

---

# **STP-Core Documentation**

*Release 0.0.1*

**Sincrotrone Trieste S.C.p.A.**

**Sep 06, 2017**



---

## Contents

---

<b>1 Features</b>	<b>3</b>
<b>2 Contribute</b>	<b>5</b>
<b>3 Content</b>	<b>7</b>
<b>Bibliography</b>	<b>129</b>
<b>Python Module Index</b>	<b>131</b>



STP Core provides .....

Here is how to add a link to your documentation [Docs](#) and here is how to add a reference *[AI]*



# CHAPTER 1

---

## Features

---

- Example of how to write documentation





## CHAPTER 2

---

### Contribute

---

- Documentation: <https://github.com/decarlof/pore3d/tree/master/doc>
- Issue Tracker: <https://github.com/decarlof/pore3d/docs/issues>
- Source Code: <https://github.com/decarlof/pore3d/>



### About

This section describes what the [STP Core](#) project is about ...

### Install

This section covers the basics of how to download and install [STP Core](#)

#### Contents:

- *Installing from source*

### Installing from source

Clone the [STP Core](#) from [GitHub](#) repository:

```
git clone https://github.com/ElettraSciComp/STP-Core.git STP-Core
```

then:

```
cd STP-Core  
python setup.py install
```

### API reference

**project Modules:****io.tdf****Functions:**

<code>parse_metadata(f, xml_command)</code>	Fill the specified HDF5 file with metadata according to the DataExchange initiative.
<code>read_tomo(dataset, index)</code>	Extract the tomographic projection at the specified relative index from the HDF5 dataset.
<code>parse_metadata(f, xml_command)</code>	Fill the specified HDF5 file with metadata according to the DataExchange initiative.
<code>read_sino(dataset, index)</code>	Extract the sinogram at the specified relative index from the HDF5 dataset.
<code>write_tomo(dataset, index, im)</code>	Modify the tomographic projection at the specified relative index from the HDF5 dataset with the image passed as input.
<code>write_sino(dataset, index, im)</code>	Modify the sinogram at the specified relative index from the HDF5 dataset with the image passed as input.
<code>get_nr_projs(dataset)</code>	Get the number of projections of the input dataset.
<code>get_nr_sinos(dataset)</code>	Get the number of sinograms (or slices) of the input dataset.
<code>get_det_size(dataset)</code>	Get the width of the detector (nr of pixels) of the input dataset.
<code>get_dset_shape(det_size, fov_height, nr_proj)</code>	Get the shape of the dataset by arranging the input parameters.
<code>get_dset_chunks(det_size)</code>	Get a good chunk combination.

`stp_core.io.tdf.get_det_size(dataset)`

Get the width of the detector (nr of pixels) of the input dataset.

**Parameters** `dataset` (*HDF5 dataset*) – HDF5 dataset as returned by the h5py API.

`stp_core.io.tdf.get_dset_chunks(det_size)`

Get a good chunk combination. This function needs improvement...

**Parameters** `det_size` (*int*) – Width of the detector.

`stp_core.io.tdf.get_dset_shape(det_size, fov_height, nr_proj)`

Get the shape of the dataset by arranging the input parameters.

**Parameters**

- `det_size` (*int*) – Width of the detector.
- `fov_height` (*int*) – Height of the FOV, i.e. the number of sinograms (or slices) of the dataset.
- `nr_proj` (*int*) – Number of collected projections.

`stp_core.io.tdf.get_nr_projs(dataset)`

Get the number of projections of the input dataset.

**Parameters** `dataset` (*HDF5 dataset*) – HDF5 dataset as returned by the h5py API.

`stp_core.io.tdf.get_nr_sinos(dataset)`

Get the number of sinograms (or slices) of the input dataset.

**Parameters** `dataset` (*HDF5 dataset*) – HDF5 dataset as returned by the h5py API.

`stp_core.io.tdf.parse_metadata(f, xml_command)`

Fill the specified HDF5 file with metadata according to the DataExchange initiative. The metadata in input are described in a XML format.

**Parameters**

- **f** (*HDF5 file*) – HDF5 file open with h5py API
- **xml\_command** (*string*) – Imaginary part of the complex X-ray refraction index.

`stp_core.io.tdf.read_sino(dataset, index)`

Extract the sinogram at the specified relative index from the HDF5 dataset.

**Parameters**

- **dataset** (*HDF5 dataset*) – HDF5 dataset as returned by the h5py API.
- **index** (*int*) – Relative position of the sinogram within the dataset.

`stp_core.io.tdf.read_tomo(dataset, index)`

Extract the tomographic projection at the specified relative index from the HDF5 dataset.

**Parameters**

- **dataset** (*HDF5 dataset*) – HDF5 dataset as returned by the h5py API.
- **index** (*int*) – Relative position of the tomographic projection within the dataset.

`stp_core.io.tdf.write_sino(dataset, index, im)`

Modify the sinogram at the specified relative index from the HDF5 dataset with the image passed as input.

**Parameters**

- **dataset** (*HDF5 dataset*) – HDF5 dataset as returned by the h5py API.
- **index** (*int*) – Relative position of the sinogram within the dataset.
- **im** (*array\_like*) – Image data as numpy array.

`stp_core.io.tdf.write_tomo(dataset, index, im)`

Modify the tomographic projection at the specified relative index from the HDF5 dataset with the image passed as input.

**Parameters**

- **dataset** (*HDF5 dataset*) – HDF5 dataset as returned by the h5py API.
- **index** (*int*) – Relative position of the tomographic projection within the dataset.
- **im** (*array\_like*) – Image data as numpy array.

## phaseretrieval.tiehom

### Functions:

<code>tiehom_plan(im, beta, delta, energy, ...)</code>	Pre-compute data to save time in further execution of phase_retrieval with TIE-HOM (Paganin's) algorithm.
<code>tiehom(im, plan[, nr_threads])</code>	Process a tomographic projection image with the TIE-HOM (Paganin's) phase retrieval algorithm.

`stp_core.phaseretrieval.tiehom.tiehom(im, plan, nr_threads=2)`

Process a tomographic projection image with the TIE-HOM (Paganin's) phase retrieval algorithm.

**Parameters**

- **im** (*array\_like*) – Flat corrected image data as numpy array.
- **plan** (*structure*) – Structure with pre-computed data (see `tiehom_plan` function).
- **nr\_threads** (*int*) – Number of threads to be used in the computation of FFT by PyFFTW (default = 2).

`stp_core.phaseretrieval.tiehom.tiehom_plan(im, beta, delta, energy, distance, pixsize, padding)`

Pre-compute data to save time in further execution of `phase_retrieval` with TIE-HOM (Paganin's) algorithm.

**Parameters**

- **im** (*array\_like*) – Image data as numpy array. Only image size (shape) is actually used.
- **beta** (*double*) – Imaginary part of the complex X-ray refraction index.
- **delta** (*double*) – Decrement from unity of the complex X-ray refraction index.
- **energy [KeV]** (*double*) – Energy in KeV of the incident X-ray beam.
- **distance [mm]** (*double*) – Sample-to-detector distance in mm.
- **pixsize [mm]** (*double*) – Size in mm of the detector element.
- **padding** (*bool*) – Apply image padding to better process the boundary of the image

**phaseretrieval.phrt****Functions:**

<code>phrt_plan(im, energy, distance, pixsize, ...)</code>	Pre-compute data to save time in further execution of <code>phase_retrieval</code> .
<code>phrt(im, plan[, method, nr_threads])</code>	Process a tomographic projection image with the selected phase retrieval algorithm.

`stp_core.phaseretrieval.phrt.phrt(im, plan, method=4, nr_threads=2)`

Process a tomographic projection image with the selected phase retrieval algorithm.

**Parameters**

- **im** (*array\_like*) – Flat corrected image data as numpy array.
- **plan** (*structure*) – Structure with pre-computed data (see `prepare_plan` function)
- **method** (*int*) – Phase retrieval filter { 1 = TIE (default), 2 = CTF, 3 = CTF first-half sine, 4 = Quasiparticle, 5 = Quasiparticle first half sine }.
- **nr\_threads** (*int*) – Number of threads to be used in the computation of FFT by PyFFTW
- **Credits**
- \_\_\_\_\_
- **Julian Moosmann, KIT (Germany) is acknowledged for this code**

`stp_core.phaseretrieval.phrt.phrt_plan(im, energy, distance, pixsize, regpar, thresh, method, padding)`

Pre-compute data to save time in further execution of `phase_retrieval`.

