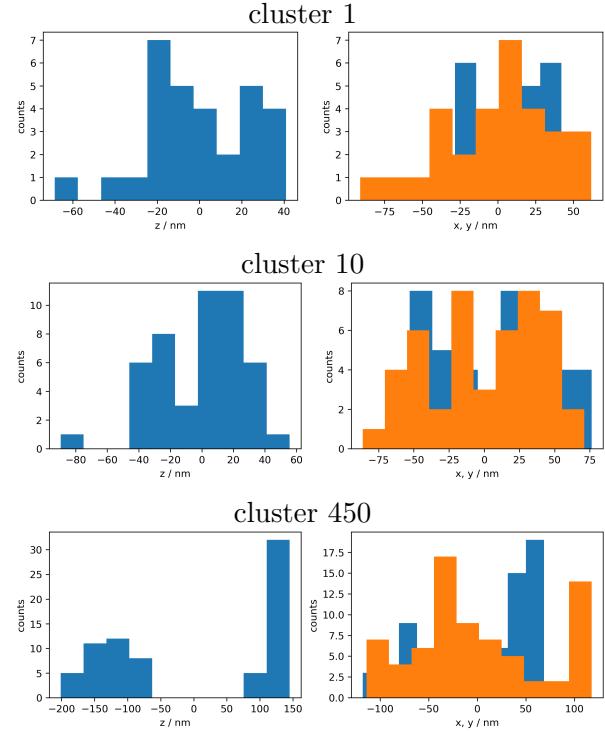


Figure 3.16: histogram of the `x,y` (right) respective `z` distribution (left) of the localisations within the best cluster (top), the worst (bottom) and one in between.

3.4 Dual Channel 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 3.4.1, as well as based on two dimensional SMLM data sets in Section 3.4.2.

3.4.1 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 3.17 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 3.17 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 3.17.

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 \mathbf{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:

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

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

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

t vec?

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

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

Rigid Projection

is:

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

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

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

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

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

t vec?

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

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

t vec?

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

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

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:

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

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

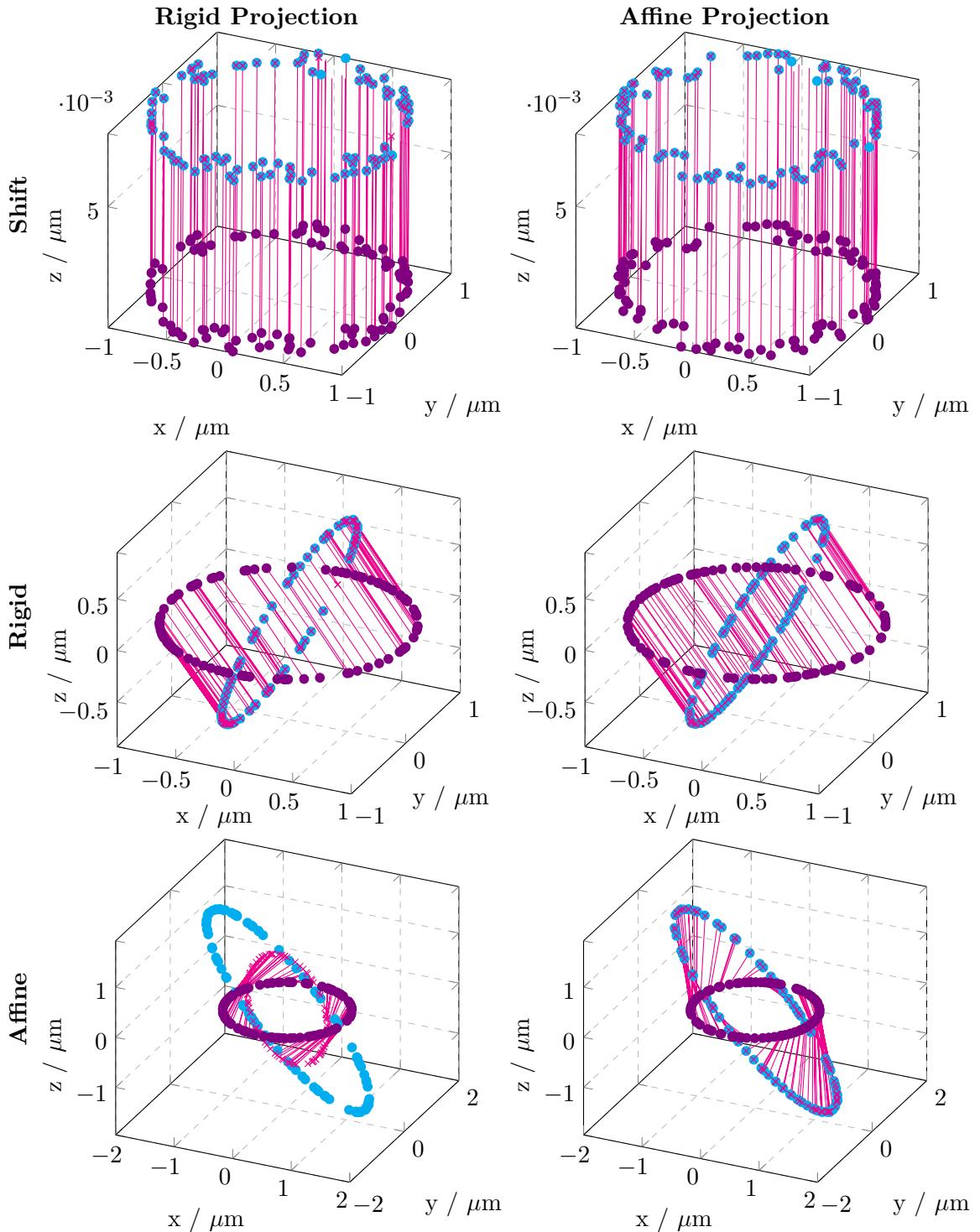


Figure 3.17: Demonstration of the recovery of rigid (left) respective affine (right) transformations, via recovering localisations of simulated two channel SMLM data; transformation (magenta) from red channel (violet) to blue channel (cyan); for the use cases of translation (top), rigid transformation (middle) and affine transformation (bottom). Obviously a rigid transformation may not correctly reconstruct an affine transformed data set (bottom left).

3.4.2 Projected 2d NPC

Lacking proper 3d dual channel 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 $\bar{\mathbf{R}}_r$ is almost unity within uncertainty. The mean translation vector $\bar{\mathbf{t}}_r$ shows a good variance over all the twelve analysed sets, with mean and standard derivation of:

$$\bar{\mathbf{t}}_r = \begin{pmatrix} 39 \\ 30 \\ 0 \end{pmatrix} \pm \begin{pmatrix} 10 \\ 4 \\ 0 \end{pmatrix} \mu\text{m} \quad (3.10)$$

$$\bar{\mathbf{R}}_r = \begin{pmatrix} 1.0000 & 0.0008 & 0 \\ -0.0008 & 1.000 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (3.11)$$

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

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 3.18, 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.

