Recon3D - a reconstruction software suite for DFXRM

V. 0.02

Alberto Cereser

September 23, 2017

Contents

1 Introduction **2**
1.1 For developers . . . 2
1.2 Reconstruction steps . . . 2
1.2.1 Data loading, cleaning and storing . . . 2
1.2.2 3D reconstruction . . . 4
1.3 How to run the scripts . . . 4
1.3.1 check_threshold.py . . . 4
1.3.2 estimate_precession.m . . . 5
1.3.3 getdata.py . . . 5
1.3.4 img_sum.py . . . 6
1.3.5 plot_angles.py . . . 6
1.3.6 plot_img.py . . . 6
1.3.7 plot_sum.py . . . 6
1.3.8 Rebin_img.py . . . 6

2 Recommendation for data acquisition **6**
2.1 Data acquisition script . . . 6

3 getdata.py **7**
3.1 Input data . . . 8
3.2 Output . . . 10
3.3 Data loading . . . 10
3.4 Data cleaning operations . . . 10

4 Recon3d.py **15**

5 Validation of the shape reconstruction procedure **15**
5.1 Validation using ASTRA 15
5.2 Validation using Art + TV 15

6 To dos **15**
6.1 Critical 15
6.2 Will make life easier 15
6.3 Would be nice to have 16

7 Code contributors **16**

1 Introduction

Recon3D is a software package to analyze datasets collected using dark-field X-ray microscopy (DFXRM) [1], a technique under development at the European Synchrotron Research Facility (ESRF), beamline ID06. With DFXRM, the 3D shape of a deeply embedded grain, and the distribution of the crystallographic orientation inside its volume, is reconstructed from the signal collected in diffraction mode. Data are collected varying three angles: the sample rotation angle ω and the "rock'n'roll" angles γ and μ [2]. Usually, the same number values, and the same angular increment, is used for the two angles γ and μ.

Recon3D has two main programs, one to load, preprocess and store the collected dataset, and another to reconstruct the sample in 3D (shape and orientation). In detail, the files are:

- `getdata.py`, which loads the collected dataset, cleans the images and stores them. In the current version, the software is designed to only reconstruct well-centered grain (see Fig 1)
-

1.1 For developers

Recon3D is an open source project hosted on GitHub. To contribute to the code development, clone[1] the Recon3D distro and have fun! Except for a few Matlab scripts, all code is written in Python.

1.2 Reconstruction steps

1.2.1 Data loading, cleaning and storing

1. Have a look at the images using fabian or `plot_img.py`
2. If necessary, rebin the data using `Rebin_img.py`

[1] If you don't know what cloning means, check this introduction to Git and GitHub.

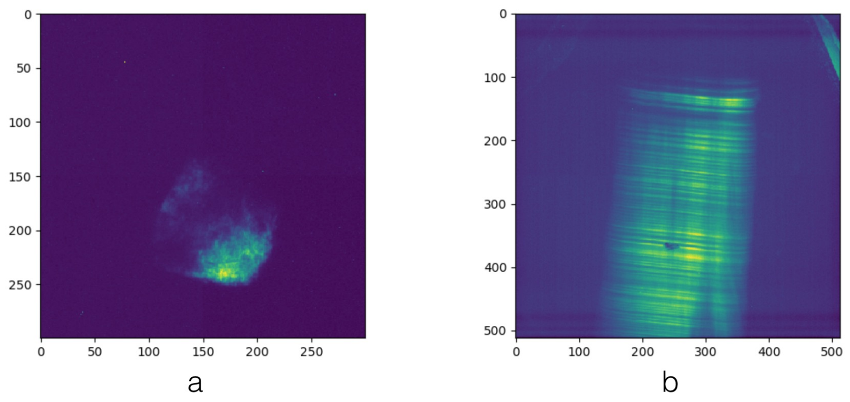

Figure 1: In the current version (v. 0.01), `getdata.py` is designed to take as input frames collected investigating a well centered grain, with the diffraction signal that doesn't touch the borders of the frame, as in *a*. The capability to reconstruct samples touching the detector frame (*b*) will be later implemented. *a* shows a grain in an Al sample (courtesy of Annika Diederichs, DTU MEK), ROI size: 300x300; *b* shows a crystal in a biomineral sample (courtesy of Phil Cook, ESRF), ROI size: 512x512 (entire detector).