**Parameters**

- **im** (*array\_like*) – Image data as numpy array. Only image size (shape) is actually used.
- **energy [KeV]** (*double*) – Energy in KeV of the incident X-ray beam.
- **distance [mm]** (*double*) – Sample-to-detector distance in mm.
- **pixsize [mm]** (*double*) – Size in mm of the detector element.
- **regpar** (*double*) – Regularization parameter: RegPar is  $-\log_{10}$  of the constant to be added to the denominator to regularize the singularity at zero frequency, i.e.  $1/\sin(x) \rightarrow 1/(\sin(x)+10^{\text{RegPar}})$ . Typical values in the range [2.0, 3.0]. (Suggestion for default: 2.5).
- **thresh** (*double*) – Parameter for Quasiparticle phase retrieval which defines the width of the rings to be cropped around the zero crossing of the CTF denominator in Fourier space. Typical values in the range [0.01, 0.1]. (Suggestion for default: 0.1).
- **method** (*int*) – Phase retrieval algorithm { 1 = TIE (default), 2 = CTF, 3 = CTF first-half sine, 4 = Quasiparticle, 5 = Quasiparticle first half sine }.
- **padding** (*bool*) – Apply image padding to better process the boundary of the image.

## References

## Notes

Credits to Julian Moosmann, KIT (Germany) is acknowledged for this code

## postprocess

### Functions:

---

<code>postprocess(im, convert_opt, crop_opt)</code>	Post-process a reconstructed image.
-----------------------------------------------------	-------------------------------------

---

`stp_core.postprocess.postprocess.postprocess(im, convert_opt, crop_opt)`  
 Post-process a reconstructed image.

**Parameters** **im** (*array\_like*) – Image data as numpy array.

**convert\_opt** [string] String containing degradation method (8-bit or 16-bit) and min/max rescaling value (e.g. “linear8:-0.01;0.01”). In current version only “linear” for 16-bit and “linear8” are implemented.

**crop\_opt** [double] String containing the parameters to crop an image separated by : with order top, bottom, left, right. (e.g. “100:100:100:100”)

## preprocess.extfov\_correction

### Functions:

---

<code>extfov_correction(im, ext_fov, ...)</code>	Apply sinogram correction for extended FOV acquisition mode
--------------------------------------------------	-------------------------------------------------------------

---

`stp_core.preprocess.extfov_correction.extfov_correction` (*im*, *ext\_fov*,  
*ext\_fov\_rot\_right*,  
*ext\_fov\_overlap*)

Apply sinogram correction for extended FOV acquisition mode

#### Parameters

- **im** (*array\_like*) – Image data (sinogram) as numpy array.
- **ext** (*bool*) – True if the extended FOV mode has been performed.
- **ext\_fov\_rot\_right** (*bool*) –  
**True if the extended FOV mode has been performed with rotation center** shifted to the right, left otherwise.
- **ext\_fov\_overlap** [*int*] Number of overlapping pixels.

## preprocess.extract\_flatdark

### Functions:

---

<code>extract_flatdark</code> ( <i>f_in</i> , <i>flat_end</i> , <i>logfilename</i> )	Extract the flat and dark reference images to be used during the pre-processing step.
--------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------

---

`stp_core.preprocess.extract_flatdark.extract_flatdark` (*f\_in*, *flat\_end*, *logfilename*)  
Extract the flat and dark reference images to be used during the pre-processing step.

#### Parameters

- **f\_in** (*HDF5 data structure*) – The data structure containing the flat and dark acquired images.
- **flat\_end** (*bool*) – Consider the flat/dark images acquired after the projections (if any).
- **logfilename** (*string*) – Absolute file of a log text file where infos are appended.

## preprocess.flat\_fielding

### Functions:

---

<code>flat_fielding</code> ( <i>im</i> , <i>i</i> , <i>plan</i> , <i>flat_end</i> , ...)	Process a sinogram with conventional flat fielding plus reference normalization.
------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------

---

`stp_core.preprocess.flat_fielding.flat_fielding` (*im*, *i*, *plan*, *flat\_end*, *half\_half*,  
*half\_half\_line*, *norm\_sx*, *norm\_dx*)  
Process a sinogram with conventional flat fielding plus reference normalization.

#### Parameters

- **im** (*array\_like*) – Image data as numpy array
- **i** (*int*) – Index of the sinogram with reference to the height of a projection
- **plan** (*structure*) – Structure created by the `extract_flatdark` function (see `extract_flatdark.py`). This structure contains the flat/dark images acquired before the acqui-



sition of the projections and the flat/dark images acquired after the acquisition of the projections as well as a few flags.

- **flat\_end** (*bool*) – True if the process considers the flat/dark images (if any) acquired after the acquisition of the projections.
- **half\_half** (*bool*) – True if the process has to be separated by processing the first part of the sinogram with the flat/dark images acquired before the acquisition of the projections and the second part with the flat/dark images acquired after the acquisition of the projections.
- **half\_half\_line** (*int*) – Usually this value is equal to the height of the projection FOV / 2 but the two parts of the sinogram to process can have a different size.
- **norm\_sx** (*int*) – Width in pixels of the left window to be consider for the normalization of the sinogram. This value has to be zero in the case of ROI-CT.
- **norm\_dx** (*int*) – Width in pixels of the right window to be consider for the normalization of the sinogram. This value has to be zero in the case of ROI-CT.
- **Example (using h5py, tdf.py, tiff.py)**
- \_\_\_\_\_
- ```
>>> sino_idx = 512
```
- ```
>>> f = getHDF5('dataset.h5', 'r')
```
- ```
>>> im = tdf.read_sino(f['exchange/data'], sino_idx)
```
- ```
>>> plan = extract_flatdark(f_in, True, False, False, 'tomo', 'dark', 'flat', 'logfile.txt')
```
- ```
>>> im = flat_fielding(im, sino_idx, plan, True, True, 900, 0, 0)
```
- ```
>>> imsave('sino_corr.tif', im)
```

## preprocess.ring\_correction

### Functions:

---

<i>ring_correction</i> (im, ringrem, flat_end, ...)	Apply ring artifacts compensation by de-striping the input sinogram.
-----------------------------------------------------	----------------------------------------------------------------------

---

```
stp_core.preprocess.ring_correction.ring_correction(im, ringrem, flat_end,
                                                    skip_flat_after, half_half,
                                                    half_half_line, ext_fov)
```

Apply ring artifacts compensation by de-striping the input sinogram.

**Parameters** **im** (*array\_like*) – Image data (sinogram) as numpy array.

**ringrem** [string] String containing ring removal method and parameters

**half\_half** [bool] True to separately process the sinogram in two parts

**half\_half\_line** [int] Line number considered to identify the two parts to be processed separately.  
(This parameter is ignored if half\_half is False)

skip\_flat\_after e ext\_fov SERVE???

## reconstruct.rec\_astra

**Functions:**

---

<code>recon_astra_fbp(im, angles, method, filter_type)</code>	Reconstruct the input sinogram by using the FBP implemented in ASTRA toolbox.
<code>recon_astra_iterative(im, angles, method, ...)</code>	Reconstruct the input sinogram by using one of the iterative algorithms implemented in ASTRA toolbox.

---

`stp_core.reconstruct.rec_astra.recon_astra_fbp(im, angles, method, filter_type)`  
Reconstruct the input sinogram by using the FBP implemented in ASTRA toolbox.

**Parameters** `im` (*array\_like*) – Image data (sinogram) as numpy array.

**angles** [double] Value in radians representing the number of angles of the sinogram.

**method** [string] A string with either “FBP” or “FBP\_CUDA”.

**filter\_type** [string] The available options are “ram-lak”, “shepp-logan”, “cosine”, “hamming”, “hann”, “tukey”, “lanczos”, “triangular”, “gaussian”, “barlett-hann”, “blackman”, “nuttall”, “blackman-harris”, “blackman-nuttall”, “flat-top”, “kaiser”, “parzen”.

`stp_core.reconstruct.rec_astra.recon_astra_iterative(im, angles, method, iterations, zerone_mode)`

Reconstruct the input sinogram by using one of the iterative algorithms implemented in ASTRA toolbox.

**Parameters** `im` (*array\_like*) – Image data (sinogram) as numpy array.

**angles** [double] Value in radians representing the number of angles of the sinogram.

**method** [string] A string with e.g “SIRT” or “SIRT\_CUDA” (see ASTRA documentation)

**iterations** [int] Number of iterations for the algebraic technique

**zerone\_mode** [bool] True if the input sinogram has been rescaled to the [0,1] range (therefore positivity constraints are applied)

**reconstruct.rec\_fista\_tv****Functions:**

---

<code>recon_fista_tv(im, angles, lam, fista_iter, iter)</code>	Reconstruct the input sinogram by using the FISTA-TV algorithm
----------------------------------------------------------------	----------------------------------------------------------------

---

`stp_core.reconstruct.rec_fista_tv.recon_fista_tv(im, angles, lam, fista_iter, iter)`  
Reconstruct the input sinogram by using the FISTA-TV algorithm

**Parameters** `im` (*array\_like*) – Image data (sinogram) as numpy array.

**angles** [double] Value in radians representing the number of angles of the input sinogram.

**lam** [double] Regularization parameter of the FISTA algorithm.

**fista\_iter** [int] Number of iterations of the FISTA algorithm.

**iter** [int] Number of iterations of the TV minimization.

**reconstruct.rec\_gridrec**

**Functions:**


---

<code>recon_gridrec(im1, im2, angles, oversampling)</code>	Reconstruct two sinograms (of the same CT scan) with direct Fourier algorithm.
------------------------------------------------------------	--------------------------------------------------------------------------------

---

`stp_core.reconstruct.rec_gridrec.recon_gridrec(im1, im2, angles, oversampling)`  
 Reconstruct two sinograms (of the same CT scan) with direct Fourier algorithm.

