DDC User Guide

By Christopher H. Bohrer and Ryan McQuillen (If you have any questions or issues please email us!!)
bohrerch@nih.gov or cbohrer3141@gmail.com

This is a user guide for **DDC**, an algorithm that corrects for blinking artifacts generated during Single-Molecule Localization Microscopy (SMLM) acquisition using pair-wise distance distributions (PWDD). Herein we will discuss the two parameters that need to be defined in order to use DDC as well as the innerworkings of the code (originally written for MATLAB but can be easily extrapolated to other platforms). Lastly, we now also discuss how to implement photon-weighted averaging within DDC, to potentially improve the resolution of the images. To properly implement DDC the following considerations must be made:

1. The raw data must be properly organized.
2. Images with a large number of localizations must be split.
3. The two main parameters (*N* and *Bin Resolution*) must be properly defined.

Data Organization & Qualitative Blinking Visualization

As stated above, in order to implement DDC the raw data must be organized in a particular way and saved within a **.mat** file.

- First, localizations need to be stored within a cell called **LocalizationsFinal**, where **LocalizationFinal{1}** contains the **[x,y,z]** vector coordinates of the localizations within the first image and **LocalizationsFinal{i}** contains the vector coordinates for the *i*th image.

- Secondly, the frame numbers corresponding to data in each image should be stored in **Frame_Information** (*i.e.* **Frame_Information{i}** are the frames for the localizations in the **LocalizationsFinal{i}** dataset).

- If you are going to utilize photon-weighted averaging, you should have the number of photons for each corresponding localization stored within a cell called **Photons** (Similar to above). Otherwise, if you still choose to utilize this functionality, for the photon weighted averaging output DDC will assign the same weight to each localization. You will have the option to use photon weighted averaging later on.

An example of how your data should be organized is contained in the **.mat** file named **Example_3d_2darkstate_random_data.mat** enclosed in the folder of the DDC code in the folder **User_Guide_Files**. This file contains 3D simulation data with 100,000 localizations distributed over a 2500nm^3 volume. The distribution of these localization is not homogeneous and instead contains clusters on a similar size scale as the localization precision: ~15% of the localizations are within the clusters and the rest are randomly distributed through the volume. The photokinetics of the simulated fluorophores are the same as the 2-dark state fluorophores simulated within in the main text (**Supplementary**).

Splitting Images with a Large Number of Localizations

If your raw data contains a large number of localizations within each image (>7000) DDC will likely converge very slowly (depending on the computing power of your machine). This is due, in part, to the fact that DDC needs to calculate the PWDD between all localizations (which increases exponentially with the number of localizations).

Note: The computational power requirement to calculate the pairwise distance distributions can be reduced if a small time and distance threshold is applied. The theory corresponding to this approach checks out, but this method has not been extensively tested by us and therefore caution should be taken.

To circumvent the computational load, raw images can be split into segments and analyzed in parallel. To split up an image use the **Split_UP_Image_Multiple_Images.m** script within the folder **DDC_Tools**. Doing so will split up your image into individual images within the cell **LocalizationsFinal**. The number of localizations in these segments is set by *min_loc*, which in the case of the example script is set to 5000 by default.

Note: if you have millions of localizations within each image you may need to split up your images multiple times. One can easily do this by first splitting up an original image with custom code (make sure there is a bit of an overlap between the images so that there are no edge effects ~200nm overlaps) and then using our code you can split up each of these smaller images and then recombine them at the end. You may need to utilize a computer cluster if this is the case.