3. Load the collected images using `getdata.py` with 0 as threshold value
4. Plot the processed images with `check_threshold.py`. Look at a few images and select a threshold value. In the next step, all intensities below the threshold value will be set to zero
5. Look at the distribution of the projection angles. Is there anything anomalous that should be taken into account? Script: `Plot_angles.py`
6. Run `getdata.py`, this time using the threshold value found in the previous step
7. For each projection, sum all collected images using `img_sum.py`
8. Save each image sum in a separate file using `plot_sum.py`
9. Load the result as a stack in ImageJ or FIJI. *At present, nor ImageJ nor FIJI are installed on Panda2. Download the data and have a look at the images on your laptop*
10. Check how the diffraction projection evolves as the grain rotates. Does the rotation axis move? Are there other grains passing by the detector, as in Fig. 2?
11. If some projections have to be discarded, change the `leno` loops in `getdata.py`: in this case, we only want to load selected projections. Example: to select projections 1 to 95, 121 to 155 and from 157 to 160, `range(leno)` becomes `(range(96) + range(121,156) + range(157, 161))`. If this is the case, repeat point 5-8

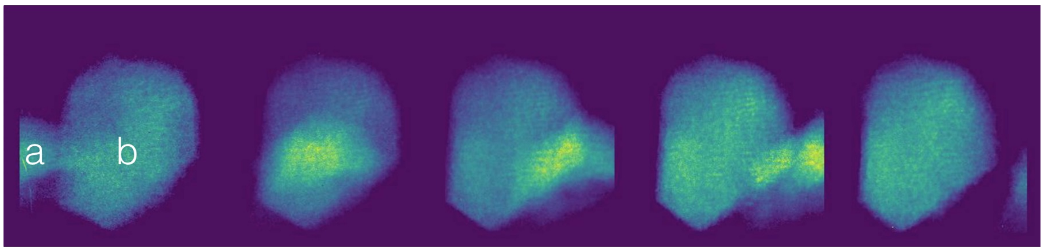

Figure 2: While scanning grain a as a function of γ, μ and ω, the transit of grain b was also recorded. This should be avoided.

1.2.2 3D reconstruction

1. Calculate the (possible) rotation axis shift using `Estimate_precession.m`
2. Reconstruct the shape of the sample and, for each voxel, calculate the (γ, μ) angles for which the maximum intensity was recorded. Code: `recon3d.py`
3. Convert the output of `recon3d.py` to a `vtk` file, that can be visualized with ParaView
4. Play with the 3D reconstruction in ParaView
5. Compare the reconstructed volume with the experimental data using `vol3D.m`

To do:

- List scripts, explain the functioning of each
- Plot how to use them in a chart

1.3 How to run the scripts

1.3.1 check_threshold.py

Script to select which threshold to use to clean the input images with `getdata.py`.

Command:

```
$ python check_threshold.py [Data directory] [Modality] [Size frame background subtraction] [Number of projections to consider]
```

- `Data directory` is the directory where files returned by `getdata.py` are stored
- `Modality`: 1 to show, for each projection, all collected images in an array (like Fig. 3); 2 to show, one by one, all images collected at a certain projection
- `Number of projections to consider` defines how many projections to take into account. These are selected by dividing the angular range covered during the tomographic scan in the selected number of steps

1.3.2 estimate_precession.m

1.3.3 getdata.py

Script to load the `.edf` files (images and data from header), organize the information and return data in a series of numpy array. Presented in detail in Sec. 3.

Command:

```
$ [mpirun -n NN] python getdata.py [Data directory] [Name data files] [Point of interest] [Image size] [Output path] [Output directory] [Initial phi value] [Initial chi value] [Angular step] [Number of angular steps] [Size frame background subtraction] [Image binarization threshold]
```

Inputs:

- `mpirun -n NN` is the command to run the script using parallel computing (multiple processors). The option is available on Panda2. The number of processors to use is `NN`. If mpirun is not available, erase this command
- `Data directory` is the directory where data are stored
- `Name data files` is the part of the file name that is common to all files (without incremental numbers)
- `Point of interest` is the center of the ROI to consider
- `Image size` is the size of the ROI to consider. Running the reconstruction code on Panda2, the maximum ROI size is 512×512. If the frames are bigger, rebin them using 1.3.8
- `Output path` is the path where to store the output directory
- `Output directory` is the output directory
- `Initial phi value` is ϕ_0 in Eq. 1
- `Initial chi value` is χ_0 in Eq. 3
- `Angular step` is the angle, in degrees, used to increment χ and ϕ in the data acquisition script
- `Number of angular steps` is the number of considered χ and ϕ increments
- `Size frame background subtraction` is the size of the cornice used to clean the image background, to consider possible spatial anisotropies of the detector response. For more details, see 3.4. *In a future version, the option to reconstruct grains occupying the entire detector frame (no cornice) will be included*
- `Image binarization threshold` is the threshold used to select the diffraction spots and set the outside region to zero. For more details, see 3.4

