Using DNA-PAINT and the Picasso Software to image Carbon Nanotube Structures

Marius Wiesner
University of Osnabrück

August 26, 2024

Contents

1 Document structure and objectives

This document has been prepared with two main objectives in mind.

Firstly, it is supposed to serve as an introductory summary of the DNA-PAINT Method and the usage of this method to image nanoscale structures.
In section 2 the document therefore starts by explaining the general principle behind a super-resolution imaging technique called "PAINT (=Point Accumulation for Imaging in Nanoscale Topography)". It then focuses on a specific PAINT method, namely "DNA-PAINT", which uses DNA strands to implement the PAINT principle. In particular, this method was used to image Single Walled Carbon Nanotubes (SWCNTs).

Secondly, this document shows how one can make use of the software environment package "Picasso" to genarate images from camera image data allocated via the DNA-PAINT method and perform data analysis on that data in section 3. Hence, after a short presentation on the structure of the Picasso software package, a more detailed look on its crucial components is given. At last, this document shows how it is possible to perform data analysis using the python programming language (although in principle other softwares can be used) on camera data aquired by the DNA-PAINT method in section 4. This entails preprocessing raw camera image data to a usable format which can then be imported into a desired python environment using the Picasso python library.

For the data analysis part this document is accompinied by two Jupyter Notebook files[1] which contain necessary code for exporting/importing data and performing data analysis. The data analysis part focuses heavily on image data of SWCNT structures.

[1] Wiesner 2024a; Wiesner 2024b

2 The PAINT Method and DNA-PAINT

The acronym PAINT stands for "Point Accumulation for Imaging in Nanoscale Topography" and describes a microscopy super-resolution technique for imaging nanoscale structures. It is localization based, meaning that instead of observing the whole structure that is supposed to be imaged at once it images individual points of the structure over multiple camera frames.

This is archieved with the use of fluorescent molecules by controlling them in such a way that only a subset of them are detectable by the camera device at any given time, i.e. within a single camera frame. That way their spacial coordinates can be localized with subdiffraction precission.

DAN-PAINT implements this principle by binding fluorescent molecules to single stranded DNA-strands called "imager strands". These imager strands can diffuse freely within an imaging solution around the structure that is supposed to be imaged. The imaging structure itself is functionalized with DNA strands referred to as "docking strands". These docking strands are complementary to the imager strands, meaning that they can transiently bind with each other. If you want more information on the process of functionalizing Carbon Nano Tubes with different molecules check out (G. Manoharan et al. 2023) and (Gririraj Manoharan 2021). If you want to know how to capture images with the DNA-PAINT method yourself in an experiment, a detailed explanation is given in (Schnitzbauer et al. 2017b).

The basic idea for imaging the structure is that for an imager strand to be detected by the camera device it needs to be attached to a docking strand.
Freely diffusing imager strands are not detected by the camera since they diffuse over many pixels within a single frame. Temporarily binded imager strands are fixed at the same place for the binding duration, allowing the camera to collect enough photons to detect a blinking spot[2]. Figure 1 illustrates this idea.

Notice that Δt_B refers to the binding duration of a DNA strand pair. Accordingly, $\frac{1}{f_B}$ represents the refraction period T_B with f_B being the binding frequency, i.e. the number of bindings that occur at an individual docking strand per unit of time.

The binding duration Δt_B depends on the stability of the DNA duplex and can be influenced by things like strand lenght, temperature, etc. The binding frequency f_B depends on the influx rate of imager strands to a docking strand and can be controlled by adjusting the concentration of imager strands.

[2]I will often refer to these blinking spots as a "localization" since such a detected blinking spot represents the localization of a docking strand attached to the structure. The Picasso software refers to these spots as localizations as well.