**Parameters**

- **im1** (*array\_like*) – Sinogram image data as numpy array.
- **im2** (*array\_like*) – Sinogram image data as numpy array.
- **angles** (*double*) – Value in radians representing the number of angles of the input sinogram.
- **oversampling** (*double*) – Input sinogram is rescaled to increase the sampling of the Fourier space and avoid artifacts. Suggested value in the range [1.2,1.6].

**reconstruct.rec\_mr\_fbp****Functions:**


---

<code>recon_mr_fbp(im, angles)</code>	Reconstruct a sinogram with the Minimum Residual FBP algorithm (Pelt, 2013).
---------------------------------------	------------------------------------------------------------------------------

---

`stp_core.reconstruct.rec_mr_fbp.recon_mr_fbp(im, angles)`  
 Reconstruct a sinogram with the Minimum Residual FBP algorithm (Pelt, 2013).

**Parameters**

- **im** (*array\_like*) – Sinogram image data as numpy array.
- **angles** (*double*) – Value in radians representing the number of angles of the input sinogram.

**utils.caching****Functions:**


---

<code>cache2plan(infile, cachepath)</code>	Read from cache the flat/dark images of the input TDF file.
<code>plan2cache(corr_plan, infile, cachepath)</code>	Write to cache the flat/dark images of the input TDF file.

---

`stp_core.utils.caching.cache2plan(infile, cachepath)`  
 Read from cache the flat/dark images of the input TDF file.

**Parameters** `infile` (*string*) – Absolute path of the input TDF dataset.

**returns** *A structure with flat/dark images and related flags.*

`stp_core.utils.caching.plan2cache(corr_plan, infile, cachepath)`  
 Write to cache the flat/dark images of the input TDF file.

**Parameters** `infile` (*string*) – Absolute path of the input TDF dataset.

**corr\_plan** [structure] The plan with flat/dark images and flags.

**returns** *No return value.*

## utils.findcenter

### Functions:

---

<code>usecorrelation(im1, im2)</code>	Assess the offset (to be used for e.g.
---------------------------------------	----------------------------------------

---

`stp_core.utils.findcenter.usecorrelation(im1, im2)`

Assess the offset (to be used for e.g. the assessment of the center of rotation or the overlap) by computation the peak of the correlation between the two input images.

**Parameters** **im1** (*array\_like*) – Image data as numpy array.

**im2** [*array\_like*] Image data as numpy array.

**returns** *An integer value of the location of the maximum peak correlation.*

## utils.padding

### Functions:

---

<code>upperPowerOfTwo(v)</code>	Return the upper power of two of input value
<code>replicatePadImage(im, marg0, marg1)</code>	Pad the input image by replicating first and last column as well as first and last row the specified number of times.
<code>zeroPadImage(im, marg0, marg1)</code>	Pad the input image by adding zeros.
<code>padImage(im, n_pad0, n_pad1)</code>	Replicate pad the input image to the specified new dimensions.
<code>padSmoothWidth(im, n_pad)</code>	Pad the input image to the specified new width by replicate padding with Hanning smoothing to zero.

---

`stp_core.utils.padding.padImage(im, n_pad0, n_pad1)`

Replicate pad the input image to the specified new dimensions.

#### Parameters

- **im** (*array\_like*) – Image data as numpy array
- **n\_pad0** (*int*) – The new height of the image
- **n\_pad1** (*int*) – The new width of the image
- **Return value**
- \_\_\_\_\_
- **A padded image**

`stp_core.utils.padding.padSmoothWidth(im, n_pad)`

Pad the input image to the specified new width by replicate padding with Hanning smoothing to zero.

**Parameters**

- **im** (*array\_like*) – Image data as numpy array.
- **n\_pad** (*int*) – The new width of the image.
- **Return value**
- \_\_\_\_\_
- **A padded image**

`stp_core.utils.padding.replicatePadImage(im, marg0, marg1)`

**Pad the input image by replicating first and last column as well as first and last row** the specified number of times.

**Parameters**

- **im** (*array\_like*) – Image data as numpy array.
- **marg0** (*int*) – The number of times first and last row have to be replicated.
- **marg1** (*int*) – The number of times first and last column have to be replicated.
- **Return value**
- \_\_\_\_\_
- **A replicated-padded image.**

`stp_core.utils.padding.upperPowerOfTwo(v)`

Return the upper power of two of input value

**Parameters**

- **v** (*int*) – A positive integer value
- **Return value**
- \_\_\_\_\_
- **An integer value**

`stp_core.utils.padding.zeroPadImage(im, marg0, marg1)`

Pad the input image by adding zeros.

**Parameters**

- **im** (*array\_like*) – Image data as numpy array.
- **marg0** (*int*) – The number of zero rows to add before first and after last row.
- **marg1** (*int*) – The number of zero rows to add before first and after last column.
- **Return value**
- \_\_\_\_\_
- **A zero-padded image.**

## Examples

Here we describe what the examples are doing. You can cite with [\[B1\]](#).

## exec\_his2tdf

This section contains the `exec_his2tdf` script.

Download file: `exec_his2tdf.py`

```

1 #####
2 # (C) 2016 Elettra - Sincrotrone Trieste S.C.p.A.. All rights reserved. #
3 # #
4 # #
5 # This file is part of STP-Core, the Python core of SYRMEP Tomo Project, #
6 # a software tool for the reconstruction of experimental CT datasets. #
7 # #
8 # STP-Core is free software: you can redistribute it and/or modify it #
9 # under the terms of the GNU General Public License as published by the #
10 # Free Software Foundation, either version 3 of the License, or (at your #
11 # option) any later version. #
12 # #
13 # STP-Core is distributed in the hope that it will be useful, but WITHOUT #
14 # ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or #
15 # FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License #
16 # for more details. #
17 # #
18 # You should have received a copy of the GNU General Public License #
19 # along with STP-Core. If not, see <http://www.gnu.org/licenses/>. #
20 # #
21 #####
22
23 #
24 # Author: Francesco Brun
25 # Last modified: July, 8th 2016
26 #
27
28 import datetime
29 import os
30 import os.path
31 import numpy
32 import time
33
34 from time import strftime
35 from sys import argv, exit
36 from glob import glob
37 from h5py import File as getHDF5
38 import stp_core.io.tdf as tdf
39
40 def _getHISdim ( HISfilename ):
41
42     dim1 = 0
43     dim2 = 0
44     dimz = 0
45     bytecode = numpy.uint16
46
47     # Open file:
48     try:
49         infile = open(HISfilename, "rb")
50
51         # Get file infos:
52         tot_bytes = os.path.getsize(HISfilename)
53

```

```

54     # Read header:
55     Image_tag = infile.read(2)
56     Comment_len = numpy.fromstring(infile.read(2), numpy.uint16)[0].astype(numpy.
↪int_)
57     dim1 = numpy.fromstring(infile.read(2), numpy.uint16)[0].astype(numpy.int_)
58     dim2 = numpy.fromstring(infile.read(2), numpy.uint16)[0].astype(numpy.int_)
59     dim1_offset = numpy.fromstring(infile.read(2), numpy.uint16)[0].astype(numpy.
↪int_)
60     dim2_offset = numpy.fromstring(infile.read(2), numpy.uint16)[0].astype(numpy.
↪int_)
61     HeaderType = numpy.fromstring(infile.read(2), numpy.uint16)[0]
62     Dump = infile.read(50)
63     Comment = infile.read(Comment_len)
64
65     # Set total number of bytes read so far:
66     bytes_read = 64 + Comment_len
67
68     # Set image type:
69     bpp = len(numpy.array(0, bytecode).tostring())
70
71     # Define chunk size:
72     chunksize = dim1 * dim2 * bpp
73
74     # Determine number of expected projections:
75     dimz = (tot_bytes - bytes_read) / (chunksize + 64) + 1
76
77     finally:
78         # Close file:
79         infile.close()
80
81     return (dim1, dim2, dimz, bytecode)
82
83
84
85 def _processHIS( HISfilename, dset, dset_offset, provenance_dset, provenance_offset,
↪time_offset, prefix, crop_top, crop_bottom, crop_left, crop_right, logfilename, int_
↪from=0, int_to=-1):
86
87     # Open file:
88     infile = open(HISfilename, "rb")
89
90     # Get file infos:
91     tot_bytes = os.path.getsize(HISfilename)
92
93     # Read header:
94     Image_tag = infile.read(2)
95     Comment_len = numpy.fromstring(infile.read(2), numpy.uint16)[0].astype(numpy.int_)
96     dim1 = numpy.fromstring(infile.read(2), numpy.uint16)[0].astype(numpy.int_)
97     dim2 = numpy.fromstring(infile.read(2), numpy.uint16)[0].astype(numpy.int_)
98     dim1_offset = numpy.fromstring(infile.read(2), numpy.uint16)[0].astype(numpy.int_)
99     dim2_offset = numpy.fromstring(infile.read(2), numpy.uint16)[0].astype(numpy.int_)
100    HeaderType = numpy.fromstring(infile.read(2), numpy.uint16)[0]
101    Dump = infile.read(50)
102    Comment = infile.read(Comment_len)
103
104    # Set total number of bytes read so far:
105    bytes_read = 64 + Comment_len
106