Example:

```
$ python getdata.py /u/data/andcj/hxrm/Al_april_2017/ c6_topotomo_frelon_far_
256,256 300,300 /u/data/alcer/DFXRM test_2 0.69 -1.625 0.0585 11 20 12
```

1.3.4 img_sum.py

Sum, for each projection, all images collected by varying γ μ. Before saving them, have a look at the summed images and select upper and lower threshold values.

Command:

```
$ python img_sum.py [Data directory] [Modality] [Lower threshold] [Upper threshold]
```

- `Data directory` is the directory where the output from `getdata.py` is stored
- `Modality`: 1 to determine threshold values, 2 to check the rotation axis and 3 to prepare the data for ASTRA
- `Lower threshold` is the possible lower threshold
- `Upper threshold` is the possible upper threshold

1.3.5 plot_angles.py

Shows the distribution of the ω angles where the sample was scanned.

1.3.6 plot_img.py

Simple script to plot `.edf` files. To show a file, change path in the code. Alternatively, use the fabian module included in FabIO [3]

1.3.7 plot_sum.py

Saves the output from `img_sum.py` in `png` files. In this way, the sum of all image scollected at a certain projection is saved in a different image. *Could be included in* `img_sum.py`

1.3.8 Rebin_img.py

Script to rebin the topotomo frames, to make the dataset manageable for Panda2. Change paths is the script.

2 Recommendation for data acquisition

2.1 Data acquisition script

`getdata.py` is designed to treat data acquired using a macro script structured as follows:

```
1  def topotomoscan '{
2
3      getinitpos
4
5      _omegastart = 0
6      _omegaend = 180
7      _omegastepsize = 0.8
8      _omegasteps = (_omegaend-_omegastart)/_omegastepsize
9
10     for(_omegai = 0; _omegai <= _omegasteps; _omegai += 1){
11
12         _omega = _omegastart + _omegai*_omegastepsize
13         printf("\n OMEGA = %g\n", _omega)
14         umv diffry _omega
15
16         _phistepsize = cos(rad(_omega))*0.032
17         _chistepsize = sin(rad(_omega))/cos(rad(_zpchi))*0.032
18         for(_secondi = 0; _secondi <= 6; _secondi += 1){
19             _phi = _zpphi + (_secondi-3)*_phistepsize
20             _chi = _zpchi + (_secondi-3)*_chistepsize
21             printf("\n chi = %g, phi = %g\n", _chi, _phi)
22             umv chi _chi phi _phi
23
24             zapline diffrz _zpdiffrz-3.5*0.032 _zpdiffrz+3.5*0.032 7 2000
25         }
26
27     }
28
29
30     movetoinit
31
32 }'
```

In particular, it is important that `chistepsize` and `phistepsize` are described as above (lines 16 and 17). If data have been acquired in the opposite way

```
        _chistepsize = cos(rad(_omega))*0.032
        _phistepsize = sin(rad(_omega))/cos(rad(_zpphi))*0.032
```

In `getdata.py`, function `calcGamma(self, data)` substitute `self.meta[ind,0]` with `self.meta[ind,1]`, and `data.alpha0` with `data.beta0`.

3 getdata.py

`getdata.py` is a script to load, clean and store the `.edf` files collected during to a topotomo scan. The goal of the data cleaning steps is to minimize the noise contribution, enhance the signal from the diffraction spot and normalize the intensity values from different projections.