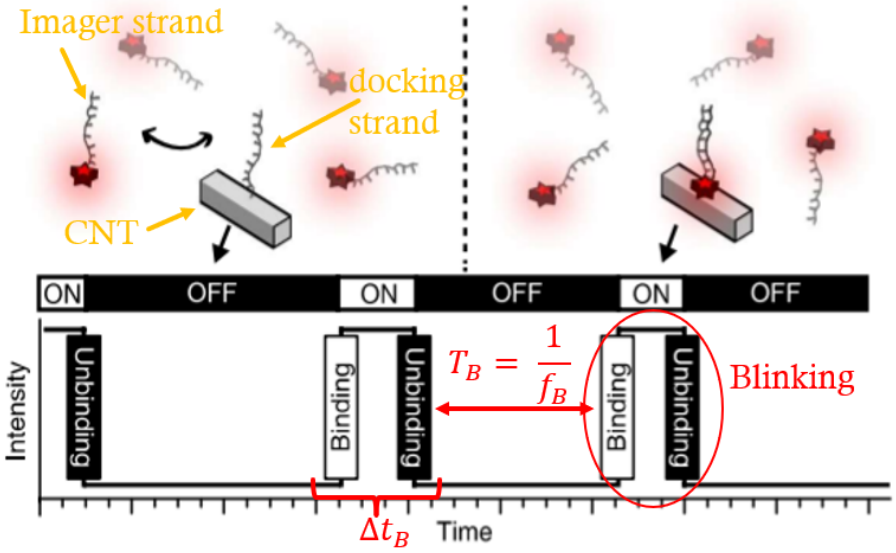


Figure 1: Transient binding of complementary DNA strands creating detectable blinking spots (= localizations).
Source: Adapted from Schnitzbauer et al. 2017b, p. 1199 and edited by the author.

One of the big advantages of this technique is the avoidance of unwanted Photobleaching effects.[3] This is due to the huge supply of imager strands for an individual docking strand. If one imager strand's conjugated fluorescent dye is affected by photobleaching it has practically no effect for the overall spot detection since there are many more imager strands the docking strand is coupling with over the duration of the experiment. An example of a DNA-PAINT image can be seen in Figure 2.

One might think that effects that prevent the imager strands from diffusing freely, such as binding of imager strands to the coversplip, impact the quality of the image. However, as shown in (Gririraj Manoharan 2021, p. 107 f.) these effects are negligible. It is also relevant to note that flourescent quenching has been shown in (Swathi and Sebastian 2010) to be distant dependent, i.e. dependent on the length of the DNA strands.[4].

[3]Photobleaching is a series of different photochemical reactions in which light-induced damage or modification of the fluorophore occurs, weakining the flourescent capabilities of the fluorophores.

[4]The term 'Quenching' refers to phenomena that weaken the flourescent intensity of fluorophores without destroying them, i.e. without bleaching. In this context this is due to energy transfer of fluorophores to the Nanotubes. See (Swathi and Sebastian 2010) for more information on this.

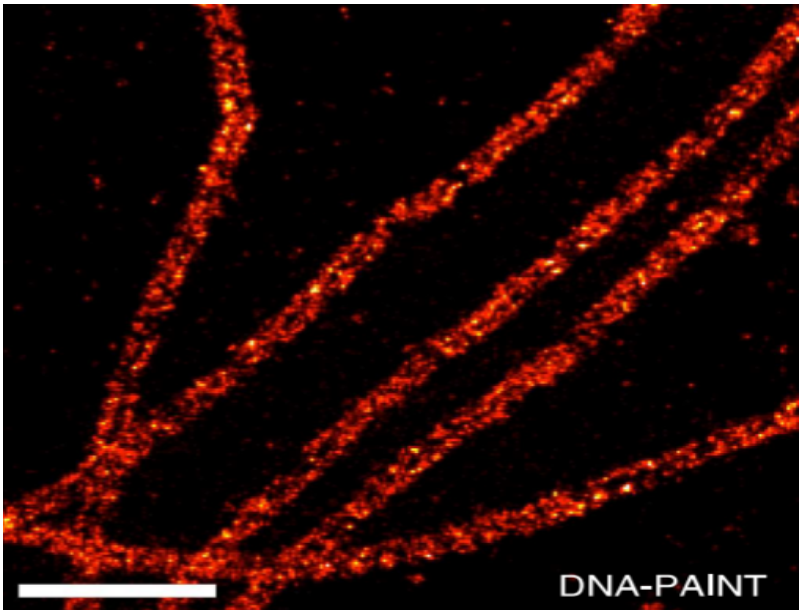


Figure 2: DNA-PAINT image of the amino acid chain α-tubuline (Scale bar: 500nm).
Source: Schnitzbauer et al. 2017b, p. 1199.

However, (Gririraj Manoharan 2021, p. 109 f.) has shown that common strand lengths do not effect the intenity of the fluorophores noticeably.

3 The Picasso software package

3.1 Purpose and Structure of the Picasso Software

The Picasso software environment was created specifically to work with raw image data that was obtained by the DNA-PAINT method. It is structured into multiple subprograms and files to generate structural images from raw image data. That way all image creating efforts (except data aquisition, i.e. performing the DNA-PAINT method) can be handled by the Picasso software environment. It also provides interfaces such as the *Filter-program* for descriptive statistics and the *picassosr* library for more sofisticated data analysis.

