Wide field synchrotron radiation based x-ray tomographic microscopy @ TOMCAT

David Haberthür

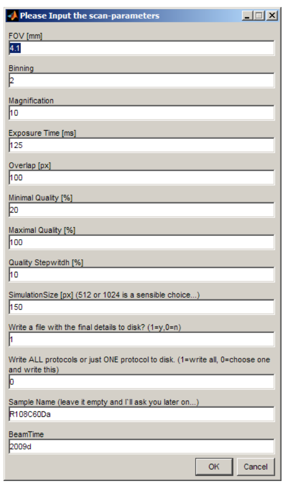
September 15, 2009

This document[1] should give a walk through for successfully performing a high resolution synchrotron radiation based x-ray tomographic microscopy with enhaced lateral field of view (WF-SRXTM, [1]) at TOMCAT, the beamline for TOmographic Microscopy and Coherent rAdiology experimenTs [2]

1. Setup of a WF-SRXTM scan

After starting MATLAB on one of the computing consoles at TOMCAT (start it with entering `matlab &` in the console), open the file ∼`/MATLAB/main.m` in the home directory of our E-Account e11126 (`/sls/X02DA/data/e11126/`).

When you start this file, a user dialog asks you for the parameters for the scan (see figure 1). You should have an idea about the diameter of your sample and some other parameters of your scan to be able to input those into this dialog and get an accurate simulation. Enter the desired field of view in the first line, the parameters of the setup like Binning, Magnification and Exposure Time on the second, third and fourth line.



FOV [mm]
4.1

Binning
2

Magnification
10

Exposure Time [ms]
125

Overlap [px]
100

Minimal Quality [%]
20

Maximal Quality [%]
100

Quality Stepwitdh [%]
10

SimulationSize [px] (512 or 1024 is a sensible choice...)
150

Write a file with the final details to disk? (1=y,0=n)
1

Write ALL protocols or just ONE protocol to disk. (1=write all, 0=choose one and write this)
0

Sample Name (leave it empty and I'll ask you later on...)
R108C60Da

BeamTime
2009d

Figure 1: User dialog

The Overlap can be left around 100 px or changed to a value you'd like. The following three parameters (Minimal and Maximal Quality (Q_{min}, Q_{max}) as well as Quality Stepwidth (Q_{step}) influence the calculation of the different protocols. Choose a broad range (like the defaults) for a good overview over the different possible protocols, choose a small range if you want to have a detailed choice over the different protocols.

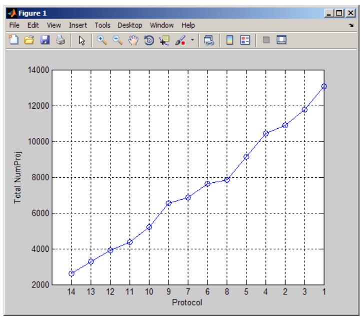
The SimulationSize greatly influences the speed and accuracy of the calculation. If you enter a small size, the different protocols are calculated for a small Shepp–Logan phantom and the whole simulation is done quickly, but not very accurate. If you want to scan all multiple

[1]A recent version is available at `http://www.ana.unibe.ch/~haberthuer/manuals`

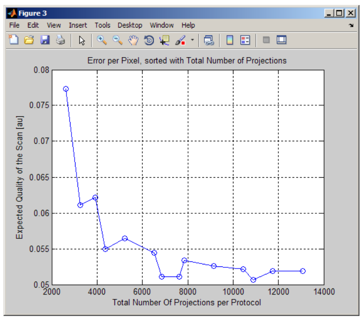
protocols between Q_{min} and Q_{max} and not only choose one of the protocols, you can safely leave this parameter on a small size, since the calculated number of Projections for each protocol does not depend on it. If you want to choose one protocol depending on the quality, you should set this parameter fairly high. This will increase the simulation time, but you'll get a more accurate estimation of the scanning quality.

The following fields are important for the writing of the parameter file, which are needed for in the next step, the actual scan. If you want to write a preference file, put "1" in the 10th field. This will generate a text file with the "Sample Name" you choose in the second to last field in the folder `/sls/X02DA/Data3/e11126/BeamTime`, where "BeamTime" is what you enter in the last field.