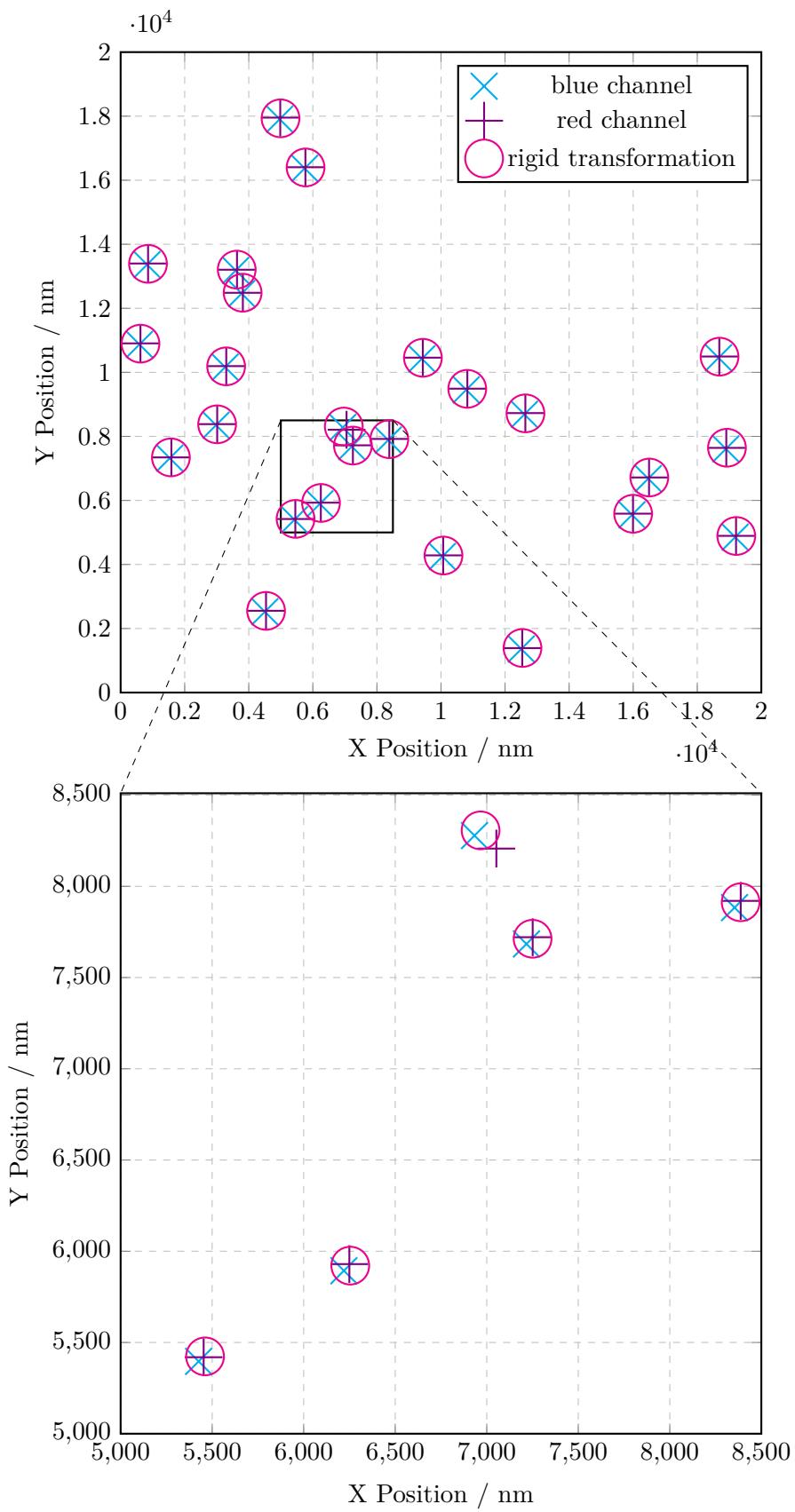


Figure 3.18: Demonstration of a rigid transformation of the localisations (magenta \circ) from blue channel (blue \times) to red channel (violet $+$); the transformed blue channel localisations mostly align well with the red channel localisations. 26

Chapter 4

Discussion

4.1 SMLM

4.2 Dual Channel Optics

4.2.1 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, as suspected by theory [zern](#).

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 $\{0.17, 0.19\}$.

4.2.2 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.

4.3 3d SMLM Analysis

The analysis performed in Section 3.2 is well suited to greatly reduce the localisations, just by omitting unphysical data and noise; in our example from initially 370k to below 100k, or to about 26%.

4.4 3d SMLM Analysis: NPC

4.4.1 Automation

The fully automatic evaluation of the NPC *quality*, shown in Section 3.3 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 3.3.4 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 vice versa. Unequal weighting based on empirical fine tuning might lessen this problem, but will most likely not solve it.

4.4.2 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 height, 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.

4.5 Dual Channel Projections

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.

4.5.1 SMLM Simulations

At first glance at the plots in Figure 3.17, 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 t and rotation matrices R in Section ?? enables a deeper under-

standing 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 Channel Optics

5.1.1 Zernike Modes

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 de-focus 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 dSTORM Buffer

5.3 3d SMLM Analysis

5.4 3d SMLM Analysis: NPC

5.5 Dual Channel 3d 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.

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

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

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

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

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 for the dual channel 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 channel

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

$$\bar{\mathbf{t}}_r = \begin{pmatrix} 39 \\ 30 \\ 0 \end{pmatrix} \pm \begin{pmatrix} 10 \\ 4 \\ 0 \end{pmatrix} \mu\text{m} \quad (5.1)$$

$$\bar{\mathbf{R}}_r = \begin{pmatrix} 1.0000 & 0.0008 & 0 \\ -0.0008 & 1.000 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (5.2)$$

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

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

```
1  #@markdown ##import core
2  import pandas as pd
3  import matplotlib.pyplot as plt
4  import numpy as np
5  import matplotlib as mpl
6  from mpl_toolkits.mplot3d import Axes3D
7  import scipy.io
8  from sklearn.cluster import DBSCAN
9  from sklearn import metrics
10 from sklearn.datasets import make_blobs
11 from sklearn.preprocessing import StandardScaler
12
13 #@markdown ##import jupyter
14 #matplotlib ipympl
15 #matplotlib widget
16 #matplotlib interactive
17 % matplotlib inline
18 import trackpy
19 #import sdt
20 #from sdt import io, chromatic, multicolor, brightness
21
22 ## local
23 wd = 'data/210422_npc_red_defocus/'
24 data = pd.read_csv( wd + 'cell1_tr1000_def500.csv',
25 header = None,
26 names=[ "x", "y", "z", "n", "bg", "fit", "id", "frame" ] )
```