If you want to download the Picasso environment you should check the corresponding GitHub repository

https://github.com/jungmannlab/picasso

and follow the instructions in the read-me file (readme.rst) to install the Picasso environment on your machine. There you will find two ways of installing Picasso.
The easiest way is to use the installer executable which installs the stand-alone program of Picasso (this contains all the subprograms) with desktop shortcuts for each subprogram.
Alternatively you can install Picasso as a python package which allows to use Picasso's internal routines in custom Python programs. The easiest way to archieve this is by having an anaconda distribution on your machine and then follow the steps of the read-me file to create an anaconda environment that supports the picasso python library called *picassosr*[5]. If you use this way of installation there will unfortunately be no desktop shortcuts for all the subprograms. You can still run them from the command line interface of your machine. Figure 3 shows how to do that on a windows operating system.

Since this document also deals with data analysis aspects of the imaged data (which means programming is necessary) in section 4, it is advised to go with the second installation route.

As already mentioned the Picasso environment consists of multiple subprograms, each with an individual purpose. A composition of all the subprograms is provided in Figure 4.

[5]Note: You should create a new anaconda environment for installing the *picassosr* package. To see why see subsection 3.5 or Wiesner 2024b.

```
Microsoft Windows [Version 10.0.19045.4780]
(c) Microsoft Corporation. Alle Rechte vorbehalten.

C:\Users\mariu>cd anaconda3

C:\Users\mariu\anaconda3>cd Scripts

C:\Users\mariu\anaconda3\Scripts>activate picasso

(picasso) C:\Users\mariu\anaconda3\Scripts>picasso localize

(picasso) C:\Users\mariu\anaconda3\Scripts>
```

Figure 3: Starting the picasso programs (here: Localize) from windows' command line interface.
Source: Captured on the author's machine.

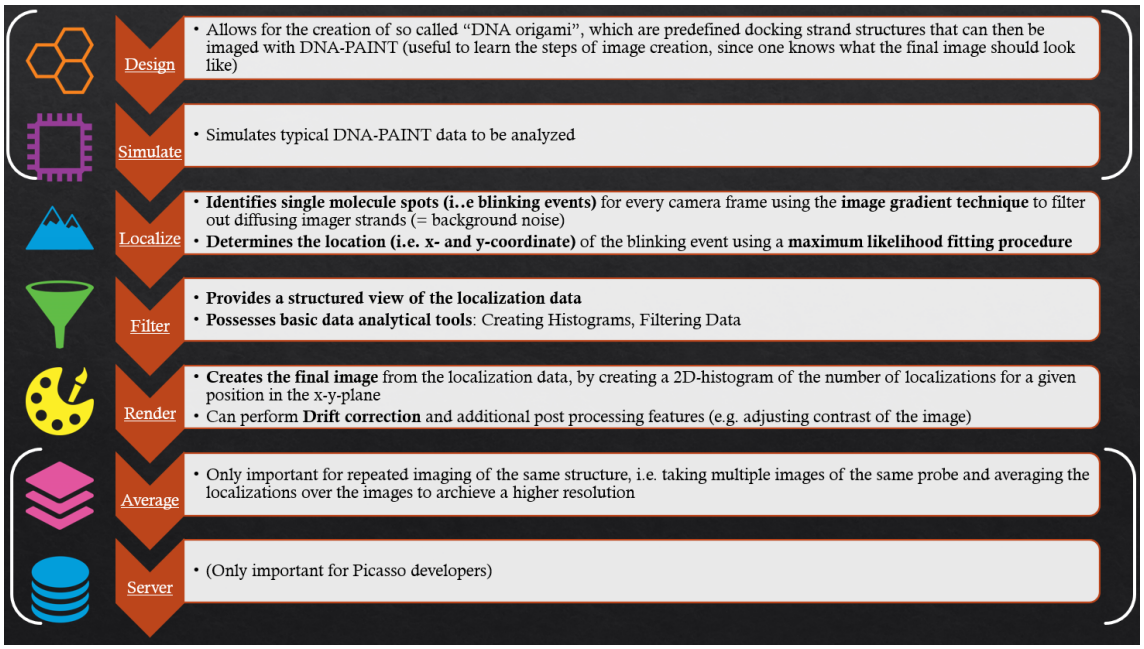


Figure 4: Subprograms of the Picasso software environment.
Source: Own creation.

The environment consists of 7 subprograms. For an in depth explanation of each program check the original picasso publication (Schnitzbauer et al. 2017b) or alternatively the official picasso documentation (Schnitzbauer et al. 2017a). We will focus on the 3 subprograms that are relevant for image creation and data analysis: Localize, Filter, Render.

3.2 Subprogram: Localize

The *Localize* program accepts raw camera image data[6] obtained by the DNA-PAINT Method and is able to localize blinking events for individual frames with sub camera pixel precesion. This is done in two steps, that can be run inside the program seperately or at once. An overview of the corresponding interface is given in Figure 5. Some of the parameters that can be adjusted in the parameter interface (see Figure 5, (b)) must be adjusted according to the camera specifications of the experiment which is explained in (Schnitzbauer et al. 2017a).