The **Split_UP_Image_Multiple_Images.m** script iterates through each image and splits it into segments containing (approximately) the correct number of localizations. It then adds a "buffer" zone around each image of ~200nm to mitigate boundary effects when recombining the segments. *Note: this "buffer" zone may need to be modified based on the resolution of the experiment. Note, one could easily write their own code to split up the localizations and recombine them.*

Upon running the **Split_UP_Image_Multiple_Images.m** script you will be prompted to select your data. *Note: The data in the large image should still (at a minimum) have the two cells containing the data **LocalizationsFinal** and **Frame_Information** (see prior section). Furthermore, there is also a cell which contains the true_localiations which will be used to evaluate DDC's performance at the end.*

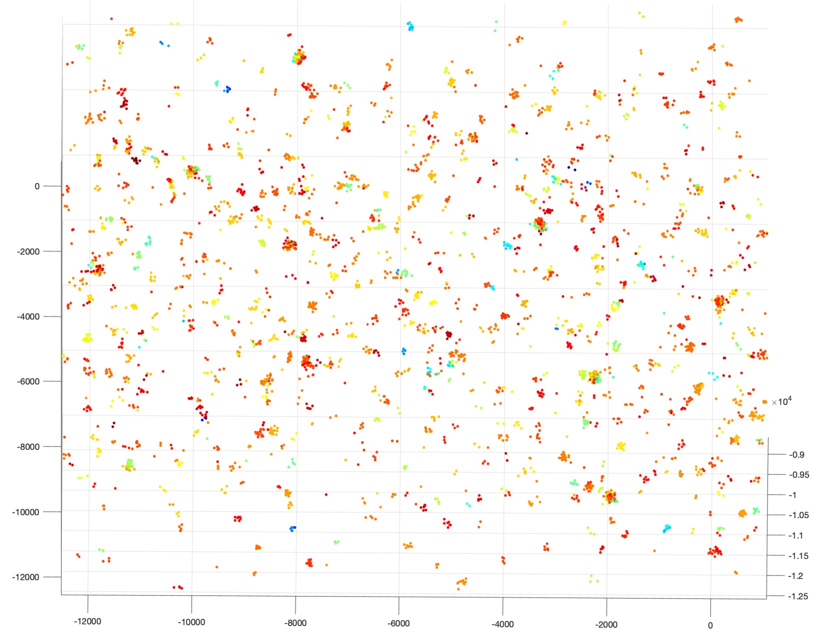


Figure 1: Sample of the localizations from the example dataset. Localizations are colored through time. Small punctate clusters of the same color indicate blinking. Small clusters that have multiple colors indicate actual clusters with a higher density of true localizations.

Click **RUN,** the program will then prompt you to select your data, doing so will initiate the script. Upon completion of the run, the **Split_UP_Image_Multiple_Images.m** will save a new **.mat** file with the same name as the original selected data with “**Split**” appended to the front. It will also generate a few 3D scatter plots colored through time.

An example scatter plot corresponding to the localizations from a single split image colored in time is shown in **Figure 1.** You will notice small punctate clusters of the same color—these are indicative of blinking. You can also see some punctate clusters of different colors—these are indicative of the real clusters. Regardless of whether one chooses to use DDC or not, it is always as good idea to look at a scatter plot of your data colored through time. As this will give you an idea as to what features are actually caused by the blinking of the fluorophores.

To recombine the image segments into a single large image after running the DDC algorithm you can use the script **Combine_Images.m** within the same folder. Running this script will produce a new analyzed data file with the same name as the originally selected image with **combined** appended to the front. *An example of how to do this can be found at the end of this guide.*

Determining the Bin Resolution Parameters & N

DDC requires two parameters to accurately eliminate blinking: (1) the bin resolution of the SMLM experiment and (2) the frame difference (*N*) at which steady state is reached. Each of these parameters and how to input them into DDC is described below.

Determining the Frame Difference (N)

In order for DDC to accurately eliminate the effects of blinking on SMLM data it needs to first obtain the "true pair-wise distance distribution" (TPDD) of the underlying molecules in the experimental data (*see the maintext for derivation and implementation within the DDC algorithm*). To determine the TPDD of your data you will first need to determine when the pairwise distance distribution for the localizations separated by a certain frame difference (*N*) reaches a steady state. To do so, you will need to run the **Determine_N.mat** script within the **DDC_Tools** folder.