A.1.2 Drift Correction

```
1  #@markdown ## import & scale drift
2  #@markdown > set **magnification** via factor in drift.
3
4  drift = pd.read_csv( wd + 'day2_cell1_driftValues.csv' )
5
6  drift['Y2']=drift['Y2']*146.6
7  drift['Y3']=drift['Y3']*146.6
8  drift['X2']=np.round(drift['X2'])
9  drift['X3']=np.round(drift['X3'])
10
11 #@markdown ## apply drift correction
```

```

12
13     for i in range(len(drift)-1):
14         fr=data[(data['frame']>=drift['X2'].iloc[i]) &
15             (data['frame']<drift['X2'].iloc[i+1])]
16         fr['y']=fr['y']-drift['Y2'].iloc[i]
17         fr['x']=fr['x']-drift['Y3'].iloc[i]
18         data[(data['frame']>=drift['X2'].iloc[i]) &
19             (data['frame']<drift['X2'].iloc[i+1])]=fr

```

A.1.3 Photon Counts

```

1     #@markdown ## Check: photon counts
2     #@markdown > set `max_photons` accordingly (default: 10000).
3     max_photons = 10000 #@param {type:"slider", min:0, max:40000, step:1000}
4
5     fig = plt.figure()
6     plt.hist( data['n'], bins=50, range=( 0, max_photons ) )

```

A.1.4 Filter

```

1     #@markdown ## filter
2     #@markdown > set `min_photons` and `max_photons` accordingly (default: 2000 < photons < 7000).
3     #@markdown > set `min_z` and `max_z` accordingly (default: 0 < z < 499). \
4     #@markdown > set `min_fit` accordingly (default: 6e6).
5     min_photons = 2000 #@param {type:"slider", min:0, max:40000, step:1000}
6     max_photons = 7000 #@param {type:"slider", min:0, max:40000, step:1000}
7     min_z = 0 #@param {type:"slider", min:0, max:500, step:1}
8     max_z = 384 #@param {type:"slider", min:0, max:500, step:1}
9     min_fit = 7192000 #@param {type:"slider", min:0, max:1e7, step:1000}
10
11    fdata = data[ ( data['n'] > min_photons ) &
12        ( data['n'] < max_photons ) &
13        ( data['z'] > min_z ) &
14        ( data['z'] < max_z ) &
15        ( data['fit'] < min_fit ) ]

```

A.1.5 Track Particles

```

1     #@markdown ## track all in x,y
2     #@markdown > set `sr` to wanted search range (default: 50). \
3     #@markdown > set `mem` to wanted memory (default: 10).
4     sr = 50 #@param {type:"slider", min:0, max:100, step:1}
5     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

```
1  #@markdown ## compute dbSCAN
2  #@markdown > set `dim = 2` for clustering in  $x$  and  $y$  (default:).\
3  #@markdown > set `dim = 3` for experimental clustering in 3d; be aware that  $x, z$  and  $y$  pre-
4  #@markdown > set `eps` (default: 200).\
5  #@markdown > set `min_samples` (default: 10).
6  dim = "2" #@param [2, 3]
7  eps = 100 #@param {type:"slider", min:0, max:500, step:10}
8  min_samples = 50 #@param {type:"slider", min:1, max:100, step:1}
9
10 alocalisations = localisations.to_numpy()
11 alocalisations[ :, 0:2 ]
12
13 db = DBSCAN( eps, min_samples ).fit( alocalisations[ :, 0:int( dim ) ] )
14 core_samples_mask = np.zeros_like( db.labels_, dtype=bool )
15 core_samples_mask[ db.core_sample_indices_ ] = True
16 labels = db.labels_
17 localisations[ "cluster" ] = labels
18
19 # count clusters (ignore noise if present)
20 n_clusters_ = len( set( labels ) ) - ( 1 if -1 in labels else 0 )
21 n_noise_ = list( labels ).count( -1 )
22
23 #print('Estimated number of clusters: %d' % n_clusters_)
24 #print('Estimated number of noise points: %d' % n_noise_)
25
26 nlocalisations = localisations.loc[ localisations['cluster'] == -1 ]
27 clocalisations = localisations.loc[ localisations['cluster'] != -1 ]
```

A.2.2 Cluster Analysis