First the blinking spots are detcted by the program's spot detection algorithm . This algorithm implements a net gradient approach to minimize background noise. The net Gradient G_{net} (1) is a quantity calculated for a square area (= box) around points with the highest photon counts in their local neighbourhood (= local maximum). The minimum net Gradient can be adjusted to filter out noise and adjust resolution at the cost of detection quantity.

$$G_{\text{net}} = \sum_{\text{box}} \vec{g_i} * \vec{u_i} \tag{1}$$

It is calculated by multiplying the central difference gradient $\vec{g_i}$ of the intensity at pixel i with the pixel's unit vector in the direction to the center $\vec{u_i}$. This is done for every pixel inside the box and the results are summed up. The net Gradient thereby effectively calculates the intensity that flows towards the spot center. Apart from filtering out noise, the net Gradient can also be used to filter out isolated spots (See section 4 and (Wiesner 2024a), section: "Filtering out isolated spots by further analyzing Photon count and net Gradient" for more information on that.)

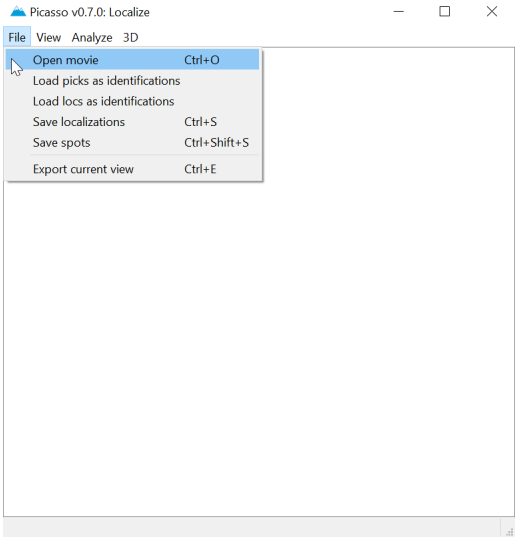
Secondly, the detected spots are "fitted". Since every spot posseses an associated point spread, i.e. the ilumminated spot is spread out and not perfectly point like, the program tries to determine the most likely position of the light source (i.e. the flourescent molecule) within the box of a spot. It does so by implementing one of three different maximum likelihood algorithms, which can be chosen before the program's algorithm is run. These algorithm's are:

[6]The accepted filetypes are listed in (Schnitzbauer et al. 2017a)

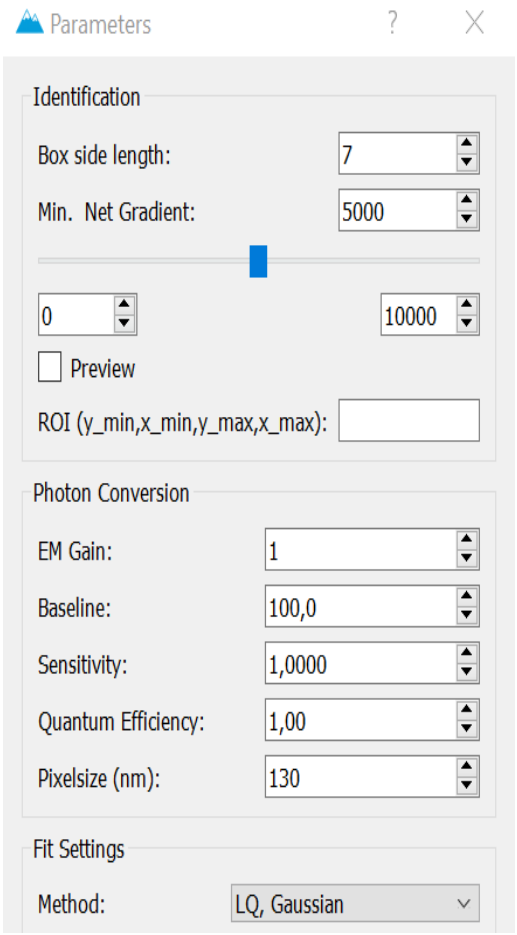
- MLE, integrated Gaussian
- LQ, Gaussian (least squares)
- Average of ROI (finds summed intensity of spots)

After running the spot identification and the fitting algorithm the localizations can be saved to a .hdf5 file accompinied by a .yaml file, which holds information on the localization specifications and is generated automatically by the Localize program. The yaml file can also be adjusted manually in a simple Texteditor. An example for such a yaml file can be seen in Figure 6 which was generated by camera data of SWCNTs that were imaged with the DNA-PAINT method.