```

```

107 # Set image type:
108 bytecode = numpy.uint16
109 bpp = len(numpy.array(0, bytecode).tostring())
110
111 # Define chunk size:
112 chunksize = dim1 * dim2 * bpp
113
114 # Determine number of expected projections:
115 num_proj = (tot_bytes - bytes_read) / (chunksize + 64) + 1
116
117 # Read first projection:
118 t1 = time.time()
119 block = infile.read(chunksize)
120
121 # Convert as numpy array:
122 data = numpy.fromstring(block, bytecode)
123 im = numpy.reshape( data, [dim2, dim1])
124 im = im[crop_top:im.shape[0]-crop_bottom,crop_left:im.shape[1]-crop_right]
125
126 print numpy.amax(im[:])
127 print dset.attrs['max']
128
129 # Set minimum and maximum:
130 if ( numpy.amin(im[:]) < float(dset.attrs['min']) ):
131     dset.attrs['min'] = str(numpy.amin(im[:]))
132 if ( numpy.amax(im[:]) > float(dset.attrs['max']) ):
133     dset.attrs['max'] = str(numpy.amax(im[:]))
134
135 print numpy.amax(im[:])
136 print dset.attrs['max']
137
138 # Check extrema (int_to == -1 means all files) for the projections:
139 if ( (int_to >= num_proj) or (int_to <= 0) ):
140     int_to = num_proj - 1
141 if ( (int_from >= num_proj) or (int_from < 0) ):
142     int_from = 0
143
144 # Process first projection (fill HDF5):
145 i = 0
146 first_index = int(provenance_dset.attrs['first_index'])
147
148 # Save processed image to HDF5 file:
149 #tiff.imwrite('tomo_' + str(i).zfill(4) + '.tif', data)
150 if (i >= int_from) and (i <= int_to):
151     tdf.write_tomo(dset, i + dset_offset - int_from, im)
152
153 # Save provenance metadata:
154 t = time.time() + time_offset*3600
155 provenance_dset["filename", provenance_offset + i - int_from] = prefix + '_' +
↪ str(i + dset_offset + first_index).zfill(4)
156 provenance_dset["timestamp", provenance_offset + i - int_from] = numpy.
↪ string_(datetime.datetime.fromtimestamp(t).strftime('%Y-%m-%d %H:%M:%S.%f')[:-3])
157
158 # Print out execution time:
159 t2 = time.time()
160 log = open(logfilename, "a")
161 log.write(os.linesep + "\t%s converted in %0.3f sec." % (provenance_dset[
↪ "filename", provenance_offset + i - int_from], t2 - t1))

```



```

162     log.close()
163
164     # Read all the other projections:
165     try:
166         while block:
167
168             # Skip a few bytes:
169             t1 = time.time()
170             dump = infile.read(64)
171
172             # Read the meaningful data:
173             block = infile.read(chunksize)
174
175             # Convert as numpy array:
176             data = numpy.fromstring(block, bytecode)
177             im = numpy.reshape( data, [dim2, dim1])
178             im = im[crop_top:im.shape[0]-crop_bottom,crop_left:im.shape[1]-crop_
↪right]
179
180             # Set minimum and maximum:
181             if ( float(numpy.amin(im[:])) < float(dset.attrs['min']) ):
182                 dset.attrs['min'] = str(numpy.amin(im[:]))
183             if ( float(numpy.amax(im[:])) > float(dset.attrs['max'])):
184                 dset.attrs['max'] = str(numpy.amax(im[:]))
185
186             # Process first projection (fill HDF5):
187             i = i + 1
188
189             # Save processed image to HDF5 file:
190             #tiff.imsave('tomo_' + str(i).zfill(4) + '.tif', data)
191             if (i >= int_from) and (i <= int_to):
192                 tdf.write_tomo( dset, i + dset_offset - int_from, im )
193
194             # Save provenance metadata:
195             t = time.time() + time_offset*3600
196             provenance_dset["filename", provenance_offset + i - int_from] = _
↪prefix + '_' + str(i + dset_offset + first_index - int_from).zfill(4)
197             provenance_dset["timestamp", provenance_offset + i - int_from] = _
↪numpy.string_(datetime.datetime.fromtimestamp(t).strftime('%Y-%m-%d %H:%M:%S.%f')[:-
↪3])
198
199             # Print out execution time:
200             t2 = time.time()
201             log = open(logfilename, "a")
202             log.write(os.linesep + "\t%s converted in %0.3f sec." % (provenance_
↪dset["filename", provenance_offset + i - int_from], t2 - t1))
203             log.close()
204
205         except Exception, e:
206             #log = open(logfilename, "a")
207             #log.write(str(e))
208             #log.close()
209             pass
210
211     finally:
212         # Close file:
213         infile.close()
214

```

```

215     return provenance_offset + i + 1
216
217
218 def main(argv):
219     """
220     Converts a set of HIS files into a TDF file (HDF5 Tomo Data Format).
221
222     Parameters
223     -----
224     from : scalar, integer
225           among all the projections (or sinogram) files, a subset of files can be
226     ↪ specified,
227           ranging from the parameter "from" to the parameter "to" (see next). In most
228           cases, this parameter is 0.
229
230     to : scalar, integer
231           among all the projections (or sinogram) files, a subset of files can be
232     ↪ specified,
233           ranging from the parameter "from" (see previous parameter) to the parameter
234           "to". If the value -1 is specified, all the projection files will be
235     ↪ considered.
236
237     data_in_path : string
238           path of the HIS file of the projections (e.g. "Z:\\sample1.his").
239
240     dark_in_path : string
241           path of the HIS file of the flat (e.g. "Z:\\sample1_dark.his").
242
243     flat_in_path : string
244           path of the HIS file of the flat (e.g. "Z:\\sample1_flat.his").
245
246     postdark_in_path : string
247           path of the HIS file of the flat (e.g. "Z:\\sample1_postdark.his").
248
249     postflat_in_path : string
250           path of the HIS file of the flat (e.g. "Z:\\sample1_postflat.his").
251
252     out_file : string
253           path with filename of the TDF to create (e.g. "Z:\\sample1.tdf"). WARNING:
254     ↪ the program
255           does NOT automatically create non-existing folders and subfolders specified
256     ↪ in the path.
257           Moreover, if a file with the same name already exists it will be
258     ↪ automatically deleted and
259           overwritten.
260
261     crop_top : scalar, integer
262           during the conversion, images can be cropped if required. This parameter
263     ↪ specifies the number
264           of pixels to crop from the top of the image. Leave 0 for no cropping.
265
266     crop_bottom : scalar, integer
267           during the conversion, images can be cropped if required. This parameter
268     ↪ specifies the number
269           of pixels to crop from the bottom of the image. Leave 0 for no cropping.
270
271     crop_left : scalar, integer
272           during the conversion, images can be cropped if required. This parameter
273     ↪ specifies the number