To determine *N* we will plot the summed difference between the cumulative distribution function of the PWDD at a certain frame difference relative to the that obtained with a frame difference of 1:

$$Z(\Delta n) = \sum |cdf(pd(\Delta n)) - cdf(pd(1))|$$

Note: there are two parameters within this script, ***NFTP*** *and* ***Gap****. NFTP is simply the highest Δn where Z(Δn) will be calculated. This value is set to 1000 by default for the example dataset. If you are analyzing an experimental dataset for the first time it is better to start with a larger NFTP value rather than a smaller one --- go out to a few thousand.*

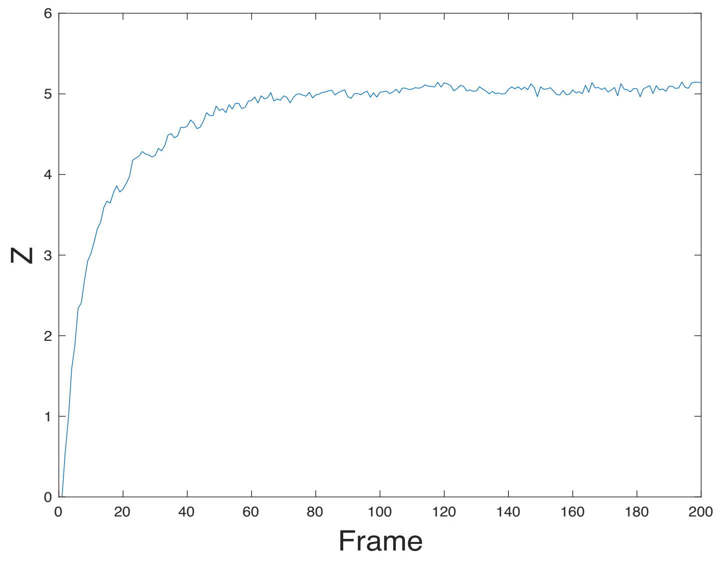


Figure 2: Summed distance values at various frame differences. Plot showing the change in the pairwise distance distributions at various frame differences relative to a frame difference of 1. You can clearly see that Z reaches a steady state after a frame difference of ~70. Therefore, a good estimation for N is any value above this, to be safe when applying DDC we will set N equal to 200 frames.

If NFTP is very large it can take a while for this script to work. You can get an idea of how long it is going to take based off of how fast the fraction complete increases once you click run. The parameter *Gap* changes the step size for Δn. So if it is equal to 1, Δn=1, 2, 3... will be investigated. But if *Gap=10,* Δn=1,11, 21... will be investigated. If the NFTP is big, decreasing *Gap* can increase the speed greatly.

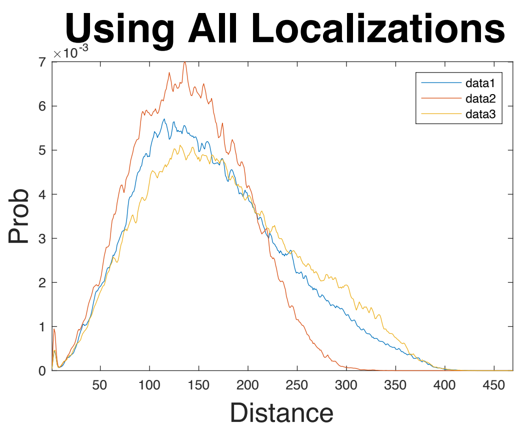
In **Figure 2** it's apparent that the pairwise distance distributions approach a steady state after ~70 frames. At this point, the PWDD between these localizations is approximately equal to the TPDD. Based off **Figure 2** any *N* above 70 frames would be sufficient to determine the TPDD. *However, to be safe, we set N to be 200 frames for our example data as the frame difference value cannot be overestimated if there is a sufficient amount of data.*

Determining the Bin Resolution

The final step toward implementing DDC is determining the bin resolution of the SMLM experiment. This value will be unique to your imaging setup and dataset. The bin resolution is the width of the bins for the histograms of the probability distributions (*e.g.* TPDD histogram).