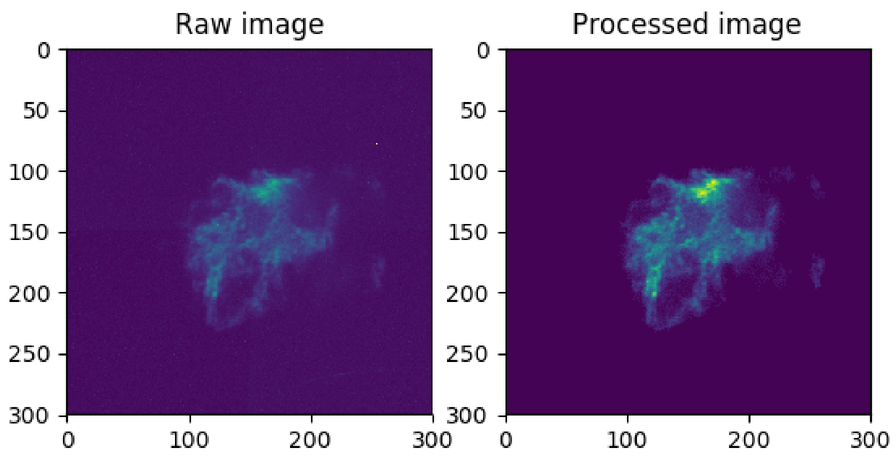

Figure 3: Typical dark-field x-ray microscopy (DFXRM) image before (*left*) and after (*right*) the data processing procedure here presented.

3.1 Input data

Fig. 4 shows the frames recorded, at a given projection, varying γ and μ by a constant angular increment (*topotomo scan*). From the image, it is evident that the shape of the recorded diffraction spot noticeably changes among frames. This agrees with what expected: for different combinations of γ and μ, the Bragg condition is satisfied by different substructures of the considered grain.

As input, `getdata.py` loads a set of images collected during a scan by varying the ω, γ, μ angles. The images are saved as `.edf` files, each consisting of the actual image and of a header, storing the parameters at which each image has been collected (e.g. motor position, ring current, . . .) For each image, `getdata.py` reads the header and the image in a five-dimensional matrix: two dimensions are used to store the image, and the other three determine its position in the space of the angles considered during the topo-tomo scan. The angles are:

- ω, the sample rotation angle
- μ, the in-plane sample tilting angle
- γ, the sample tilting angle perpendicular to μ. The value of γ is determined by a *pseudomotor*, which is not a real motor but instead a combination of the motors controlling the angles χ and ϕ (a.k.a. α and β), regulated so that γ and μ always move on two perpendicular planes. In the data acquisition script in Sec. 2, χ and ϕ are defined as