```
1  #@markdown ## analyse clusters
2  #@markdown > set `npc_radius` to NPC radius /nm (default: 50).\
3  #@markdown > set `npc_height` to NPC height /nm (default: 150).
4  npc_radius = 50 #@param {type:"slider", min:0, max:500, step:1}
5  npc_height = 25 #@param {type:"slider", min:0, max:500, step:1}
6
7  clabels = set(labels)
8  cnames = [ "counts", "xmean", "ymean", "zmean", "nmean", "xvar", "yvar", "zvar",
9  "nvar", "label", "ringness", "twofold" ]
10 clusters = pd.DataFrame( index = clabels, columns = cnames, dtype="float64" )
```

```

11 clusters[ "label" ] = clabels
12
13 for k in clabels:
14     tmp = localisations.loc[ localisations['cluster'] == k ]
15     clusters.loc[ k, "counts" ] = len( tmp )
16     for label in [ "x", "y", "z", "n" ]:
17         clusters.loc[ k, label+"mean" ] = np.mean( tmp.loc[ :, label ] )
18         clusters.loc[ k, label+"var" ] = np.var( tmp.loc[ :, label ] )
19
20     ## xy: radius (distance to centroid)
21     rad = np.sqrt( ( tmp.loc[ :, "x" ] - clusters.loc[ k, "xmean" ] )**2 +
22     ( tmp.loc[ :, "y" ] - clusters.loc[ k, "ymean" ] )**2 )
23     ## xy: radius rms deviation from NPC radius
24     clusters.loc[ k, "ringness" ] = np.sqrt( sum( ( rad - npc_radius )**2 ) )
25
26     ## z: radius (distance to centroid)
27     rad = abs( tmp.loc[ :, "z" ] - clusters.loc[ k, "zmean" ] )
28     ## z: radius rms deviation from NPC radius
29     clusters.loc[ k, "twofold" ] = np.sqrt( sum( ( rad - npc_height )**2 ) )
30
31 #@markdown ## filter clusters
32 #@markdown > set `count_threshold` to min elements (counts) in cluster (default: 30)\n
33 #@markdown > set `diameter_threshold` to max x,y (radius) deviation of cluster from NPC\n
34 #@markdown > set `twofold_threshold` to max z (radius) deviation of cluster from NPC height\n
35 #@markdown > set `xyvar_threshold` to wanted x,y variance (default: 1e4)\n
36 #@markdown > set `zvar_threshold` to wanted z variance (default: 1e4)
37 count_threshold = 100 #@param {type:"slider", min:0, max:200, step:1}
38 diameter_threshold = 500 #@param {type:"slider", min:0, max:500, step:10}
39 twofold_threshold = 1000 #@param {type:"slider", min:0, max:1000, step:10}
40 xyvar_threshold = 100000 #@param {type:"slider", min:0, max:1e5, step:1e3}
41 zvar_threshold = 100000 #@param {type:"slider", min:0, max:1e5, step:1e3}
42
43 fclusters = clusters[ ( clusters['counts'] < count_threshold ) &
44     ( clusters['ringness'] < diameter_threshold ) &
45     ( clusters['twofold'] < twofold_threshold ) &
46     ( clusters['xvar'] < xyvar_threshold ) &
47     ( clusters['yvar'] < xyvar_threshold ) &
48     ( clusters['zvar'] < zvar_threshold ) ]

```

A.2.3 Select Clusters

```

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

```

```

3  #@markdown > set `sort` to sort the best clusters (default: ringness + twofold).
4  show_clusters = 10 #@param {type:"slider", min:0, max:1000, step:1}
5  sort = 'ringness + twofold' #@param [ "ringness + twofold", "twofold", "ringness", "xyvar"
6
7  if sort == "xvar": # sort by variance
8      sclusters = fclusters.sort_values( "xvar" )
9  elif sort == "xyvar": # sort by x and y variance using least squares
10     sclusters = fclusters.loc[
11         ( fclusters.xvar ** 2 + fclusters.yvar ** 2 ).sort_values().index ]
12  elif sort == "ringness": # sort by ringness (deviation to ringness)
13     sclusters = fclusters.sort_values( "ringness" )
14  elif sort == "twofold": # sort by ringness (deviation to ringness)
15     sclusters = fclusters.sort_values( "twofold" )
16  elif sort == "ringness + twofold": # sort by ringness (deviation to ringness)
17     sclusters = fclusters.loc[
18         ( fclusters.twofold ** 2 + fclusters.ringness ** 2 ).sort_values().index ]
19
20  show_clusters = min( show_clusters, len( sclusters ) )
21  selected_clusters = sclusters["label"].iloc[ 0:show_clusters ]
22
23  fig = plt.figure()
24  ax = fig.add_subplot( projection = '3d' )
25  for clus in selected_clusters:
26      flocalisations = localisations[ ( localisations['cluster'] == clus ) ]
27      ff = ax.scatter( flocalisations['x'],
28                      flocalisations['y'],
29                      flocalisations['z'],
30                      s=1 ,c = flocalisations['n'] )

```

A.2.4 X,Y,Z Histograms