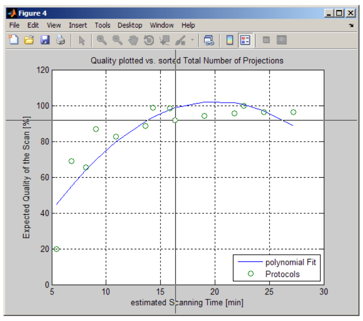
The MATLAB script will output several figures, some of them are shown in figure 2.



(a) Total Amount of Projections per Protocol



(b) Error per pixel. x-axis shows the total amount of projections

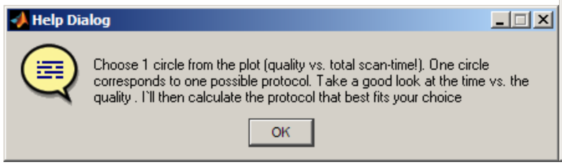


(c) Quality of the scan plotted vs. scanning time needed (or vs. protocols)

Figure 2: Output from main.m script. a) Total amount of projections per protocol. This is the total amount of projections scanned for all subscans. b) Error of the difference image to the phantom compared to the total amount of projections. You can see a sharp increase in error for low amount of projections. c) Quality of the scan vs. scanning time, including a polynomial fit (only to guide the eye). The scanning time is estimates using the total amount of projections $\times$ the exposure time. This is where the user actually chooses a protocol (shown as cross at approx. 16 min scanning time).

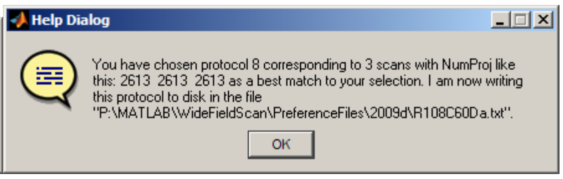
If you choose to scan only one protocol, you've entered "0" in the 11th input field, the script asks you to choose one of the protocols in the quality plot shown in figure 2c and then outputs a preference-file for one scan (see figures 3a and 3b). If you entered "1", you will not be asked to choose a protocol and the script outputs all possible protocols from Q_{min} to Q_{max} with a step width of Q_{step} to a preference file (see figure 3c). An example of a preference file for one protocol is shown in appendix B.4.

After you get this preference file, you can proceed to the next step.



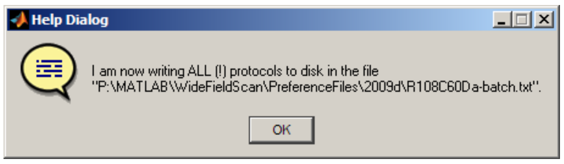
Choose 1 circle from the plot (quality vs. total scan-time!). One circle corresponds to one possible protocol. Take a good look at the time vs. the quality . I`ll then calculate the protocol that best fits your choice

(a) Choose one protocol



You have chosen protocol 8 corresponding to 3 scans with NumProj like this: 2613 2613 2613 as a best match to your selection. I am now writing this protocol to disk in the file "P:\MATLAB\WideFieldScan\PreferenceFiles\2009d\R108C60Da.txt".

(b) Output for one protocol



I am now writing ALL (!) protocols to disk in the file "P:\MATLAB\WideFieldScan\PreferenceFiles\2009d\R108C60Da-batch.txt".

(c) Output for all protocols

Figure 3: Output Options

2. Scanning a sample with increased field of view

To perform a wide field scan at TOMCAT, you need to adjust the sample in the beam, as you would do with a conventional scan. Make sure that you've aligned the sample in such a way that the feature you want to have in the center of the resulting wide field scan reconstructions is in the center of the preview window before proceeding (see the wiki-page "perform_run" for a bit of details on how to align the sample). When you've aligned the sample, enter the "SampleName" into the corresponding field in the control panel of TOMCAT (on the machine with three screen).