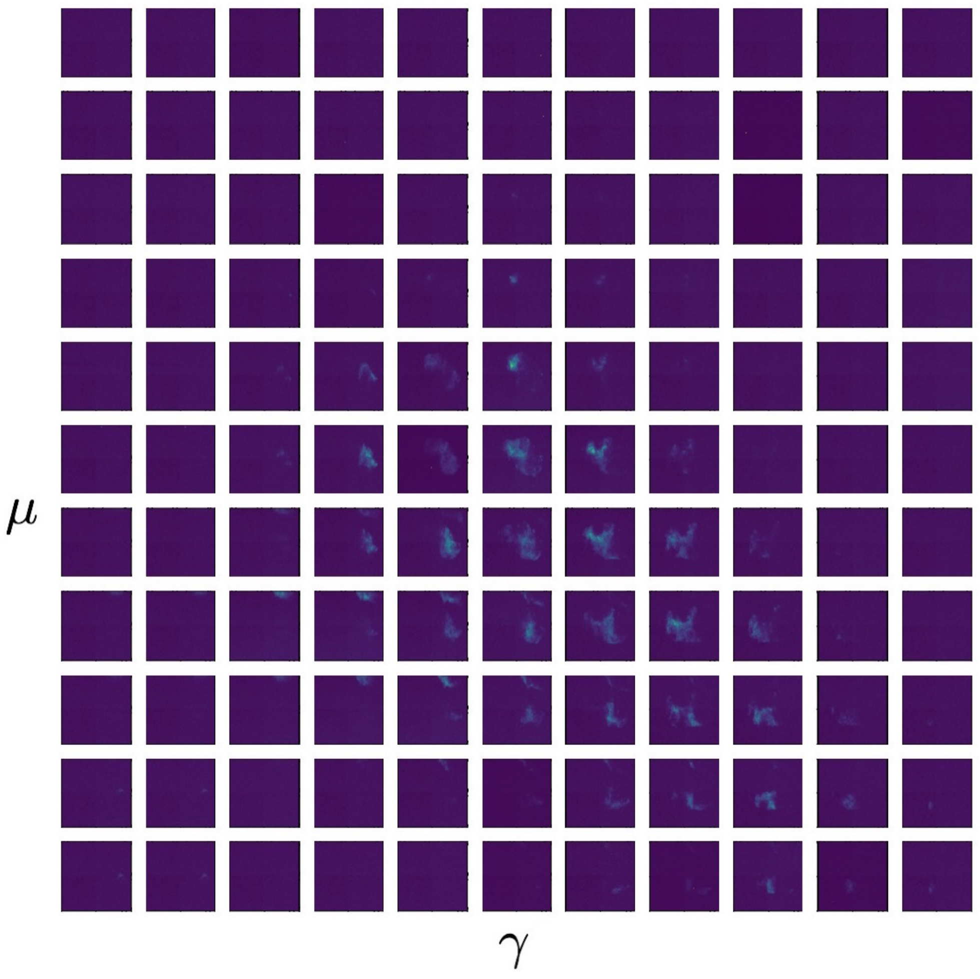

Figure 4: Images collected, at a certain rotation angle ω, by varying the γ and μ angles. For both angles, images were recorded at regular steps with an angular width of $0.0585°$. Sample: deeply embedded grain in a dog-boned Al bar.

$$\begin{aligned}
\phi &= \phi_0 + i \cdot \Delta\phi && (1)\\
&= \phi_0 + i \cdot \cos\omega \cdot \Delta\gamma && (2)\\
\chi &= \chi_0 + i \cdot \Delta\chi && (3)\\
&= \phi_0 + i \cdot \frac{\sin\omega}{\cos\phi_0} \cdot \Delta\gamma && (4)
\end{aligned}$$

From the expression of ϕ, γ can be calculated as

$$\gamma = i\Delta\gamma = \frac{\phi - \phi_0}{\cos\omega} \tag{5}$$

To read the header of the `.edf` images, modules from the FabIO[3] package were used.

3.2 Output

As output, `getdata.py` generates the following files:

- `gamma.npy`, a file listing the values of γ, a pseudo angle (it's a combination of α and β, and it's not controlled by a real motor) perpendicular to μ. α and β are read from the header of the `.edf` files
- `mu.npy`, a file listing all the considered scattering values where frames where collected. Values read from the header of the `.edf` files
- `omega.npy`, a file listing, in degrees, the rotation angles where diffraction signal was collected. Values read from the header of the `.edf` files
- `dataarray.npy`, a file storing, as a five-dimensional matrix, the coordinates of each diffracted intensity value collected during a topotomo scan. The dimensions are: γ angle, μ angle, ω angle, X coordinate, Y coordinate. In other words, `dataarray.npy` stores the raw data as a function of the angles where they were collected
- `cleaning_img.npy`, a file containing, for each considered projection, the frame using to correct for the background current. Dimensions: ω angle, X coordinate, Y coordinate
- `dataarray_clean.npy`, a file structured as `dataarray.npy`, containing the collected frames after they have been normalized by the mean, so that the totatl integrated intensity is the same for each projection
- `dataarray_final.npy`, a file structured as `dataarray.npy`, which contains the frames as they are at the end of the preprocessing procedure
- `Image_properties.txt`, a text file listing the number of each image and the γ, θ and ω angle where it was collected. γ, θ and ω are read from the header of the `.edf` files

3.3 Data loading

3.4 Data cleaning operations

Used nomenclature: *projection* is a sample rotation angle where data was acquired; *frame* is the image collected at a certain set of (ω, γ, μ) angles.

These are the steps followed to clean the collected frames:

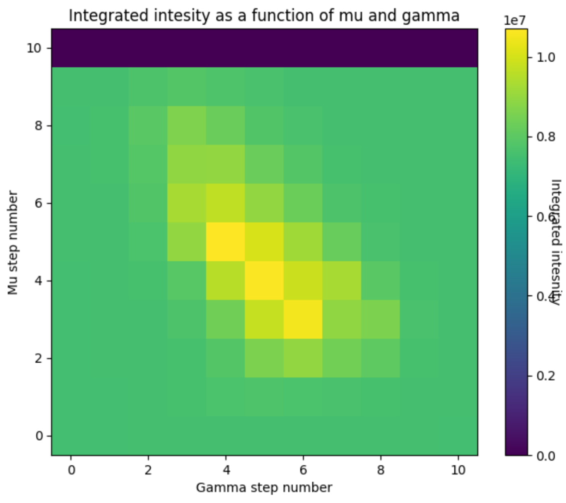

Figure 5: Distribution of the integrated intensity for the frames recorded at a certain projection by varying γ and μ. Each pixel represents a different frame. For both γ and μ, the step size is $0.0585°$. For a given projection, the first step to clean the recorded frames is to subtract, pixel by pixel, from each frame the average of the two images with the lower integrated intensity.