```

1  #@markdown ## Check: z distribution
2  #@markdown > set `plot_cluster` to wanted cluster (default: 0).
3  plot_cluster = 3 #@param {type:"slider", min:0, max:100, step:1}
4  plot_cluster = min( plot_cluster, len( sclusters ) -1 )
5
6  tmp = localisations.loc[ localisations['cluster'] ==
7      sclusters.loc[ sclusters.index[ plot_cluster ],
8      "label" ] ]
9
10 z = tmp.z - np.mean( tmp.z )
11
12 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 Channel 3d SMLM

A.3.1 Cockpit: cockpit.m

```

41 addpath( hd ) # function & stuff needed
42 stamp = strftime("%Y_%m_%d_%H%M%S", localtime (time ())); # create timestamp for saving file
43 global cwd = sprintf( "%s/%s/all_%s%s%s", hd, wd, method, stamp, nn ); # concat working dir
44 mkdir( cwd )
45 chdir( cwd )
46 global rf = fopen ( "cockpit.log", "w" );
47
48 disp( 'cycle ' );
49 rotations = nan( nmax, 3, 3 );
50 translations = nan( nmax, 3 );
51 for n = 1 : nmax
52     if ( any( exclude == n ) )
53         disp( sprintf( "      warning: skipping file positions%d.csv", n ) );
54         continue
55     endif
56     disp( sprintf( "      importing positions%d.csv", n ) );
57     disp( '      load ' );
58
59 ## "id", "frame", "x [nm]", "y [nm]", "sigma [nm]", "intensity [photon]", "offset [photon]", "bk
60 try
61     pos = csvread( sprintf( "%s/%s/%s%d.csv", hd, wd, fn, n ) );
62 catch
63     continue
64 end_try_catch
65
66
67 disp( '      analyse ' );
68
69 [ centroid_red, centroid_blue_transformed, centroid_blue, red, blue, rotation, translation
70
71 ## save optimal rigid transform for comparison
72 rotations( n, :, : ) = rotation;
73 translations( n, : ) = translation;
74 endfor

```

A.3.2 Splice Channels: ampel.m

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

```

7  ## cluster. Depending on "method" either finds best rigid transform, or
8  ## best affine transform correlating the red and blue sets via least
9  ## squares (singular-value-decomposition).
10 ## @ moritz siegel
11
12 global nwd
13 global rf
14 global minpts
15 global dist
16
17 ## sort red/blue/empty of many frames.
18 red = blue = empty = zeros( size( pos ) );
19 nb = nr = ne = 1;
20 for k = 1 : size( pos, 1 )
21
22 ## exclude blank lines (text).
23 if ( all( pos( k, : ) == 0 ) )
24     disp( sprintf( 'warning: line %d is empty; skipping.', k ) );
25     fprintf( rf, 'warning: empty line; skipping.\n' );
26     continue
27 endif
28
29 ## red pill / blue pill?
30 channel = mod( pos( k, 2 ), 3 );
31 switch ( channel )
32     case channel_order(1)
33         blue( nb, : ) = pos( k, : );
34         nb = nb + 1;
35     case channel_order(2)
36         empty( ne, : ) = pos( k, : );
37         ne = ne + 1;
38     case channel_order(3)
39         red( nr, : ) = pos( k, : );
40         nr = nr + 1;
41 endswitch
42 endfor
43 disp( sprintf( 'sorted localisations:\n %d red\n %d blue\n %d empty', nr, nb, ne ) );
44 fprintf( rf, 'sorted localisations:\n %d red\n %d blue\n %d empty\n', nr, nb, ne );
45
46 ## analyse clusters & average centroids.
47 [ assignments_blue, c_blue ] = dbscan( blue( 1:300, 3:4 ), minpts, dist );
48 for k = 1 : c_blue
49     idx = find( assignments_blue == k );
50     centroid_blue( k, 1:2 ) = mean( blue( idx, 3:4 ), 1 );