```

```

265     of pixels to crop from the left of the image. Leave 0 for no cropping.
266
267     crop_right : scalar, integer
268     during the conversion, images can be cropped if required. This parameter_
↪ specifies the number
269     of pixels to crop from the right of the image. Leave 0 for no cropping.
270
271     privilege_sino : boolean string
272     specify the string "True" if the TDF will privilege a fast read/write of_
↪ sinograms (the most common
273     case), "False" for fast read/write of projections.
274
275     compression : scalar, integer
276     an integer value in the range of [1,9] to be used as GZIP compression factor_
↪ in the HDF5 file, where
277     1 is the minimum compression (and maximum speed) and 9 is the maximum (and_
↪ slow) compression.
278     The value 0 can be specified with the meaning of no compression.
279
280     log_file : string
281     path with filename of a log file (e.g. "R:\\log.txt") where info about the_
↪ conversion is reported.
282
283     Returns
284     -----
285     no return value
286
287     Example
288     -----
289     Example call to convert all the tomo*.tif* projections to a TDF with no cropping_
↪ and minimum compression:
290
291     python his2tdf.py 0 -1 "tomo.his" "dark.his" "flat.his" "postdark.his"
↪ "postflat.his" "dataset.tdf" 0 0 0 0
292     True True 1 "S:\\conversion.txt"
293
294     Requirements
295     -----
296     - Python 2.7 with the latest NumPy, SciPy, H5Py.
297     - tdf.py
298
299     Tests
300     -----
301     Tested with WinPython-64bit-2.7.6.3 (Windows) and Anaconda 2.1.0 (Linux 64-bit). _
↪
302
303     """
304
305     # Get the from and to number of files to process:
306     int_from = int(argv[0])
307     int_to = int(argv[1]) # -1 means "all files"
308
309     # Get paths:
310     tomo_file = argv[2]
311     dark_file = argv[3]
312     flat_file = argv[4]
313     darkpost_file = argv[5]
314     flatpost_file = argv[6]

```

```

315 outfile = argv[7]
316
317
318 crop_top      = int(argv[8]) # 0 for all means "no cropping"
319 crop_bottom  = int(argv[9])
320 crop_left    = int(argv[10])
321 crop_right   = int(argv[11])
322
323 projorder = argv[12]
324 if projorder == "True":
325     projorder = True
326 else:
327     projorder = False
328
329 privilege_sino = argv[13]
330 if privilege_sino == "True":
331     privilege_sino = True
332 else:
333     privilege_sino = False
334
335 # Get compression factor:
336 compr_opts = int(argv[14])
337 compressionFlag = True;
338 if (compr_opts <= 0):
339     compressionFlag = False;
340 elif (compr_opts > 9):
341     compr_opts = 9
342
343 logfilename = argv[15]
344
345 # Get the files in inpath:
346 log = open(logfilename, "w")
347 log.write(os.linesep + "\tInput HIS files:")
348 log.write(os.linesep + "\t\tProjections: %s" % (tomo_file))
349 log.write(os.linesep + "\t\tDark: %s" % (dark_file))
350 log.write(os.linesep + "\t\tFlat: %s" % (flat_file))
351 log.write(os.linesep + "\t\tPost dark: %s" % (darkpost_file))
352 log.write(os.linesep + "\t\tPost flat: %s" % (flatpost_file))
353 log.write(os.linesep + "\tOutput TDF file: %s" % (outfile))
354 log.write(os.linesep + "\t-----")
355 log.write(os.linesep + "\tCropping:")
356 log.write(os.linesep + "\t\tTop: %d pixels" % (crop_top))
357 log.write(os.linesep + "\t\tBottom: %d pixels" % (crop_bottom))
358 log.write(os.linesep + "\t\tLeft: %d pixels" % (crop_left))
359 log.write(os.linesep + "\t\tRight: %d pixels" % (crop_right))
360 if (int_to != -1):
361     log.write(os.linesep + "\tThe subset [%d,%d] of the input files will be_
↪considered." % (int_from, int_to))
362
363 if (projorder):
364     log.write(os.linesep + "\tProjection order assumed.")
365 else:
366     log.write(os.linesep + "\tSinogram order assumed.")
367
368 if (privilege_sino):
369     log.write(os.linesep + "\tFast I/O for sinograms privileged.")
370 else:
371     log.write(os.linesep + "\tFast I/O for projections privileged.")

```

```

372
373     if (compressionFlag):
374         log.write(os.linesep + "\tTDF compression factor: %d" % (compr_opts))
375     else:
376         log.write(os.linesep + "\tTDF compression: none.")
377
378     log.write(os.linesep + "\t-----")
379     log.close()
380
381     # Remove a previous copy of output:
382     if os.path.exists(outfile):
383         log = open(logfilename, "a")
384         log.write(os.linesep + "\tWarning: an output file with the same name was_
↪overwritten.")
385         os.remove(outfile)
386         log.close()
387
388     # Check input file:
389     if not os.path.exists(tomo_file):
390         log = open(logfilename, "a")
391         log.write(os.linesep + "\tError: input HIS file for projections does not_
↪exist. Process will end.")
392         log.close()
393         exit()
394
395     # First time get the plan:
396     log = open(logfilename, "a")
397     log.write(os.linesep + "\tPreparing the work plan...")
398     log.close()
399
400     # Get info from projection file:
401     dim1, dim2, dimz, dtype = _getHISdim ( tomo_file )
402
403
404     if ( ((int_to - int_from + 1) > 0) and ((int_to - int_from + 1) < dimz) ):
405         dimz = int_to - int_from + 1
406
407     #dsetshape = (num_files,) + im.shape
408     if projorder:
409         #dsetshape = tdf.get_dset_shape(privilege_sino, im.shape[1], im.shape[0], num_
↪files)
410         dsetshape = tdf.get_dset_shape(dim1 - crop_left - crop_right, dim2 - crop_top_
↪crop_bottom, dimz)
411     else:
412         #dsetshape = tdf.get_dset_shape(privilege_sino, im.shape[1], num_files, im.
↪shape[0])
413         dsetshape = tdf.get_dset_shape(dim1 - crop_left - crop_right, dim2 - crop_top_
↪crop_bottom, dimz)
414
415     f = getHDF5( outfile, 'w' )
416     print dsetshape
417
418     f.attrs['version'] = '1.0'
419     f.attrs['implements'] = "exchange:provenance"
420     echange_group = f.create_group( 'exchange' )
421
422     if (compressionFlag):
423         dset = f.create_dataset('exchange/data', dsetshape, dtype, chunks=tdf.get_
↪dset_chunks(dim1 - crop_left - crop_right), compression="gzip", compression_
↪opts=compr_opts, shuffle=True, fletcher32=True)

```

```

424     else:
425         dset = f.create_dataset('exchange/data', dsetshape, dtype)
426
427     if privilege_sino:
428         dset.attrs['axes'] = "y:theta:x"
429     else:
430         dset.attrs['axes'] = "theta:y:x"
431
432     dset.attrs['min'] = str(numpy.iinfo(dtype).max)
433     dset.attrs['max'] = str(numpy.iinfo(dtype).min)
434
435     # Get the total number of files to consider:
436     num_darks = 0
437     num_flats = 0
438     num_postdarks = 0
439     num_postflats = 0
440
441     if os.path.exists(dark_file):
442         dim1, dim2, num_darks, dtype = _getHISdim ( dark_file )
443     if os.path.exists(flat_file):
444         dim1, dim2, num_flats, dtype = _getHISdim ( flat_file )
445     if os.path.exists(darkpost_file):
446         dim1, dim2, num_postdarks, dtype = _getHISdim ( darkpost_file )
447     if os.path.exists(flatpost_file):
448         dim1, dim2, num_postflats, dtype = _getHISdim ( flatpost_file )
449
450     tot_files = dimz + num_darks + num_flats + num_postdarks + num_postflats
451
452     # Create provenance dataset:
453     provenance_dt = numpy.dtype([("filename", numpy.dtype("S255")), ("timestamp",
↳numpy.dtype("S255"))])
454     metadata_group = f.create_group( 'provenance' )
455     provenance_dset = metadata_group.create_dataset('detector_output', (tot_files,),
↳dtype=provenance_dt)
456
457     provenance_dset.attrs['tomo_prefix'] = 'tomo';
458     provenance_dset.attrs['dark_prefix'] = 'dark';
459     provenance_dset.attrs['flat_prefix'] = 'flat';
460     provenance_dset.attrs['first_index'] = 1;
461
462     # Handle the metadata:
463     if (os.path.isfile(os.path.dirname(tomo_file) + os.sep + 'logfile.xml')):
464         with open (os.path.dirname(tomo_file) + os.sep + 'logfile.xml', "r") as file:
465             xml_command = file.read()
466             tdf.parse_metadata(f, xml_command)
467
468     # Print out about plan preparation:
469     first_done = True
470     log = open(logfilename, "a")
471     log.write(os.linesep + "\tWork plan prepared succesfully.")
472     log.close()
473
474
475     # Get the data from HIS:
476     if (num_darks > 0) or (num_postdarks > 0):
477         #dsetshape = (num_files,) + im.shape
478         if projorder:
479             #dsetshape = tdf.get_dset_shape(privilege_sino, im.shape[1], im.shape[0],
↳num_files)