1. For a given projection, subtract from each frame the dark current (detector readout when there is no incoming signal). The spatial distribution of the dark current on the detector plane is estimated as follows (lines 164-205 in `getdata.py`):
   a) Calculate the integrated intensity of each frame (see Fig. 5)
   b) Select the two frames with the lower (but greater than 0) integrated intensities: $I_{min,1}$ and $I_{min,2}$. If there aren't at least two images with nonzero integrated intensity, define the dark current frame as made of zeros
   c) Calculate,pixel by pixel, the average value of $I_{min,1}$ and $I_{min,2}$: $< I_{min}(x,y) >= \frac{I_{min,1}(x,y)+I_{min,2}(x,y)}{2}$. The pixel by pixel operation is designed to take into account possible spatial anisotropies
   d) Subtract the dark current frame ($< I_{min}(x,y) >$) from each frame
   e) Set negative pixels to 0
   f) Threshold hot pixels
2. Normalize the recorded frames, requiring the total integrated intensity (sum of images collected varying γ and μ) to be the same for each projection. In this way, we take into account possible variations of the beam power during the experiment, and

eventual anisotropies in the sample shape. Steps for a given projection (lines 207-221 in `getdata.py`):

a) Sum all recorded frames

b) Calculate the mean pixel intensity

c) Normalize each frame pixel by pixel dividing by the mean intensity of the relative projection, and multiplying by the mean of all mean values, each relative to a different projection (see Fig. 6)

$$I_{normalized}(x,y)_{\gamma,\mu,\omega} = \frac{I(x,y)_{\gamma,\mu,\omega}}{< \sum_{\gamma,\mu}(x,y) >_{\omega}} \cdot << \sum_{\gamma,\mu}(x,y) >_{\omega}> \tag{6}$$

If there are projections where no intensity is recorded, they should be excluded from the calculation of the mean values.

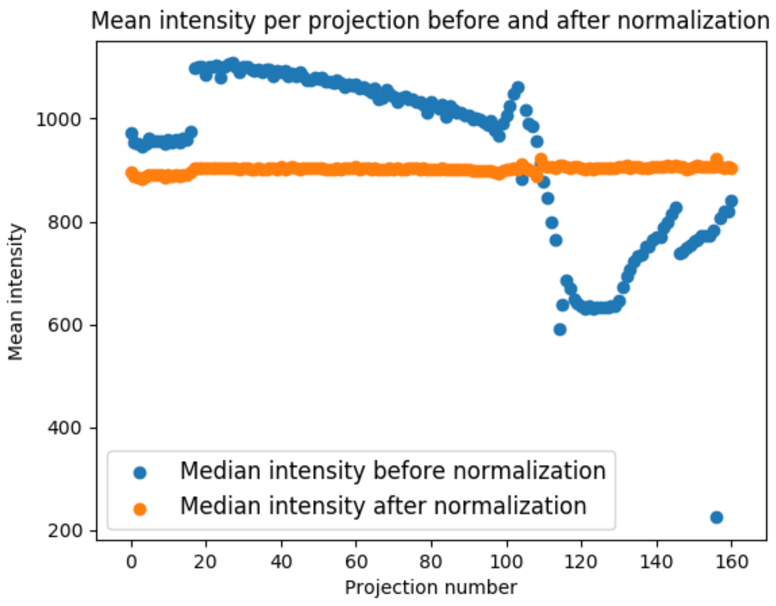

Figure 6: Mean pixel intensity per projection before and after normalization.

3. For each frame collected in a given projection, characterize the noise distribution and subtract it from the recorded intensity (lines 233-273 in `getdata.py`). Goal: minimize the noise and consider possible spatial anisotropies of the detector response. *Wolfgang Ludwig suggested that this should not be necessary if the transfocator is managed correctly*. For a given projection the steps are, as illustrated in Fig. 7:

   a) Rebin the image (the size of the bin is given as an input when launching `getdata.py`)