Copy `widefieldscan_final.py` from ~`/MATLAB/` to the current working directory (`/sls/X02DA/Data3/e11126/Beamtime`). Start the Terminal of the machine next to the one with the three screens and "cd" to the directory of the BeamTime. Start the wide field scan with `widefieldscan_fila.py PreferenceFile` in the Terminal in the correct directory. "PreferenceFile" is the file that MATLAB has written after it has calculated all the protocols and you've selected one. If you've entered the parameters exactly like in figure 1, the preference file you'll get is for one protocol and will be written to `/sls/X02DA/Data3/e11126/2009d/R108C60Da.txt`.

After pressing "Enter", the script reads the SampleName from the EPICS-panel (you did set it correctly before, did you?), sets all other necessary parameters (Number of projections, sample position, start and stop angle, etc.), waits for 10 s, and performs all necessary subscans with the correct amount of projections at the correct positions. The only thing you need to do is wait and maybe keep an eye on the console output, which informs you how things are progressing. The projections of all subscans are saved in the directories `/sls/.../Beamtime/SampleName_si`), where s_i is a directory for each of the three or five (or seven, but let's leave it below that) subscans.

3. Stitching of the projections of the partial scans into merged projections

When the scan is finished, you have to stitch the projections of the several subscans to merged projections covering the chosen field of view. There's a MATLAB script in ∼`/MATLAB/MergeProjections/fct_mergeSubScansInterpolatedSelector.m` which performs such a merging.

This function needs "AmountOfSubScans", "NumDarks", "NumFlats", "Tiff", "OutputSampleName", "OutputSuffix" as input parameters and asks you to select the correct subscans to merge, performs the merging and writes the files to `/sls/.../Beamtime/mrg/OutputSampleName-OutputSuffix-mrg`).

The details of the parameters are:

AmountOfSubScans Amount of subscans which you've scanned to cover the field of view (s_1–s_3 $\rightarrow$ 3 subscans)

NumDarks The number of dark images you've acquired prior to the scan

NumFlats The number of flat images you've acquired prior and after the scan. Both are needed to correct the projections. It's much easier to input them here than to parse them from the log-file, so I've chosen this way. If you don't know it, look at the log file of the firs subscan, and there you got everything.

Tiff set this to "1" if you want .tif files as output. If you set it to "0", the script writes .DMP to disk (for historic reasons. . .)

OutputSampleName It might be necessary to restart a batch-scan after a beam dump, but you want to have all scans to have the same Name, so you can set this name here.

OutputSuffix If you performed a batch scan of the same sample with different parameters, you can add an suffix to the Output, which then makes the protocols/parameters distinguishable in the output files.

If you've entered everything correctly, MATLAB asks you about the location of the directories of each subscan, counts the files in the directories (again, much easier than log-file parsing), then reads the Darks and Flats to correct the projections. To set the correct gray values in the output files, MATLAB reads a subset of the projections of each subscan (the amount (`GrayValueLoadHowMany`)can be changed around line 160 of `fct_mergeSubScansInterpolatedSelector.m`, at the moment it's 100 randomly loaded projections of each subscan). Afterwards, the scripts calculates the Cutline using Chris' `function_cutline.m`, merges the projections of the individual subscans together and writes a corrected and merged projection to the output path as described above. Additionally, MATLAB generates a faked log-file in the correct spot, and starts the generation of the sinograms with a call to `sinooff_tomcat_j.py` in the correct directory. Since the RecoManager (used for reconstruction) expects the (fake) .log-file in a certain spot, the script also hard-links the log-files.

The whole thing takes a while, so you can start the other scans while performing the merging. To be able to generate sinograms, you'll have to use MATLAB on 'x02da-cons-2', or else it won't work.

To be able to consistently perform the merging and choose a sane way of setting the OutputName I generally prefer to cobble together a little script which sets all parameters and calls the merging function, so I don't have to think too much. It's probably easiest if you open one of the old scripts (`do_mergeSubScan2009c.m`) in MATLAB, adjust the parameters at the beginning, save it as a new file and then run this updated script, which calls the function with the correct parameters. With this you can easily keep track of which scans you've already merged.