```

```

480         dsetshape = tdf.get_dset_shape(dim1 - crop_left - crop_right, dim2 - crop_
↳top - crop_bottom, num_darks + num_postdarks)
481     else:
482         #dsetshape = tdf.get_dset_shape(privilege_sino, im.shape[1], num_files,
↳im.shape[0])
483         dsetshape = tdf.get_dset_shape(dim1 - crop_left - crop_right, dim2 - crop_
↳top - crop_bottom, num_darks + num_postdarks)
484
485     if (compressionFlag):
486         darkdset = f.create_dataset('exchange/data_dark', dsetshape, dtype,
↳chunks=tdf.get_dset_chunks(dim1 - crop_left - crop_right), compression="gzip",
↳compression_opts=compr_opts, shuffle=True, fletcher32=True)
487     else:
488         darkdset = f.create_dataset('exchange/data_dark', dsetshape, dtype)
489
490     if privilege_sino:
491         darkdset.attrs['axes'] = "y:theta:x"
492     else:
493         darkdset.attrs['axes'] = "theta:y:x"
494
495     darkdset.attrs['min'] = str(numpy.iinfo(dtype).max)
496     darkdset.attrs['max'] = str(numpy.iinfo(dtype).min)
497     else:
498         log = open(logfilename, "a")
499         log.write(os.linesep + "\tWarning: dark images (if any) not considered.")
500         log.close()
501
502     if (num_flats > 0) or (num_postflats > 0):
503
504         #dsetshape = (num_files,) + im.shape
505         if projorder:
506             #dsetshape = tdf.get_dset_shape(privilege_sino, im.shape[1], im.shape[0],
↳num_files)
507             dsetshape = tdf.get_dset_shape(dim1 - crop_left - crop_right, dim2 - crop_
↳top - crop_bottom, num_flats + num_postflats)
508         else:
509             #dsetshape = tdf.get_dset_shape(privilege_sino, im.shape[1], num_files,
↳im.shape[0])
510             dsetshape = tdf.get_dset_shape(dim1 - crop_left - crop_right, dim2 - crop_
↳top - crop_bottom, num_flats + num_postflats)
511
512     if (compressionFlag):
513         flatdset = f.create_dataset('exchange/data_white', dsetshape, dtype,
↳chunks=tdf.get_dset_chunks(dim1 - crop_left - crop_right), compression="gzip",
↳compression_opts=compr_opts, shuffle=True, fletcher32=True)
514     else:
515         flatdset = f.create_dataset('exchange/data_white', dsetshape, dtype)
516
517     if privilege_sino:
518         flatdset.attrs['axes'] = "y:theta:x"
519     else:
520         flatdset.attrs['axes'] = "theta:y:x"
521
522     flatdset.attrs['min'] = str(numpy.iinfo(dtype).max)
523     flatdset.attrs['max'] = str(numpy.iinfo(dtype).min)
524
525     else:
526         log = open(logfilename, "a")

```





```

15 # FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License #
16 # for more details. #
17 # #
18 # You should have received a copy of the GNU General Public License #
19 # along with STP-Core. If not, see <http://www.gnu.org/licenses/>. #
20 # #
21 #####
22
23 #
24 # Author: Francesco Brun
25 # Last modified: August, 8th 2016
26 #
27
28 from sys import argv, exit
29 from os import remove, sep, linesep
30 from os.path import exists
31 from numpy import float32, amin, amax, isscalar
32 from time import time
33 from multiprocessing import Process, Lock
34
35 # pystp-specific:
36 from stp_core.preprocess.extfov_correction import extfov_correction
37 from stp_core.preprocess.flat_fielding import flat_fielding
38 from stp_core.preprocess.dynamic_flatfielding import dff_prepare_plan, dynamic_flat_
39 ↪fielding
40 from stp_core.preprocess.ring_correction import ring_correction
41 from stp_core.preprocess.extract_flatdark import extract_flatdark, _medianize
42
43 from h5py import File as getHDF5
44
45 # pystp-specific:
46 import stp_core.io.tdf as tdf
47
48 def _write_data(lock, im, index, outfile, outshape, outtype, logfilefilename, cputime,
49 ↪itime):
50
51     lock.acquire()
52     try:
53         t0 = time()
54         f_out = getHDF5( outfile, 'a' )
55         f_out_dset = f_out.require_dataset('exchange/data', outshape, outtype,
56 ↪chunks=tdf.get_dset_chunks(outshape[0]))
57         tdf.write_sino(f_out_dset, index, im.astype(float32))
58
59         # Set minimum and maximum:
60         if ( amin(im[:]) < float(f_out_dset.attrs['min']) ):
61             f_out_dset.attrs['min'] = str(amin(im[:]))
62         if ( amax(im[:]) > float(f_out_dset.attrs['max']) ):
63             f_out_dset.attrs['max'] = str(amax(im[:]))
64         f_out.close()
65         t1 = time()
66
67         # Print out execution time:
68         log = open(logfilefilename, "a")
69         log.write(linesep + "\tsino_%s processed (CPU: %0.3f sec - I/O: %0.3f sec)."
70 ↪% (str(index).zfill(4), cputime, t1 - t0 + itime))

```

```

68     log.close()
69
70     finally:
71         lock.release()
72
73 def _process (lock, int_from, int_to, infile, outfile, outshape, outtype, skipflat,
74 ↪ plan, norm_sx, norm_dx, flat_end,
75 ↪ half_half, half_half_line, ext_fov, ext_fov_rot_right, ext_fov_overlap,
76 ↪ ringrem, dynamic_ff, EFF,
77 ↪ filtEFF, im_dark, logfilename):
78
79     # Process the required subset of images:
80     for i in range(int_from, int_to + 1):
81
82         # Read input image:
83         t0 = time()
84         f_in = getHDF5(infile, 'r')
85         if "/tomo" in f_in:
86             dset = f_in['tomo']
87         else:
88             dset = f_in['exchange/data']
89         im = tdf.read_sino(dset, i).astype(float32)
90         f_in.close()
91         t1 = time()
92
93         # Perform pre-processing (flat fielding, extended FOV, ring removal):
94         if not skipflat:
95             if dynamic_ff:
96                 # Dynamic flat fielding with downsampling = 2:
97                 im = dynamic_flat_fielding(im, i, EFF, filtEFF, 2, im_dark, norm_sx,
98 ↪ norm_dx)
99             else:
100                 im = flat_fielding(im, i, plan, flat_end, half_half, half_half_line,
101 ↪ norm_sx, norm_dx)
102                 im = extfov_correction(im, ext_fov, ext_fov_rot_right, ext_fov_overlap)
103                 if not skipflat and not dynamic_ff:
104                     im = ring_correction(im, ringrem, flat_end, plan['skip_flat_after'],
105 ↪ half_half, half_half_line, ext_fov)
106                 else:
107                     im = ring_correction(im, ringrem, False, False, half_half, half_half_
108 ↪ line, ext_fov)
109                 t2 = time()
110
111         # Save processed image to HDF5 file (atomic procedure - lock used):
112         _write_data(lock, im, i, outfile, outshape, outtype, logfilename, t2 - t1, t1,
113 ↪ - t0)
114
115 def main(argv):
116     """To do...
117
118     Usage
119     ----
120
121     Parameters
122     -----
123
124     Example

```

```

119 -----
120 The following line processes the first ten TIFF files of input path
121 "/home/in" and saves the processed files to "/home/out" with the
122 application of the Boin and Haibel filter with smoothing via a Butterworth
123 filter of order 4 and cutoff frequency 0.01:
124
125 destripe /home/in /home/out 1 10 1 0.01 4
126
127 ""
128 lock = Lock()
129
130 # Get the from and to number of files to process:
131 int_from = int(argv[0])
132 int_to = int(argv[1])
133
134 # Get paths:
135 infile = argv[2]
136 outfile = argv[3]
137
138 # Normalization parameters:
139 norm_sx = int(argv[4])
140 norm_dx = int(argv[5])
141
142 # Params for flat fielding with post flats/darks:
143 flat_end = True if argv[6] == "True" else False
144 half_half = True if argv[7] == "True" else False
145 half_half_line = int(argv[8])
146
147 # Params for extended FOV:
148 ext_fov = True if argv[9] == "True" else False
149 ext_fov_rot_right = argv[10]
150 if ext_fov_rot_right == "True":
151     ext_fov_rot_right = True
152     if (ext_fov):
153         norm_sx = 0
154 else:
155     ext_fov_rot_right = False
156     if (ext_fov):
157         norm_dx = 0
158 ext_fov_overlap = int(argv[11])
159
160 # Method and parameters coded into a string:
161 ringrem = argv[12]
162
163 # Flat fielding method (conventional or dynamic):
164 dynamic_ff = True if argv[13] == "True" else False
165
166 # Nr of threads and log file:
167 nr_threads = int(argv[14])
168 logfilename = argv[15]
169
170
171
172
173 # Log input parameters:
174 log = open(logfilename, "w")
175 log.write(linesep + "\tInput TDF file: %s" % (infile))