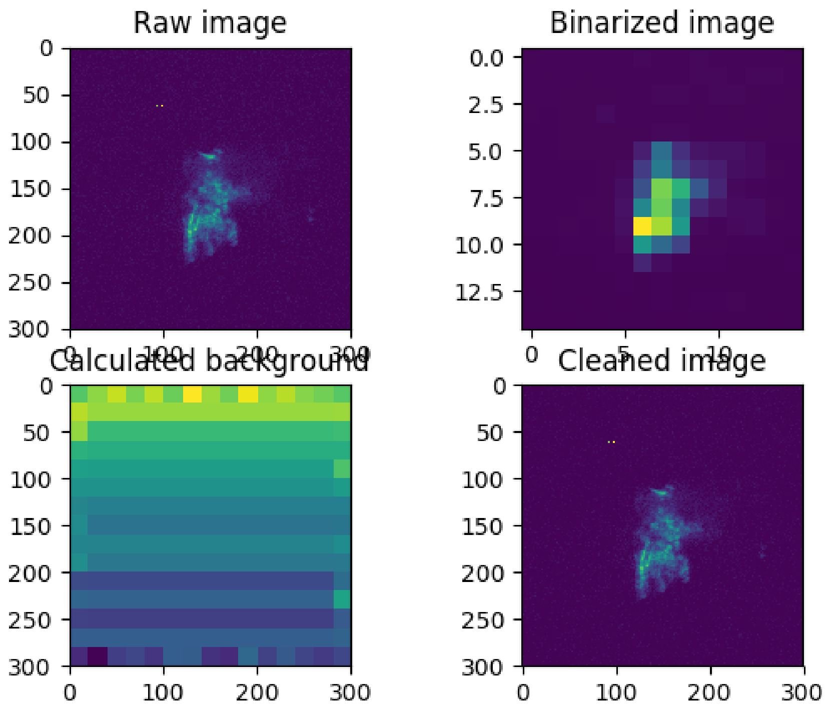

Figure 7: Steps used to clean each image from its background signal. After binning (bin size provided as `getdata.py` input), the intensity values of the bins in the cornice, where no diffraction signal is expected, are used to calculate the expected background distribution for the entire image. The background is then subtracted from the image.

b) Consider the external cornice (first and last column, first and last row), where normally there is no diffraction signal. **This should be changed in case of bad alignment**

c) Notice that the background signal tends to progressively increase/decrease along Y

d) Calculate, for each pixel, the relative expected mean background signal using the values of the top and bottom bin from the same column. For a pixel with coordinates (x, y), $IB_{bin,min}$ and $IB_{bin,max}$ are the maximum and minimum intensity of the background signal, calculated considering the top and bottom bins corresponding to the x column. The intensity of the background signal in (x, y), $IB(x, y)$, is then

$$IB(x,y) = IB_{bin,min} + \frac{IB_{bin,max} - IB_{bin,min}}{sz(frame) - 2 \cdot sz(cornice)} \cdot (y - sz(cornice)) \tag{7}$$

Where $sz(frame)$ and $sz(cornice)$ are the size of the frame and the cornice,

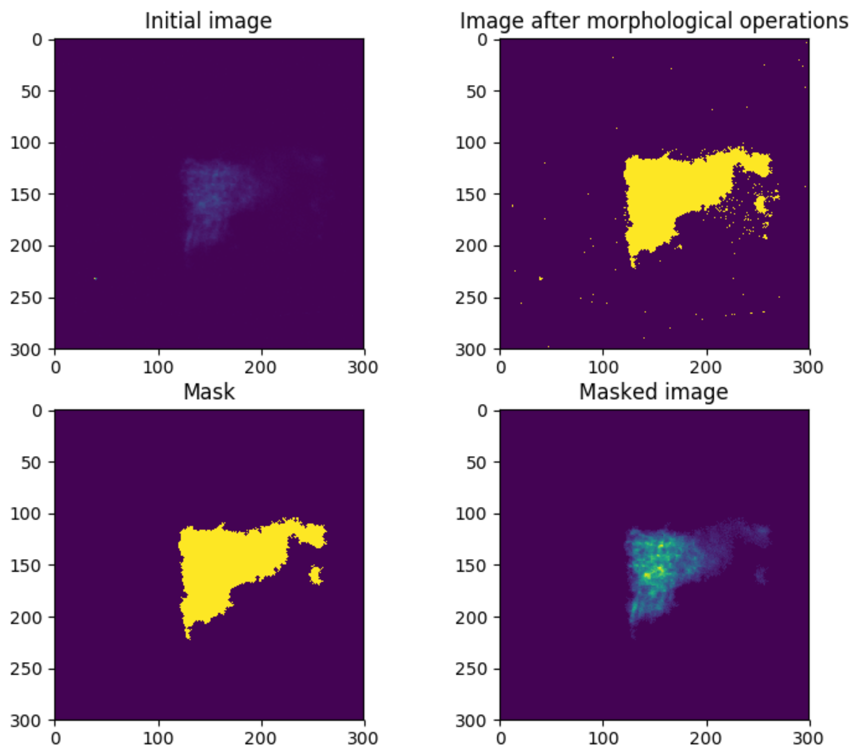

Figure 8: The final step of the developed image processing procedure consists in localizing the diffraction spots, and setting to 0 all pixels that are not part of them. This is done by binarizing the image, performing morphological operations, selecting the larger regions and making a mask using them.