4. Reconstructing a WF-SRXTM scan

4.1. Sinograms

To reconstruct the merged projections, we need the Sinograms, the RecoManager and the command-line on "x02da-cons-2", so either sit on that machine or make an ssh-connection into the machine with `ssh e11126@x02da-cons-2`.

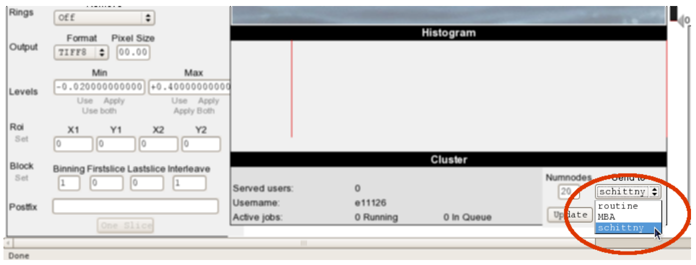
The sinograms of the merged projections should have been generated by the MATLAB script in the step before. If that didn't happen, then you can generate the sinograms with issuing the command

```
/work/sls/bin/sinooff_tomcat_j.py␣/sls/X02DA/Data3/e11126/
    Beamtime/mrg/SampleName_mrg/tif
```

in the terminal (␣ denotes when you need to enter a space). It's probably the best if you issue `pwd` in the directory with the projections, which prints the current path and you can then just copy-paste it. This will generate a bunch of sinograms in `/sls/.../BeamTime/mrg/SampleName_mrg/sin`. But again, MATLAB should have taken care of it.

4.2. RecoManager

Start the RecoManager in Firefox (`http://x02da-reco-1`) and check and note the necessary parameters for the reconstruction (Rotation center as well as gray-value scale). The gray-value scale needs to be adjusted, so that all the grey values are contained inside the minimum and maximum. Use the sliders on the right of the reconstructed slice to change them. Note the values for the sample you want to reconstruct, you'll need them later. It seems like you'd be able to reconstruct directly from the RecoManager, since the "Send to"-menu has an entry for the "Schittny"-cluster (see figure 4), but that does not work correctly for wide field scans (yet). So: DO NOT SUBMIT WITH RECOMANAGER!


Rings: Off
Output: Format TIFF8, Pixel Size 00.00
Levels: Min -0.020000000000, Max +0.40000000000
Roi: X1 0, Y1 0, X2 0, Y2 0
Block: Binning 1, Firstslice 0, Lastslice 0, Interleave 1
Postfix:

Cluster
Served users: 0
Username: e11126
Active jobs: 0 Running 0 In Queue
Numnodes: 20
Send to: schittny

Figure 4: Server entry for our two cluster nodes

4.3. Reconstruction on the Cluster

You've noted the rotation center (RotCtr) and both the minimal and maximal gray value (G_{min} and G_{max}), you'll need them both now. Open the console and enter

```
tif2rec_batch_web_grid_cn_david.py␣3␣20␣/sls/X02DA/Data3/e11126
    /Beamtime/mrg/SampleName−mrg/tif␣RotCtr−1␣4,10,0.5␣0␣0,0,0,0
    ␣0,0.3,10␣8,Gmin,Gmax,0.0␣"suffix"␣1,0,0␣1
```

and you'll get the reconstructed slices after a short waiting time (again: ␣ =space). They will be written into `/sls/.../BeamTime/mrg/SampleName_mrg/rec_8bit_suffix`.

tif2rec_batch_web_grid_cn_david.py Is the Python-script that was made by Fede. This is needed to reconstruct the projections with the correct size. If you use the RecoManager or another script the size of the reconstructions will be $2^n \times 2^n$ and not the correct size.

3 20 Tells the script to use our two cluster nodes (if we're having regular beamtime, substitute 3 with 1 (?)), and to use all available nodes (20).

/sls.../tif Is the full path to the merged projections. Inside this path is the log-file which is parsed by the script (that's the reason we've generated a fake log-file with MATLAB). It's probably the best if you issue `pwd` in the directory with the projections, which prints the current path and you can then just copy-paste it.