```

```

176 log.write(linesep + "\tOutput TDF file: %s" % (outfile))
177 log.write(linesep + "\t-----")
178 log.write(linesep + "\tOpening input dataset...")
179 log.close()
180
181 # Remove a previous copy of output:
182 if exists(outfile):
183     remove(outfile)
184
185 # Open the HDF5 file:
186 f_in = getHDF5(infile, 'r')
187
188
189 if "/tomo" in f_in:
190     dset = f_in['tomo']
191
192     tomaprefix = 'tomo'
193     flatprefix = 'flat'
194     darkprefix = 'dark'
195 else:
196     dset = f_in['exchange/data']
197     if "/provenance/detector_output" in f_in:
198         prov_dset = f_in['provenance/detector_output']
199
200         tomaprefix = prov_dset.attrs['tomo_prefix']
201         flatprefix = prov_dset.attrs['flat_prefix']
202         darkprefix = prov_dset.attrs['dark_prefix']
203
204 num_proj = tdf.get_nr_projs(dset)
205 num_sinos = tdf.get_nr_sinos(dset)
206
207 if (num_sinos == 0):
208     log = open(logfilename, "a")
209     log.write(linesep + "\tNo projections found. Process will end.")
210     log.close()
211     exit()
212
213 # Check extrema (int_to == -1 means all files):
214 if ((int_to >= num_sinos) or (int_to == -1)):
215     int_to = num_sinos - 1
216
217 # Prepare the work plan for flat and dark images:
218 log = open(logfilename, "a")
219 log.write(linesep + "\t-----")
220 log.write(linesep + "\tPreparing the work plan...")
221 log.close()
222
223 # Extract flat and darks:
224 skipflat = False
225 skipdark = False
226
227 # Following variables make sense only for dynamic flat fielding:
228 EFF = -1
229 filteFF = -1
230 im_dark = -1
231
232 # Following variable makes sense only for conventional flat fielding:

```

```

233 plan = -1
234
235 if not dynamic_ff:
236     plan = extract_flatdark(f_in, flat_end, logfile)
237     if (isscalar(plan['im_flat']) and isscalar(plan['im_flat_after']))):
238         skipflat = True
239     else:
240         skipflat = False
241 else:
242     # Dynamic flat fielding:
243     if "/tomo" in f_in:
244         if "/flat" in f_in:
245             flat_dset = f_in['flat']
246             if "/dark" in f_in:
247                 im_dark = _medianize(f_in['dark'])
248             else:
249                 skipdark = True
250         else:
251             skipflat = True # Nothing to do in this case
252     else:
253         if "/exchange/data_white" in f_in:
254             flat_dset = f_in['/exchange/data_white']
255             if "/exchange/data_dark" in f_in:
256                 im_dark = _medianize(f_in['/exchange/data_dark'])
257             else:
258                 skipdark = True
259         else:
260             skipflat = True # Nothing to do in this case
261
262     # Prepare plan for dynamic flat fielding with 16 repetitions:
263     if not skipflat:
264         EFF, filtEFF = dff_prepare_plan(flat_dset, 16, im_dark)
265
266     # Outfile shape can be determined only after first processing in ext FOV mode:
267     if (ext_fov):
268
269         # Read input sino:
270         idx = num_sinos / 2
271         im = tdf.read_sino(dset, idx).astype(float32)
272         im = extfov_correction(im, ext_fov, ext_fov_rot_right, ext_fov_overlap)
273
274         # Get the corrected outshape:
275         outshape = tdf.get_dset_shape(im.shape[1], num_sinos, im.shape[0])
276
277     else:
278         # Get the corrected outshape (in this case it's easy):
279         im = tdf.read_tomo(dset, 0).astype(float32)
280         outshape = tdf.get_dset_shape(im.shape[1], im.shape[0], num_proj)
281
282     # Create the output HDF5 file:
283     f_out = getHDF5(outfile, 'w')
284     f_out_dset = f_out.create_dataset('exchange/data', outshape, im.dtype)
285     f_out_dset.attrs['min'] = str(amin(im[:]))
286     f_out_dset.attrs['max'] = str(amax(im[:]))
287     f_out_dset.attrs['version'] = '1.0'
288     f_out_dset.attrs['axes'] = "y:theta:x"
289

```

```

290     f_out.close()
291     f_in.close()
292
293     # Log infos:
294     log = open(logfilename, "a")
295     log.write(linesep + "\tWork plan prepared correctly.")
296     log.write(linesep + "\t-----")
297     log.write(linesep + "\tPerforming pre processing...")
298     log.close()
299
300     # Run several threads for independent computation without waiting for threads_
↪ completion:
301     for num in range(nr_threads):
302         start = (num_sinos / nr_threads) * num
303         if (num == nr_threads - 1):
304             end = num_sinos - 1
305         else:
306             end = (num_sinos / nr_threads) * (num + 1) - 1
307         Process(target=_process, args=(lock, start, end, infile, outfile, outshape,
↪ im.dtype, skipflat, plan, norm_sx,
308             norm_dx, flat_end, half_half, half_half_line, ext_fov, ext_fov_rot_
↪ right, ext_fov_overlap, ringrem,
309             dynamic_ff, EFF, filtEFF, im_dark, logfilename)).start()
310
311
312     #start = int_from # 0
313     #end = int_to # num_sinos - 1
314     #_process(lock, start, end, infile, outfile, outshape, im.dtype, skipflat, plan,
↪ norm_sx,
315     #             norm_dx, flat_end, half_half, half_half_line, ext_fov, ext_fov_rot_
↪ right, ext_fov_overlap, ringrem,
316     #             dynamic_ff, EFF, filtEFF, im_dark, logfilename)
317
318     #255 256 C:\Temp\BrunGeorgos.tdf C:\Temp\BrunGeorgos_corr.tdf 0 0 True True 900_
↪ False False 0 rivers:11;0 False 1 C:\Temp\log_00.txt
319
320
321 if __name__ == "__main__":
322     main(argv[1:])

```

## exec\_reconstruct

This section contains the `exec_reconstruct` script.

Download file: [exec\\_reconstruct.py](#)

```

1 #####
2 # (C) 2016 Elettra - Sincrotrone Trieste S.C.p.A.. All rights reserved. #
3 # # #
4 # # #
5 # This file is part of STP-Core, the Python core of SYRMEP Tomo Project, #
6 # a software tool for the reconstruction of experimental CT datasets. #
7 # # #
8 # STP-Core is free software: you can redistribute it and/or modify it #
9 # under the terms of the GNU General Public License as published by the #
10 # Free Software Foundation, either version 3 of the License, or (at your #

```

```

11 # option) any later version. #
12 # #
13 # STP-Core is distributed in the hope that it will be useful, but WITHOUT #
14 # ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or #
15 # FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License #
16 # for more details. #
17 # #
18 # You should have received a copy of the GNU General Public License #
19 # along with STP-Core. If not, see <http://www.gnu.org/licenses/>. #
20 # #
21 #####
22
23 #
24 # Author: Francesco Brun
25 # Last modified: July, 8th 2016
26 #
27
28 # python:
29 from sys import argv, exit
30 from os import remove, sep, makedirs, linesep
31 from os.path import basename, exists
32 from numpy import finfo, copy, float32, double, amin, amax, tile, concatenate, log as_
↳nplog
33 from numpy import arange, meshgrid, isscalar, ndarray, pi, roll
34 from time import time
35 from multiprocessing import Process, Lock
36
37 # pystp-specific:
38 from stp_core.preprocess.extfov_correction import extfov_correction
39 from stp_core.preprocess.flat_fielding import flat_fielding
40 from stp_core.preprocess.ring_correction import ring_correction
41 from stp_core.preprocess.extract_flatdark import extract_flatdark, _medianize
42 from stp_core.preprocess.dynamic_flatfielding import dff_prepare_plan, dynamic_flat_
↳fielding
43
44 from stp_core.reconstruct.rec_astra import recon_astra_fbp, recon_astra_iterative
45 from stp_core.reconstruct.rec_fista_tv import recon_fista_tv
46 from stp_core.reconstruct.rec_mr_fbp import recon_mr_fbp
47 from stp_core.reconstruct.rec_gridrec import recon_gridrec
48
49 from stp_core.postprocess.postprocess import postprocess
50
51 from stp_core.utils.padding import upperPowerOfTwo, padImage, padSmoothWidth
52
53 from tifffile import imread, imsave
54 from h5py import File as getHDF5
55
56 # pystp-specific:
57 import stp_core.io.tdf as tdf
58
59
60 def reconstruct(im, angles, offset, logtransform, param1, circle, scale, pad, method,↳
↳rolling, roll_shift,
61                 zerone_mode, dset_min, dset_max, decim_factor, downsc_factor, corr_
↳offset):
62     """Reconstruct a sinogram with FBP algorithm (from ASTRA toolbox).
63
64     Parameters