respectively

e) Subtract the calculated background from the frame

f) Set to zero all negative pixels

This procedure can be performed only if the diffraction spot occupies a well defined region at the center of the detector. If that is not the case, set the cornice size to 0.

4. Localize the regions containing the diffraction signal and set to 0 all pixels outside it (see Fig. 8). Lines 274-296 in `getdata.py`. These are the steps in detail:

   a) Threshold the input image (threshold value provided as input to `getdata.py`) and binarize the result

   b) Fill the holes in the image, and perform the morphological operations of erosion and dilation [4]. In Python, a fast and easy way to do so is to use the functions included in skimage and ndimage. Code snippet from `getdata.py`:

```
from skimage.morphology import disk, dilation, erosion
```

```
Cleared = ndimage.binary_fill_holes(IM_clean_bin).astype(int)
Dilated = erosion(dilation(Cleared, disk(1)), disk(1))
Dilated_c = ndimage.binary_fill_holes(Dilated).astype(int)
```

c) Label the regions in the resulting binary image, and select only those with an area larger than 100 pixels. *The minimum number of pixels required to consider a cluster as due to a diffraction spot depends on the noise and the signal distribution for a specific sample. A reasonable value should be determined through tests*

d) Using the selected regions, mask the initial image and set all pixels outside the mask to 0

`getdata.py` also returns an array containing, for each projection, the sum of all collected images. The summed images have been normalized so that they have the same summed intensity. For a given projection i, the summed image was normalized pixel by pixel using the formula

$$I_{i,normalized}(x,y) = \frac{I_i(x,y)}{mean(I_i(x,y))} \cdot max\{mean(I_i(x,y))\}_i \tag{8}$$

4 Recon3d.py

5 Validation of the shape reconstruction procedure

5.1 Validation using ASTRA

5.2 Validation using Art + TV

6 To dos

6.1 Critical

1. In `getdata.py`, include the option to consider the case when the diffraction signal covers the the entire frame, and the cornice methid cannot be used

6.2 Will make life easier

1. Provide paths as input in
   - `Rebin_img.py`
   - `plot_img.py`
   - `plot_angles.py`
2. Install ImageJ and/or FIJI on Panda2, to visualize stack of images without the need to download them

3. Install ParaView on Panda2, to visualize 3D vtk files
4. Combine `img_sum.py` and `plot_sum.py`

6.3 Would be nice to have

1. Adapted version of `fabian` (image visualizer included in FabIO [3]) for visualizing sets of `.edf` files. At the moment, `fabian` only works for list of files numbered incrementally. Usually, the images collected during a topotomo scan are named `file_XXX_YYY.edf`, with `YYY` going from 0 to the number of angular steps (one motor only), and `XXX` increasing by 1 after `YYY` went through all motor values

7 Code contributors

In alphabetic order:

- Alberto Cereser, contributor 2017

  Email: alberto.cereser@gmail.com, alcer@fysik.dtu.dk

  Recon3D repository: https://github.com/albusdemens/Recon3D

- Anders Clemen Jakobsen, contributor 2016/2017

  Email: andcj@fysik.dtu.dk

  Recon3D repository: https://github.com/acjak/Recon3D

References

[1] H. Simons, A. King, W. Ludwig, C. Detlefs, W. Pantleon, S. Schmidt, I. Snigireva, A. Snigirev, and H. F. Poulsen, "Dark-field X-ray microscopy for multiscale structural characterization," *Nature Communications*, vol. 6, 2015.

[2] H. F. Poulsen and et al. *In preparation*, 2017.

[3] E. B. Knudsen, H. O. Sørensen, J. P. Wright, G. Goret, and J. Kieffer, "FabIO: easy access to two-dimensional X-ray detector images in Python," *Journal of Applied Crystallography*, vol. 46, no. 2, pp. 537–539, 2013.

[4] C. J. Russ, *The Image Processing Handbook*. Boca Raton, Florida: CRC Press, 2016.