To start, you will need to determine the "classic" resolution of your SMLM data. This can be accomplished in a number of ways and can be easily found. *For the example dataset the resolution of the SMLM data is 40nm (or 1 sigma).*

*Note/Update: DDC has a new script called "**New_Determine_Res.m**" which can calculate the bin width for you (in the **DDC_Tools** folder). All you need to do is open the script, input your determine* **N** as **N_f** toward the top of the script and click run. You will then be prompted to select your data (Use the split images if you have a large number of localizations, or else you will probably run out of memory). Once the script is done running it will state a bin "resolution" (about 2x the classical resolution) that you can use to obtain good results with DDC! You should still check to make sure that the bin resolution gives "smooth" true pairwise distance distributions, see below.



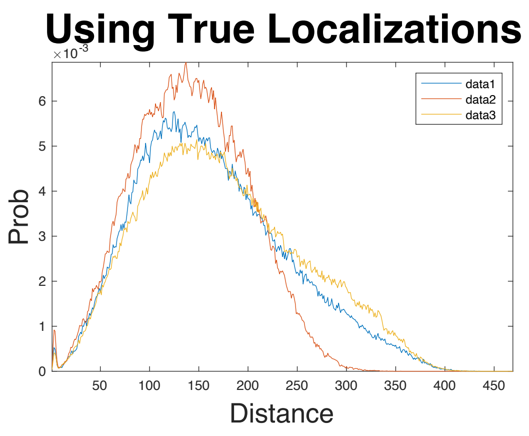


Figure 3: TPDD plots. Here you can see the true pairwise distance distributions for the first 3 images (top plot) using a bin width of 40nm (equal to the "classical" resolution of the simulation data). You can see that given the amount of data and *N* the TPDD using all localizations is well defined and is very close to the TPDD determined only using the True Localizations (bottom plot).

To get an idea of how well the TPDD for each image segment are sampled, you will need to run the **Determine_Res.m** script in the **DDC_Tools** folder. Input your "classical" SMLM resolution x2 (Or the bin width determined by the new script) at the top of the script. This will serve as a starting point for determining the bin resolution for DDC. Additionally, you will also need to input N_f (determined in the previous section) at the top of the script. This is necessary for generating the TPDD.

Note: 2x the "classical" resolution is normally adequate for defining the bin widths of the probability distributions used within DDC. This value is not necessarily set in stone and can be smaller than 2x the normal SMLM resolution if the probability distribution is still well sampled. As long as the pairwise distance distribution looks relatively smooth (like to the left), with an adequate probability in each bin, you are ok (the plots to the right are bin width=40nm). Note, results are not really that sensitive to specific values of this parameter.

After running this script, two plots (**Figure 3**) of the TPDDs for the first 3 image segments will be generated: one using **all** of the localizations in the images and one using only the **true** localizations (*i.e.* the simulated ground-truth localizations). *Note: if you are using DDC on experimental data you will not be able to generate the second plot. (we ran the script with 1x the classical resolution of the experiment, as it was still well sampled at this small of a bin width). Therefore, one could utilize any value between 40nm and 80nm for this parameter for this dataset and DDC would work fine.*

Figure 3 illustrates that the TPDD plot using all the localizations are approximately the same as those using only the ground truth localizations and that all the probability distributions are well-defined. This suggests that, in this case, 1x the “classical” resolution is sufficient to create the probability distribution. *Note: Generally, you do not want to go lower than the 1x the “classical” resolution of the experiment.*

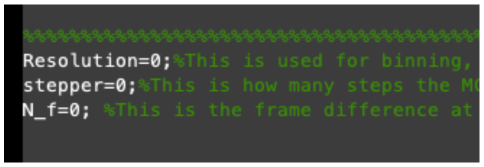
Again, it should be noted that DDC is fairly insensitive to the accuracy of the bin resolution measurement. This parameter simply defines the bin size of the PDD and should just be large enough that there are a sufficient number of distances within each bin.

We are finished determining the two parameters needed to run DDC for this dataset (*N* = 200 & *Res* = 40nm to 80nm).