RotCtr-1 Is the rotation center you've checked in the RecoManager in the step before. Remember that for using `gridrec` you need to subtract the rotation center by one. Enter the rotation center with the decimal places, even if there are none ("1234.00").

4,10,0.5 Filter and Filter parameters

0 0,0,0,0 Rotation and Region of Interest, generally left at zero

0,0.3,10 Ring removal filter. Generally, we don't need it. If rings are very prominent, then swap the "0" for a "1"

8,G_{min},G_{min},0.0 Reconstructions with a bit depth of 8=8 bit (or "16" for 16 bit) scaled to the gray values G_{min} and G_{min}. The last parameter "0.0" is used, if the files should be reconstructed into .ISQ-files for the Scanco-Workstation. We just leave it as "0.0".

"suffix" the suffix to add to the output directory (rec_8bit_suffix), nice if you want to test different things. Leave empty ("") if you don't want to have a suffix.

1,0,0 The first parameter can be used to reconstruct only the x^{th} sinogram (we want each and every sinogram, thus "1"). The "0,0" can be used to only reconstruct a region of the slices, e.g. if a ROI should be reconstructed; "0,0" reconstructs the full slices.

1 "1" is the binning which can be set in addition to the binning of the camera.

It's probably easiest for you if you copy the command into a text file so you can make sure you've got no typo and then enter it on the command line. Wait a bit and enjoy your reconstructions of a WF-SRXTM scan.

References

[1] David Haberthür, Christoph Hintermüller, Johannes C. Schittny, and Marco Stampanoni. Quality-driven expansion of the field of view in synchrotron radiation x-ray tomographic microscopy. *Journal of Synchrotron Radiation*, 2009. doi: 10.1107/. URL `http://dx.doi.org/10.1107/`. in preparation.

[2] M. Stampanoni, A. Groso, A. Isenegger, G. Mikuljan, Q. Chen, D. Meister, M. Lange, R. Betemps, S. Henein, and R. Abela. TOMCAT: A beamline for TOmographic Microscopy and Coherent rAdiology experimenTs. *AIP Conference Proceedings*, 879 (1):848–851, 2007. doi: 10.1063/1.2436193. URL `http://link.aip.org/link/?APC/879/848/1`.

[3] Christoph Hintermüller, Federica Marone, Andreas Isenegger, and Marco Stampanoni. Image processing pipeline for fast Synchrotron based X-Ray Micro-Tomographic Microscopy. *Journal of Synchrotron Radiation*, 2009. submitted.

A. Appendix - Detailed Explanation of all involved MATLAB files

Explanation of the several involved files, for documentation and "working n"-purpose. All necessary files to perform the steps mentioned above are under version control with subversion at `http://code.ana.unibe.ch/` in the "wfs-sim" repository (the repository is called "wfs-sim", because it first contained only the files for the simulation. Now it included everything to perform a wide field scan).

Most of the files are functions which are called to perform several sub-tasks of the `main.m`-file, and are denoted with a `fct_`-prefix. A lot of files in the directory are used for testing the functions. Those testing files are denoted with a `test_`-prefix.

B. Setup of the Scan - main.m

`main.m` is the main MATLAB-File that is used for the simulation. The code should be fairly commented, nonetheless I'm explaining it here in a bit more details.

The user is first asked for some input parameters (see figure 3) which are input through an `InputDialog`, which is seeded with some sensible defaults. The desired FOV (`FOV_mm`) is probably the most important factor in the whole calculation, since it influences the `AmountOfSubScans`. `Binning`, `Magnification` and chosen Overlap between SubScans (`Overlap_px`) are used to calculate the number of subscans needed. Using the Binning, we can calculate the `DetectorWidth`, since the camera is 2048 pixels wide and we assume that the full sensor is used. The `pixelsize` can be calculated using the Magnification, according to the TOMCAT website. The DetectorWidth and the Overlap define the width of one subscan or segment (`SegmentWidth_px`) and thus the `AmountOfSubScans`. If the sample does not perfectly fit in the diameter of AmountOfSubScans $\times$ SegmentWidth, we're actually "loosing" imaging real estate, and we inform the user. The actual FOV (`ActualFOV_px`) is used for the calculation of the correct number of projections.