```

```

51    endfor
52    [ assignments_red, c_red ] = dbscan( red( 1:300, 3:4 ), minpts, dist );
53    for k = 1 : c_red
54        idx = find( assignments_red == k );
55        centroid_red( k, 1:2 ) = mean( red( idx, 3:4 ), 1 );
56    endfor
57
58    ## need to ditch clusters if the sets are not equally sized.
59    assert( c_red == c_blue, "clusters are not equally sized" );
60
61    ## recover transform.
62    switch( method )
63        case "rigid"
64            ## we only have 2d data currently, introduce linear dependent z component.
65            centroid_red = [ centroid_red, zeros( c_red, 1 ) ]
66            centroid_blue = [ centroid_blue, zeros( c_blue, 1 ) ]
67            [ rotation, translation ] = rig( centroid_blue, centroid_red )
68        case "affine"
69            ## we only have 2d data currently, introduce noise for z component,
70            ## matrix cant be singular, else cholesky decomposition fails.
71            z = 1 + 1e-3 * rand( c_red, 1 );
72            centroid_red = [ centroid_red, z ];
73            centroid_blue = [ centroid_blue, z ];
74            [ rotation, translation ] = affine( centroid_blue, centroid_red );
75    endswitch
76
77    ## transform blue set to red set.
78    centroid_blue_transformed = ( rotation * centroid_blue' )' + translation;
79
80 endfunction

```

A.3.3 Rigid Transformation: rig.m

```

1  #!/bin/octave
2  function [ rotation, translation, s ] = rig( p, q )
3      ## this function "rig(p,q)" finds the optimal rigid 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 the rotation matrtix "rotation", and the
7      ## 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 ## moritz siegel @ 210322
15
16 ## check stuff.
17 assert( nARGIN == 2 && size( p ) == size( q ), ...
18 "need 2 identical input matrices\n" );
19 assert( size( p, 2 ) == 3, "input matrix p must be nx3\n" );
20 assert( size( q, 2 ) == 3, "input matrix q must be nx3\n" );
21 n = size( p, 1 );
22 assert( n > 2, "need at least 3 points\n" );
23
24 ## 1) center to get rid of translation.
25 centroid_p = mean( p, 1 );
26 centroid_q = mean( q, 1 );
27 p_shifted = p - repmat( centroid_p, n, 1 );
28 q_shifted = q - repmat( centroid_q, n, 1 );
29
30 ## 2) solve least squares problem for best rotation.
31 ## -----
32
33 ## covariance matrix.
34 h = p_shifted' * q_shifted;
35
36 ## singular value decomposition.
37 [ u, s, v ] = svd( h );
38 rotation = v * u'
39
40 ## reflection? theres more to that.
41 if ( det( rotation ) < 0 )
42 printf( "warning: det(r) < 0\n" );
43 if ( any( s( : ) ) < 0 )
44 printf( "found reflection, correcting.\n" );
45 v( :, 3 ) = -v( :, 3 );
46 rotation = v * u';
47 else
48 printf( "error: single-value-decomposition failed! \
49 provided data seems is too noisy for least-squares\n." );
50 endif
51 endif
52
53 ## 3) compute translation.
54 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 matrix "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 repository 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://citesearx.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;

```

```

21 isneighbour = [];
22 neighbourcount = 0;
23 for j=1: size(X)(1)
24     dist = sqrt(sum((X(i,:)-X(j,:)).^2));
25     if(dist<EPS)
26         neighbourcount++;
27         isneighbour = [isneighbour j];
28     endif
29 endfor
30 if(neighbourcount<minpts)
31     continue;
32 else
33     C++;
34     assignments(i) = C;
35     for k=isneighbour
36         if(clustered(k)==0)
37             clustered(k) = 1;
38             _isneighbour = [];
39             _neighbourcount = 0;
40             for j=1: size(X)(1)
41                 dist = sqrt(sum((X(k,:)-X(j,:)).^2));
42                 if(dist<EPS)
43                     _neighbourcount++;
44                     _isneighbour = [_isneighbour j];
45                 endif
46             endfor
47             if(_neighbourcount>=minpts)
48                 isneighbour = [isneighbour _isneighbour];
49             endif
50         endif
51         assignments(k) = C;
52     endfor
53     endif
54 endfor
55 endfunction

```

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

```

1 #!/bin/octave
2 ## @ moritz siegel
3
4 clear all
5 close all