Running DDC

With the two parameters (*N* and *Res*) determined using the methods outlined above, we can run DDC on the image segments. To do so, you will run the **run_scipt_DDC.m** script. Prior to doing so you will need to set four values within the script (see **Figure 4**):

1. **Resolution** which is the bin width determined in the previous section (40nm to 80nm for the example dataset).
2. **N_f** which is the frame difference value (*N*) determined in the previous section (200 for the example dataset).
3. **stepper** which is defined as the number of steps that the Markov Chain Monte Carlo (MCMC) will go after reaching a new maximum of the Likelihood (set to 100 for the example dataset, more than enough).
4. (New addition) **Photon_weighted_Correction**, setting this equal to one will cause DDC to average the locations of localizations that share a trajectory (weighted according to the number of photons in each one), potentially improving the resolution. Note that if one is having issues with blinking and there are incorrect definitions of trajectories, linking them together could give a false sense of resolution with many metrics, and one can easily see how the mis-linking would lead to additional artifacts.



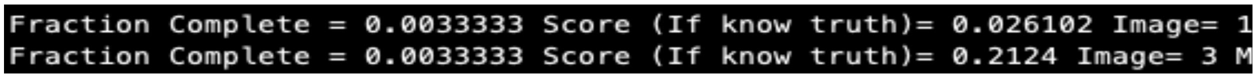
```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Resolution=0;%This is used for binning,
stepper=0;%This is how many steps the MC
N_f=0; %This is the frame difference at
```

Figure 4: Locations of the three variables that need to be set in the DDC script.

Now that everything is setup you may initiate the DDC script by clicking **RUN**. After doing so you will be prompted to select your split image file. After doing so DDC will start working. First, DDC will complete **STEP 1** and will alert you when it is finished.

STEP 2 depends upon your hardware configuration and runs in parallel. For a 6-core setup DDC took ~5hr to finish the example dataset run. *Note: If you do not have time to wait for this the example dataset the analyzed data is stored in the* **Analyzed_...2019_Split_Example_3d_2darkstate_random_data.mat** *file.*

The output during a run looks like the following:



```
Fraction Complete = 0.0033333 Score (If know truth)= 0.026102 Image= 1
Fraction Complete = 0.0033333 Score (If know truth)= 0.2124 Image= 3 M
```

The **Score** is the relative image error if the truth is known and is described in the maintext.

When DDC is finished, the final localizations with their associated frames will be stored in the following two cells: (1) **Final_Loc_Bliking_Corrected** and (2) **Final_Frame_Blinking_Corrected**. These will either be the true localizations (as defined within the main text) or the photon weighted average positions. The **.mat** file will be stored within the same location as the **run_scipt_DDC.m**. Useful note, DDC saves out data throughout the run so you can stop it and get it going again with ease. To get it going again, first load in your data and then run the section with the **parfor loop**, this will start the analysis again where you left off.

Combining the Split Images

Once DDC is finished running on the split image segments, they will need to be recombined together. To do this you should run the **Combine_Images.m** script within the **DDC_Tools** folder. As with the other scripts you will click run, select your analyzed data and the script will generate a new file with the same name as the selected data but with “**Combined**” appended to the beginning. The localization will be stored in two arrays: **Final_Loc_Blinking_Corrected** and **Final_Frame_Blinking_Corrected**. Note, if **Photon_weighted_Correction** was set to one, the localizations considered to be within the same trajectory were already linked together.

. You can then visualize the localizations by whatever method you choose.