After we make sure that we have an odd amount of SubScans, we use the function `fct_SegmentGenerator` to calculate the `NumberOfProjections` from the inputs. The resulting table is X amounts of SubScans wide and Y protocols tall.

The user specifies the image size that should be used for the calculation of the simulated error (`SimulationSize_px`). Since the radon and inverse radon transform is quite processor-intensive we simulate with a model where the images and reconstructions are of a smaller size. While making sure that the overlap for this model still stays above 5 pixels, we calculate the reduction factor according to the user input (`ModelReductionFactor`) and scale the table with the number of projections, the phantom image (`ModelImage`) and the detector width (`ModelDetectorWidth`) with this reduction factor. After we have calculated a full size sinogram and a full size reconstruction (`ModelMaximalSinogram` and `ModelMaximalReconstruction`), we use the function `fct_ErrorCalculation` for details on this function) to calculate the error of each protocol.

This error is kept track of in `AbsoluteError` and `ErrorPerPixel(Protocol)`, once as absolute and once normalized to the total amount of pixels in the image. After we've sorted the number of projections and summed the amount of projections for each protocol (`TotalSubScans`), we plot this value versus the `QualityMeasure`. The Quality is calculated in such a way that we subtract the maximum of the Error per Pixel from all the Errors per Pixel, so we get a high value for low Errors and vice versa. After we've scaled this value to the minimal and maximal quality the user has input, we provide the user a mean of choosing the protocol, that suits his needs in terms of quality and scanning time.

The chosen value is saved into `User_NumProj` and subsequently written to disk to a file `UserSampleName`.txt which contains the details of the chosen Protocol, including `InBeamPosition` and the angles of the rotation stage (which are currently hardcoded as `RotationStartAngle`$=0°$ and `RotationStopAngle`$=180°$, but can be easily changed on line 325 and 326 of the `main.m`-file.

B.1. fct_ProtocolGenerator

This function calculates and outputs a matrix of Projection Numbers after it gets the SampleWidth, the Amount of Subscans and the parameters of the Quality (Q_{min}, Q_{max} and Q_{step}). After checking for a cutoff quality of 30% (and setting it lower if the user insists), the function calculates the minimal number of images needed to fulfill the sampling theorem and calls the function `fct_GenerateSegments` to generate the projection numbers for the different subscans.

B.1.1. fct_GenerateSegments

This function internally calls itself again and generates the matrix of number of projections while dividing the number of images by two (which decreases the amount of projections for the central scan (or central scans if the amount of subscans is bigger than 3).

B.2. fct_HowLongDoesItTake

We'd like to inform the user about an estimate on how long the whole scan (with 3, 5 or more subscans) will take, so he or she has an estimate on how many scans can be squeezed into the beamtime. The time needed obviously depends on the total amount of projections. The news stepping technique implemented at TOMCAT uses a hardware-trigger for the stage and includes a wait for 200ms (`TriggerTime`). Since with our setup, we always rotate for 180° and the stage rotates with a speed of 90°/s, the time needed to record 1 projection can estimated with $t_{proj} = \frac{1}{90}180° \frac{1}{NumProj}$. Since—according to Fede—the camera has a readout time of 451ms, we set this as time needed for one projections if the Exposuretime plus TriggerTime plus t_{proj} are smaller than this readout time. Additionally, we add 1min for moving the sample between each subscan. The output of this function is the `TotalTime` needed for one protocol in minutes.