```

```

6 clc
7 graphics_toolkit('gnuplot')
8 #graphics_toolkit('qt')
9 #graphics_toolkit('fltk')

10
11 ## /////////////////////////////////
12 ## soft settings /////////////////////////////////
13 global hd = "~/biophysics"; # parent directory
14 global wd = "data";
15 global nn = "AFFINE"; # additional luminosity suffix ( default: "" )
16 method = "affine"; # /char, "rigid", "affine", method to reconstruct
17 #method = "affine"; # /char, "rigid", "affine", method to reconstruct
18 eps = 1e-8; # precision ( default: 1e-8 )
19 ## end of settings: hands off /////////////////////////////////
20 ## /////////////////////////////////
21
22
23 disp( 'init /////////////////////////////////' );
24
25 ## lets move it move it.
26 addpath( hd ) # function & stuff needed
27 stamp = strftime("%Y_%m_%d_%H%M%S", localtime (time ())); # create timestamp for saving file
28 global nwd = sprintf( "%s/%s/simulation_%s_%s_%s", hd, wd, method, stamp, nn ); # concat w/
29 mkdir( nwd )
30 chdir( nwd )

31
32 ## init first point cloud. introduce noise for z component, matrix cant
33 ## be singular (e.g. points on a plane), else cholesky decomposition fails.
34 n = 100;
35 theta = 100 * rand( n, 1 );
36 p = [ cos( theta ), sin( theta ), 1e-3 * rand( size( theta ) ) ];
37
38 ## introduce noise
39 salt = [ 1e-2*rand( size( theta ) ), 1e-2*rand( size( theta ) ), 1e-5*rand( size( theta ) ) ]
40 q = p + salt;

41
42 ## define affine transforms & derive second point cloud.
43 simulated_translation = [ 0, 0.005, 0.003 ];
44 reflect = [ -1, 0, 0; 0, 1, 0; 0, 0, 1 ];
45 scale = [ 2, 0, 0; 0, 1, 0; 0, 0, 1 ];
46 theta = 70;
47 rot = [ cosd(theta), 0, -sind(theta); 0, 1, 0; sind(theta), 0, cosd(theta) ];
48 shear = [ 1, 0.5, 0; 0, 1, 0; 0, 0, 1 ];
49 simulated_transform = eye( 3 );

```


List of Tables

2.1	Ingredients used for preparation of OxEA buffer for dual channel dSTORM buffer.	6
5.1	Quality of the recovered rotation translation vectors (transformation) matrices $\mathbf{t}_{s,r,a}^{r,a}$	30
5.2	Quality of the recovered rotation (transformation) matrices $\mathbf{R}_{s,r,a}^{r,a}$	30

List of Figures

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)	10
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)	11
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.	12
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.	13
3.5	Red and blue channel Zernike modes $\{1 \dots 37, 56\}$ versus aberrations of PSF model via phase retrieval, for correction collar setting of 0.19.	14
3.6	Estimated Cramer Rao Lower Bound for different correction Collar settings (varying 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).	16
3.7	Estimated Cramer 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).	17
3.8	Estimated Cramer 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.	17
3.9	3d plot of all 370148 drift corrected localisations, color coded by photon count. .	18
3.10	histogram of the photon count for the localisations.	19
3.11	3d plot of all 154237 filtered localisations, color coded by photon count.	19
3.12	3d plot of all 98456 filtered localisations, color coded by photon count.	19
3.13	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. .	20
3.14	3d plot of the positions of the 451 filtered clusters, color coded by photon count. .	20
3.15	3d plot of the localisations within the 10 <i>best</i> clusters, color coded by photon count.	21
3.16	histogram of the x,y (right) respective z distribution (left) of the localisations within the <i>best</i> cluster (top), the <i>worst</i> (bottom) and one in between.	21

3.17 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).	24
3.18 Demonstration of a rigid transformation of the localisations (magenta \circ) from blue channel (blue \times) to red channel (violet $+$); the transformed blue channel localisations mostly align well with the red channel localisations.	26

Acronyms

CRLB cramer rao lower bound. 6

dSTORM stochastic optical reconstruction microscopy. 4–7, 51

MEA β -mercaptopropylamine hydrochloride. 5, 6

MLE maximum-likelihood estimation. 5

NPC nuclear pore complex. 1, 2, 7, 18–22, 25, 27, 28

PBS phosphate-buffered saline. 5, 6

SMLM single molecule localisation microscopy. 2, 4–7, 18, 19, 22, 24, 25, 27, 30, 31, 53