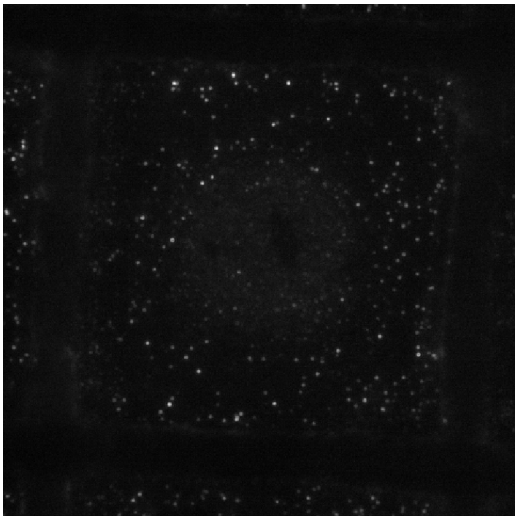
The localization data, i.e. the hdf5 file, can then be used for filtering, data analysis and the final image creation.



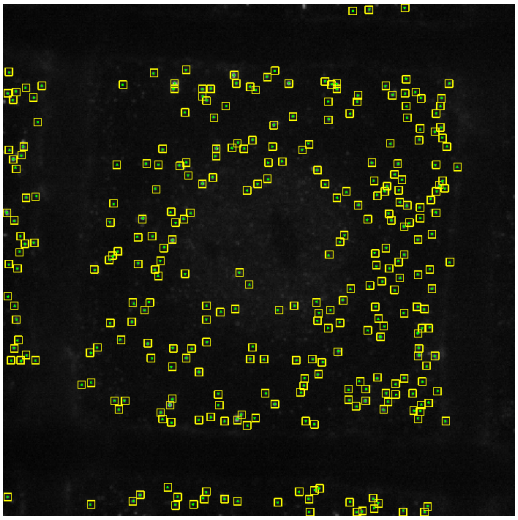
(a) File-menu to import raw image data (='movie file').



(b) Parameter specification for the spotting and fitting algortihm.



(c) Image data before before running spotting and fitting algorithm.



(d) Image data after running spotting and fitting algorithm.

Figure 5: Interface of the Localize program.
Source: Captured in the localize program by the author.

```
Byte Order: <
Data Type: int16
Frames: 40000
Height: 480
Width: 480
---
Box Size: 7
Fit method: LQ, Gaussian
Generated by: Picasso Localize
Min. Net Gradient: 8000
Pixelsize: 130
ROI: null
baseline: 100.0
gain: 1
qe: 1.0
sensitivity: 1.0
---
Filtered by: Marius Wiesner
```

Figure 6: A yaml file for storing localization specifications based on localization data of SWCNTs.
Source: Generated from SWCNTs' localization data by the author.

3.3 Subprogram: Filter

The *Filter* program offers a way to filter and analyze the localization data on a very basic level. It also provides a very convenient way of visualizing the localization data in tabular form. Figure 7 provides an example for SWCNTs' localization data that was opened with the Filter program.

Picasso: Filter. File: 220826_h2_143_60k_469wx481h_locs_render.hdf5

File Plot Filter

	frame	x	y	photons	sx	sy	bg	lpx	lpy	ellipticity	net_gradient
0	0	8.333531	352.1244	2195.9001	1.4159813	1.2804426	90.56728	0.061232064	0.052877028	0.095720686	10591.752
1	0	10.498271	117.929634	852.14233	1.2469183	1.1034104	87.24215	0.10733364	0.088451006	0.1150901	5082.1904
2	0	11.414806	242.23323	2984.789	1.0874194	1.1652932	103.093124	0.03492751	0.038243055	0.06682767	19342.4
3	0	13.907038	142.66939	718.628	1.064343	0.93110526	89.12655	0.09707695	0.07916979	0.12518308	5504.037
4	0	14.583622	281.3133	889.35956	1.2532604	1.2212492	88.74082	0.1049771	0.10070096	0.025542298	4926.0776
5	0	17.94731	180.82846	516.1395	0.9242142	1.1374615	95.362076	0.10552927	0.14766863	0.18747655	4321.947
6	0	32.882812	278.38202	3534.7168	1.2519146	1.3407894	89.73626	0.0363591	0.039809205	0.06628544	17793.709
7	0	35.35138	77.3307	679.06244	1.5977767	1.2173961	87.030495	0.19966684	0.12538588	0.23806864	3547.462
8	0	36.113525	434.484	3000.8428	1.1812611	1.0752293	99.096954	0.03848095	0.034046583	0.08976151	19503.736
9	0	38.520267	249.64894	850.2804	0.91379035	1.0385996	98.7846	0.06939897	0.0840627	0.12017072	6242.647
10	0	40.22122	241.93845	799.874	1.451975	1.2204254	93.21861	0.14986181	0.11204268	0.15947218	4239.023
11	0	46.641506	209.96686	499.4986	1.0885546	1.1419244	88.81861	0.13749753	0.14885429	0.046736684	2959.2017
12	0	48.225033	420.5546	506.1769	0.9770197	1.7306471	89.94121	0.11467025	0.30728376	0.43545988	2637.7847
13	0	49.241673	310.68747	536.99774	1.3975419	1.1189473	88.98687	0.19724591	0.13509654	0.19934617	2524.9326
14	0	52.00119	293.7804	2553.88	1.1158346	1.0823444	95.97313	0.03971578	0.038154546	0.03001357	16945.188
15	0	55.96619	75.81116	271.0296	0.71270984	0.6801865	91.22781	0.12161899	0.11351622	0.045633346	3023.447