B.3. fct_ErrorCalculation.m

This function uses an input image as baseline image (a Shepp–Logan phantom with added Gaussian noise), the number of projections of each subscan and the maximal reconstruction (`radon`-transformation and `iradon`-reconstruction of the phantom) as an input. Inside the function we split the image into different parts (according to the amount of subscans). According to the different amount of projections from the outer to the inner scans we interpolate rows of these parts from the sinogram of the maximal image using another function (`fct_InterpolateImage`). This corresponds to a different amount of recorded projections, since these are "encoded" in the rows of the sinogram. Since MATLAB calculates the sinogram 90° rotated to what we know and like, the sinogram is internally flipped (thus we get the "interpolating in horizontal direction" in the command-line output of MATLAB) when the numbers of projections for the scans need to be interpolated. The interpolated sinogram is reconstructed and is shown in together with the difference image if the user requests it with the input `ShowFigure`.

As an error-measure we calculate the absolute image difference and take the sum over each pixel—$\sum_{horiz.}(\sum_{vert.}(\text{DiffImg}))$—of the difference image of the resulting reconstruction and the model image. The difference image is calculated with the MATLAB-function `imabsdiff` which calculates the absolute difference between two images, so we don't have to take the square of this error.

The output of the function is the absolute error `AbsoluteError` and the error per pixel `ErrorPerPixel` which is the absolute error divided by the amount of pixels in the image.

B.3.1. fct_InterpolateImageRows

Since I'm very bad at remembering how to interpolate in MATLAB I've written a small function which takes an input image `InputImage`, a parameter which describes every xth line that should be interpolated (`InterpolateEveryXthLnie`). The output is an image which has every xth line interpolated. This function is used to generate "merged" sinograms with reduced amount of projections in the central part of the sample, which is needed for the function `fct_ErrorCalculation.m` specified above. An optional parameter ("1") tells the function to flip the input image, since MATLAB generates the sinograms 90° rotated to what we know (and is generally set).

B.4. Example of a preference file

Below is an example of a preference file which is generated as output by the `main.m`-MATLAB file and contains all parameters for one wide field scan. Such a .txt-file is then subsequently read by the `widefieldscan_final.py`-file and used to set the parameters in the EPICS-panel at TOMCAT.

```
# Path = P:\MATLAB\WideFieldScan\PreferenceFiles\2009d
# SampleName = R108C60Da
# chosen FOV = 4.1 mm
# chosen FOV = 2770 pixels
# actual FOV = 2772 pixels
# DetectorWidth = 1024 pixels
# Exposure Time = 125 ms per Projection
# Magnification = 10 x
# Binning = 2 x 2
# Overlap = 100 pixels
# Chosen Protocol = Nr. 8
#---
# NumProj InBeamPosition StartAngle StopAngle
#--- Protocol 8/7839 total Proj./62 minutes ---#
2613 -1367.5 0 180
2613 0 0 180
2613 1367.5 0 180
```

C. Merge Subscans - /MergeProjection/fct_-mergeSubScansInterpolatedSelector.m

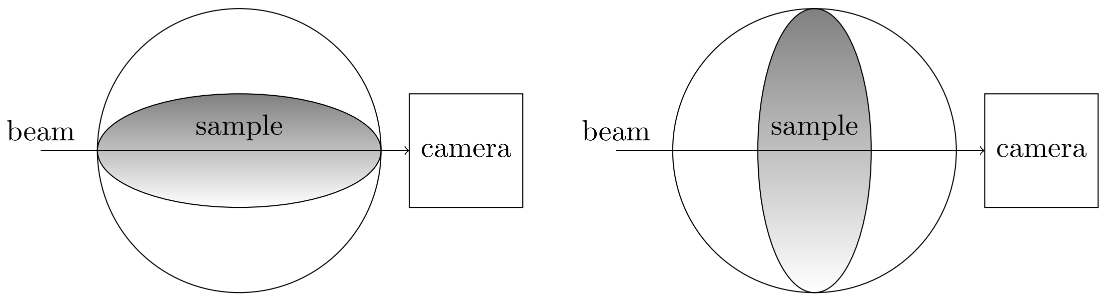
This function is a further development of the function (`fct_mergeSubScansSelector.m`) which asks the user to point to the directories of the subscans and merges them. This developed version does interpolate the necessary projections and does not only dumbly re-use the existing projection at each point. The demanded input parameters are `AmountOfSubScans`, Number of Darks and Flats, `Tiff`, `BeamTime`, `OutputSampleName` and `OutputSuffix`.