Evaluating the Performance of DDC on Split Images

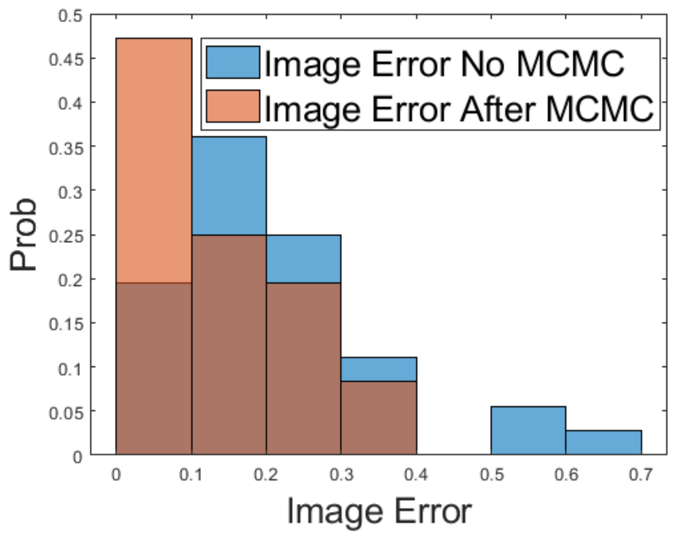


Figure 5: Performance improvement in the image error by performing the MCMC within DDC. As you can see there is a very significant improvement in terms of the relative image error when you compared the distribution before (blue histogram) and after (peach histogram) the MCMC phase space search.

Note, this section will only be really relevant if **Photon_weighted_Correction** was not utilized.

To illustrate the effectiveness of DDC's ability to eliminate blinking artifacts by maximization of the likelihood, you can plot the Relative Image Errors before and after the MCMC phase space search. This data was saved out during the DDC run and is stored in a file called **FILENAME**.

To look at the effect of the MCMC search you can use the **Evaluate_Your_Data_USER_GUIDE.m** script within the folder **User_Guide_Files**. Then load the *Analyzed_file* that DDC originally output and click **RUN**. Doing so will generate a plot showing the overlapping image error distributions for before and after the MCMC search (see **Figure 5**). In **Figure 5** you can clearly see a dramatic improvement in the **Image Error** after MCMC (peach color).

Other Useful side notes:

Investigate which Localizations are linked into trajectories: one of the outputs of DDC is the trajectory of each localization. This information is saved within the cell **Trajectory_of_Localizations{i}**. One could possibly use this information to increase the resolution of the experiment, but caution should be used (New addition: This is exactly what setting **Photon_weighted_Correction** to 1 does) --- for if one is having issues with blinking and there are incorrect definitions of trajectories, linking them together will give a false sense of resolution with many metrics, and one can easily see how the mis-linking would lead to additional artifacts. Overall though, DDC does an excellent job liking localizations into the correct trajectories, **Figure 6**.

Note, the specific number assigned to each trajectory is not important.

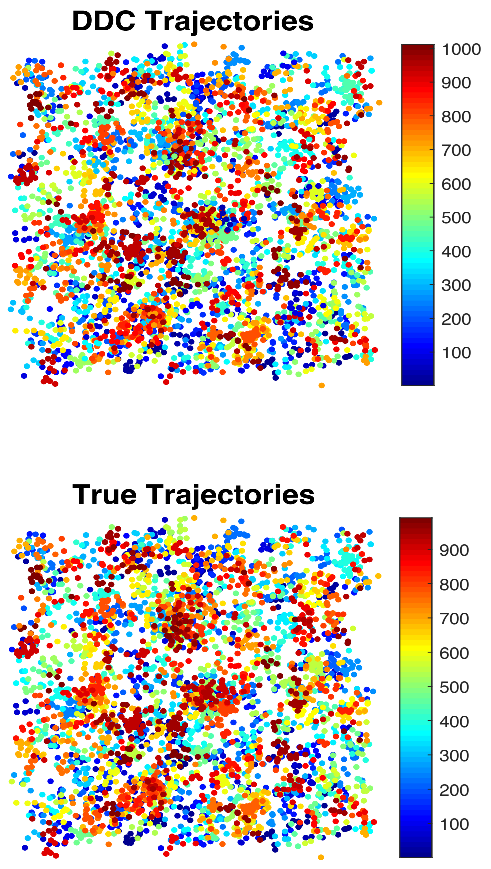


Figure 6: Example of the trajectories determined by DDC.