Figure 7: The Filtering program describing the localization data in tabular form. Source: Captured in the Filter program by the author.

There are 11 columns describing the localizations (plus an additional column on the far left counting the localizations). Each row represents a localization within a camera frame that was spotted and fitted by the Localization program.

The first column represents the camera frame in which the localization was spotted. The fitted grid coordinates are saved in the **x** and **y** coordinates. The **photons** column describes the number of photons collected from a spot, while **sx** and **sy** describe the point spread, i.e. the size of the box for the net gradient calculation in pixel units. The precision of the sub-camera pixel localization (in camera pixels), i.e. the standard deviation, is given with **lpx** and **lpy**. Finally, the **net Gradient** is represented in the last coloumn.

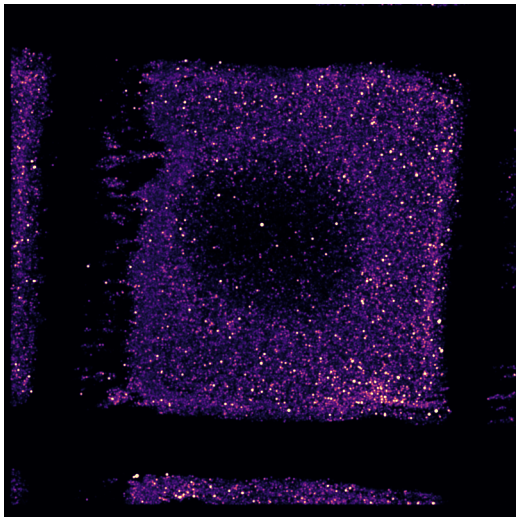
It is also possible to filter the data by specifying min. and max. values for the coloumns (e.g. a minimum and maximum photon count) and to create one- and two-dimensional histograms of individual coloumns. However, for more sophisticated filtering and analysis of the data the python library needs to be leveraged which will be explained in section 4.

3.4 Subprogram: Render

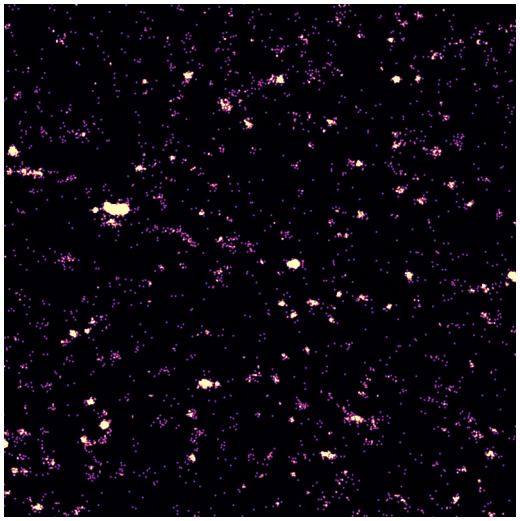
The program for creating the final image of the structure from the localization data is *Render*.

The Render program takes, just like the Filter program, localization data (i.e. hdf5 files accompinied by a yaml file[7]) as input to create a final image of the localization data. Pictures of the program's interface can be seen in Figure 8. The program creates the image by constructing a two dimensional grid of the x and y coordinates of the localization data. Additionaly, it creates a contrast that is proportional to the photon count of each localization, i.e. the intensity. Things like contrast colours, contrast range and pixel blurring can be adjusted in the program's settings (see Figure 8 (b)). It is also possible to zoom in and out to explore specific regions of interest.