The amount of subscans is used to know how many projections need to be stitched together. We ask for the input of the number of dark and flat images since parsing the log-files has been shown to be too complicated, so entering it is straight-forward. The option `Tiff` is mainly for historic reasons. If it's set to "1", the output are .tiff files, if it's set to "0", the output are .DMP-files. The `BeamTime` is used to construct the correct directory for reading the projections. The `OutputSampleName` is used to override the SampleName, which can come in handy if there was a beam dump at TOMCAT and you had to change the SampleName, but want to have a common output name to all samples. The `OutputSuffix` can be used to distinguish several protocols with the same output name.

The script asks the user to input the directories of the subscans (so arbitrary scans could be stitched together, thus the script is very flexible), counts the amount of projections in the directories `(CurrentSubScan).NumProj` to get a grasp of the number of projections (again, the script is flexible and could be used with a "wrong" number of projections). With this number of projections we calculate a modulo of projections of the different subscans, so we know how many projections need to be interpolated to be able to correctly stitch the projections.

Since MATLAB handles gray values of .tiff-files differently than the beamline scripts at TOMCAT, we load a random subset of images (`GrayValueLoadHowMany` projections from each subscan) and look for a global minimum and maximum of the gray value in this subset of projections (`GlobalMin` and `GlobalMax`). These values are padded with a safety margin and used to scale the merged projections later on in the script.

We load a projection image of each subscan and calculate the cutline where the files are to be merged using Chris' function `function_cutline`. Generally, we orient the sample that the longest side of the lung tissue is parallel to the beam. The cutline function needs features in the projection images to be able to calculate a cutline, so we choose a projection in the middle of the set, where the sample is oriented perpendicular to the beam. If the sample is oriented perpendicular to the beam, we would need to calculate the cutline from the first projections. This made it necessary to introduce an internal option `perpendicular` for choosing the correct projections. Since we generally orient the sample parallel to the beam, this option is only accessible internally from the file, but can be easily changed around line 217. Figure 5 shows how the parameter should be set according to the position of the sample at the start of the scan. If you see that the projections from the subscans are stitched wrongly, or the command-line prints "cutline is 0 pixels" but you know that there is an overlap, then this is something to double-check.



(a) perpendicular=0, standard (b) change option to "perpendicular=1"

Figure 5: The position of the sample at the start of the scan influences the "perpendicular" option.

After the detection of the cutline, the script loads the projections from each subscan, interpolates projections with `fct_ImageInterpolator` if necessary, merges them to big projections with the correct cutline and writes those .tiff-filed to disk in the directory `/sls/.../BeamTime/mrg/OutputSampleName/tif`.

After all merged projections have been written to disk, the MATLAB script generates a log-file (`/sls/.../BeamTime/mrg/OutputSampleName/tif/OutputSampleName.log`) using `dlmwrite`, which just writes a text file which looks exactly the same as the log-files generated by TOMCAT. This log-file is then hard-linked to `/sls/.../BeamTime/mrg/log/OutputSampleName.log`, where it's kept for future access.

With a command-line call to `/sinooff_tomcat_j.py` the script automatically generates the sinograms, so that further processing in the RecoManager can be done without issuing another command.

C.1. function_cutline

This is a MATLAB script that has been made by Christoph Hintermüller, which takes two images as input and outputs the pixel-position at which those two overlap. It extracts a similarity of the images while the images are slid over each other, using the mean quadratic difference. The function is described in Chris' paper [3].

C.2. fct_ImageInterpolator

This function takes two Input Images (`ImageToInterpolate1, ,ImageToInterpolate2`) and a factor of how many Images should be interpolated between the two Input Images (`InterpolateHowManyInbetween`). This factor influences the grid distance of the `meshgrid`, which is used to calculate the interpolation. The output of the function is a stack of images with the first and the last slice set as the two original input images filled with the interpolated images. The resulting output stack has a size of (x,y,`InterpolateHowManyInbetween`+2) (where x and y are the size of the two input-images, which obviously need to be of the same size).