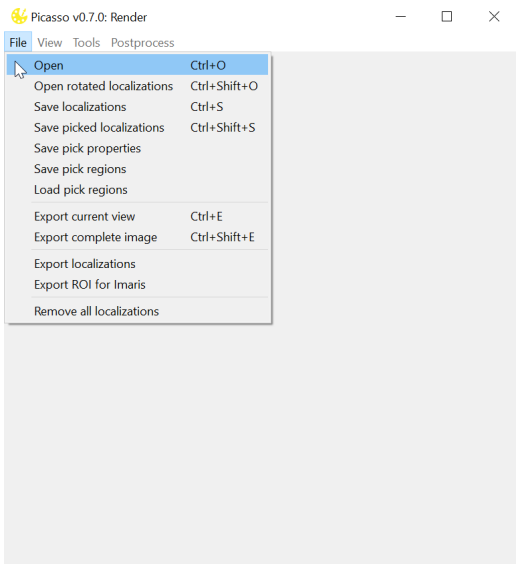
[7]Note that every hdf5 file needs to have a corresponding yaml file with the same name. Otherwise Picasso will throw an error.



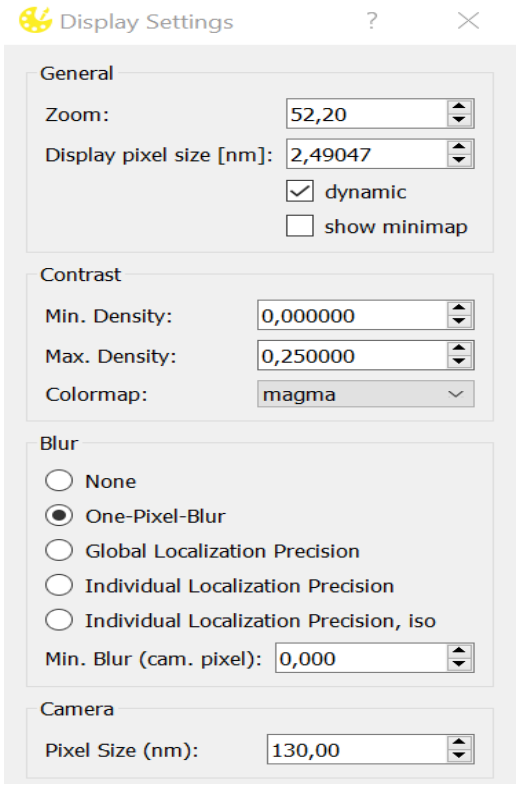
(a) Rendered imag of a grid with SWCNTs functionalized and imaged via the DNA-PAINT method.



(b) Zoomed-in image of (a).



(c) Dialogue for importing localization data into the Render program.



(d) Rendering settings.

Figure 8: Interface of the Render program.
Source: Captured in the Render program by the author.

3.5 The Picasso Library for Data analysis

As previously mentioned (see section 2) the Picasso software comes with a python library called *picassosr*. This library contains an inout-output-module (called *io*) for importing localization data, i.e. hdf5 files, into your custom python script, where it can be analyzed and filtered using either the *postprocess* module or other python libraries (e.g. Pandas, NumPy, etc.). Using the input-output-module, the localization data can be exported into a hdf5 file afterwards to render the final image via the *Render* program.

If you want to work with the *picassosr* package you should have used the second installation route by installing the package into a new anaconda environemnt (see section 2). The reason for using a new environment is that the *picassosr* package is dependent on older versions of numpy, matplotlib, etc. and other libraries (such as NumPy) may cause compatibility issues. For that reason one should use a different environment (i.e. a standard environment with all up to date packages) to analyze and filter the localization data (see the (Wiesner 2024b) notebook for that part in comnbination with section 4).

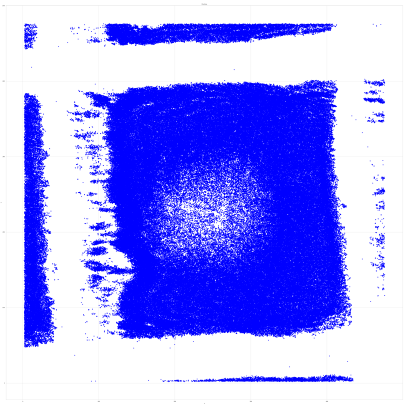
The Jupyter Notebook "WorkingWithThePicassoLibrary.ipynb" (Wiesner 2024b), that is attached to this document, provides the code and more in depth explanation to work with the picasso python library.

4 Application: Imaging Single Walled Carbon Nanotubes with DNA-PAINT and analyzing functionalisation properties

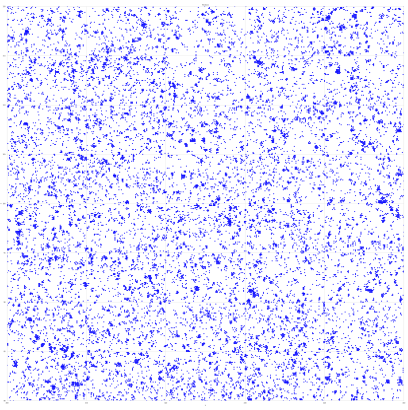
To demonstrate how the DNA-PAINT imaging method combined with the Picasso software can be used to create images of nano structures and how to analyze the corresponding data, the attached Jupyter Notebook "WorkingWithPicassoLocalizationData.ipynb" (Wiesner 2024a) was created. It contains the code for filtering and analyzing localization data in a python script and the detailed explanation of it. The Notebooks code can be used for any localization data, but was written specifically for the localization data of SWCNTs.

Figures 9a and 9b show images of SWCNTs that were created with the notebooks code.
Figures 9c and 9d show some statistics on the localization data created by the code.

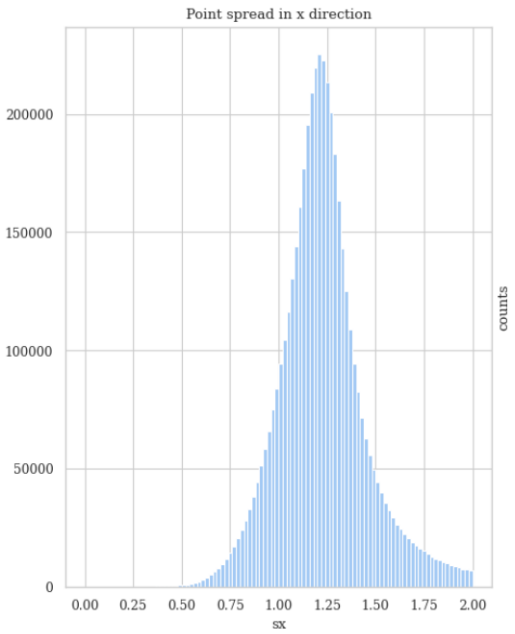
For more information check out the Notebook (Wiesner 2024a).



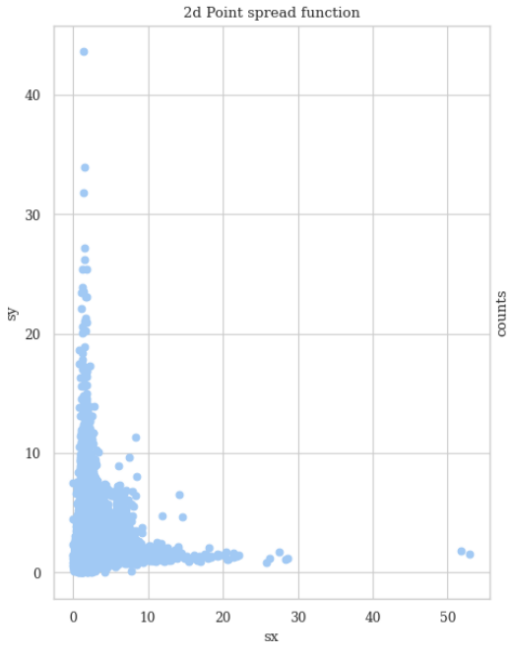
(a) Image of SWCNTs captured with the DNA-PAINT method.



(b) Zoomed-in image of (a).



(c) Distribution of the point spread in x direction.



(d) Distribution of the 2d point spread.

Figure 9: Images and parameter distributions from the Juypter Notebook. Source: Own creation.

References

Manoharan, G. et al. (2023). "Click-Functionalization of Silanized Carbon Nanotubes: From Inorganic Heterostructures to Biosensing Nanohybrids". In: *Molecules* 28, 2161. URL: https://doi.org/10.3390/molecules28052161.

Manoharan, Griririaj (2021). "Click Fucntionalization of Carbon Nanotubes for Nano-Bio Applications". PhD thesis. Department of Physics, University of Osnabrück.

Schnitzbauer, Joerg et al. (2017a). *Picasso documentation*. URL: https://picassosr.readthedocs.io/en/latest/?badge=latest.

— (2017b). "Super-resolution microscopy with DNA-PAINT". In: *Nature Protocols* 12, pp. 1198–1228.

Swathi, R.S. and K. L. Sebastian (2010). "Excitation Energy Transfer from a Fluorophore to Single-Walled Carbon Nanotubes". In: *Journal of Chemical Physics* 132(10), 104502. URL: https://pubs.aip.org/aip/jcp/article-abstract/132/10/104502/188887/Excitation-energy-transfer-from-a-fluorophore-to?redirectedFrom=fulltext.

Wiesner, Marius (2024a). *WorkingWithPicassoLocalizationData.ipynb*. URL: https://github.com/username/repository/notebook.ipynb.

— (2024b). *WorkingWithThePicassoLibrary.ipynb*. URL: https://github.com/username/repository/notebook.ipynb.