```

```

65 -----
66 iml : array_like
67     Sinogram image data as numpy array.
68 center : float
69     Offset of the center of rotation to use for the tomographic
70     reconstruction with respect to the half of sinogram width
71     (default=0, i.e. half width).
72 logtransform : boolean
73     Apply logarithmic transformation before reconstruction (default=True).
74 filter : string
75     Filter to apply before the application of the reconstruction algorithm.
↪Filter
76     types are: ram-lak, shepp-logan, cosine, hamming, hann, tukey, lanczos,
↪triangular,
77     gaussian, barlett-hann, blackman, nuttall, blackman-harris, blackman-nuttall,
78     flat-top, kaiser, parzen.
79 circle : boolean
80     Create a circle in the reconstructed image and set to zero pixels outside the
81     circle (default=False).
82
83 Example (using tiff file.py)
84 -----
85 >>> # Read input (uncorrected) sinogram
86 >>> sino_iml = imread('sino_0050.tif')
87 >>>
88 >>> # Get flat and dark correction images:
89 >>> im_dark = medianize("\project\tomo", "dark*.tif")
90 >>> im_flat = medianize("\project\tomo", "flat*.tif")
91 >>>
92 >>> # Perform flat fielding and normalization:
93 >>> sino_im = normalize(sino_iml, (10,10), (0,0), im_dark, im_flat, 50)
94 >>>
95 >>> # Actual reconstruction:
96 >>> out = reconstruct_fbp(sino_im, -3.0)
97 >>>
98 >>> # Save output slice:
99 >>> imsave('slice_0050.tif', out)
100
101 """
102
103 # Copy images and ensure they are of type float32:
104 #im_f = copy(im.astype(float32))
105 im_f = im.astype(float32)
106
107 # Decimate projections if required:
108 if decim_factor > 1:
109     im_f = im_f[::decim_factor,:]
110
111 # Upscale projections (if required):
112 if (abs(scale - 1.0) > finfo(float32).eps):
113     siz_orig1 = im_f.shape[1]
114     im_f = imresize(im_f, (im_f.shape[0], int(round(scale * im_f.shape[1]))),
↪interp='bicubic', mode='F')
115     offset = int(offset * scale)
116
117 # Apply transformation for changes in the center of rotation:
118 if (offset != 0):
119     if (offset >= 0):

```



```

120         im_f = im_f[:, :-offset]
121
122         tmp = im_f[:,0] # Get first column
123         tmp = tile(tmp, (offset,1)) # Replicate the first column the right number_
↳of times
124         im_f = concatenate((tmp.T,im_f), axis=1) # Concatenate tmp before the_
↳image
125
126     else:
127         im_f = im_f[:,abs(offset):]
128
129         tmp = im_f[:,im_f.shape[1] - 1] # Get last column
130         tmp = tile(tmp, (abs(offset),1)) # Replicate the last column the right_
↳number of times
131         im_f = concatenate((im_f,tmp.T), axis=1) # Concatenate tmp after the image
132
133     # Downscale projections (without pixel averaging):
134     if downsc_factor > 1:
135         im_f = im_f[:,::downsc_factor]
136
137     # Sinogram rolling (if required). It doesn't make sense in limited angle_
↳tomography, so check if 180 or 360:
138     if ((rolling == True) and (roll_shift > 0)):
139         if ( (angles - pi) < finfo(float32).eps ):
140             # Flip the last rows:
141             im_f[-roll_shift:,:] = im_f[-roll_shift:,:-1]
142             # Now roll the sinogram:
143             im_f = roll(im_f, roll_shift, axis=0)
144         elif ((angles - pi*2.0) < finfo(float32).eps):
145             # Only roll the sinogram:
146             im_f = roll(im_f, roll_shift, axis=0)
147
148     # Scale image to [0,1] range (if required):
149     if (zerone_mode):
150
151         #print dset_min
152         #print dset_max
153         #print numpy.amin(im_f[:])
154         #print numpy.amax(im_f[:])
155         #im_f = (im_f - dset_min) / (dset_max - dset_min)
156
157         # Cheating the whole process:
158         im_f = (im_f - numpy.amin(im_f[:])) / (numpy.amax(im_f[:]) - numpy.amin(im_
↳f[:]))
159
160     # Apply log transform:
161     if (logtransform == True):
162         im_f[im_f <= finfo(float32).eps] = finfo(float32).eps
163         im_f = -nplog(im_f + corr_offset)
164
165     # Replicate pad image to double the width:
166     if (pad):
167
168         dim_o = im_f.shape[1]
169         n_pad = im_f.shape[1] + im_f.shape[1] / 2
170         marg = (n_pad - dim_o) / 2
171
172         # Pad image:

```

```

173     im_f = padSmoothWidth(im_f, n_pad)
174
175     # Perform the actual reconstruction:
176     if (method.startswith('FBP')):
177         im_f = recon_astra_fbp(im_f, angles, method, param1)
178     elif (method == 'MR-FBP_CUDA'):
179         im_f = recon_mr_fbp(im_f, angles)
180     elif (method == 'FISTA-TV_CUDA'):
181         im_f = recon_fista_tv(im_f, angles, param1, param1)
182     else:
183         im_f = recon_astra_iterative(im_f, angles, method, param1, zerone_mode)
184
185
186     # Crop:
187     if (pad):
188         im_f = im_f[marg:dim_o + marg, marg:dim_o + marg]
189
190     # Resize (if necessary):
191     if (abs(scale - 1.0) > finfo(float32).eps):
192         im_f = imresize(im_f, (siz_orig1, siz_orig1), interp='nearest', mode='F')
193
194     # Return output:
195     return im_f.astype(float32)
196
197 def reconstruct_gridrec(im1, im2, angles, offset, logtransform, param1, circle, scale,
198 ↪ pad, rolling, roll_shift,
199 ↪ zerone_mode, dset_min, dset_max, decim_factor, downsc_factor, corr_
200 ↪ offset):
201     """Reconstruct a sinogram with FBP algorithm (from ASTRA toolbox).
202
203     Parameters
204     -----
205     im1 : array_like
206         Sinogram image data as numpy array.
207     center : float
208         Offset of the center of rotation to use for the tomographic
209         reconstruction with respect to the half of sinogram width
210         (default=0, i.e. half width).
211     logtransform : boolean
212         Apply logarithmic transformation before reconstruction (default=True).
213     filter : string
214         Filter to apply before the application of the reconstruction algorithm.
215     ↪ Filter
216         types are: ram-lak, shepp-logan, cosine, hamming, hann, tukey, lanczos,
217     ↪ triangular,
218         gaussian, barlett-hann, blackman, nuttall, blackman-harris, blackman-nuttall,
219         flat-top, kaiser, parzen.
220     circle : boolean
221         Create a circle in the reconstructed image and set to zero pixels outside the
222         circle (default=False).
223
224     Example (using tiff file.py)
225     -----
226     >>> # Read input (uncorrected) sinogram
227     >>> sino_im1 = imread('sino_0050.tif')
228     >>>
229     >>> # Get flat and dark correction images:
230     >>> im_dark = medianize("\project\tomo", "dark*.tif")

```

```

227 >>> im_flat = medianize("\project\tomo", "flat*.tif")
228 >>>
229 >>> # Perform flat fielding and normalization:
230 >>> sino_im = normalize(sino_im1, (10,10), (0,0), im_dark, im_flat, 50)
231 >>>
232 >>> # Actual reconstruction:
233 >>> out = reconstruct_fbp(sino_im, -3.0)
234 >>>
235 >>> # Save output slice:
236 >>> imsave('slice_0050.tif', out)
237
238 """
239 # Ensure images are of type float32:
240 im_f1 = im1.astype(float32)
241 im_f2 = im2.astype(float32)
242
243 # Decimate projections if required:
244 if decim_factor > 1:
245     im_f1 = im_f1[::decim_factor,:]
246     im_f2 = im_f2[::decim_factor,:]
247
248 # Upscale projections (if required):
249 if (abs(scale - 1.0) > finfo(float32).eps):
250     siz_orig1 = im_f.shape[1]
251     im_f1 = imresize(im_f1, (im_f1.shape[0], int(round(scale * im_f1.shape[1]))),
↳interp='bicubic', mode='F')
252     im_f2 = imresize(im_f2, (im_f2.shape[0], int(round(scale * im_f2.shape[1]))),
↳interp='bicubic', mode='F')
253     offset = int(offset * scale)
254
255 # Apply transformation for changes in the center of rotation:
256 if (offset != 0):
257     if (offset >= 0):
258         im_f1 = im_f1[:, :-offset]
259
260         tmp = im_f1[:,0] # Get first column
261         tmp = tile(tmp, (offset,1)) # Replicate the first column the right number
↳of times
262         im_f1 = concatenate((tmp.T, im_f1), axis=1) # Concatenate tmp before the
↳image
263
264         im_f2 = im_f2[:, :-offset]
265
266         tmp = im_f2[:,0] # Get first column
267         tmp = tile(tmp, (offset,1)) # Replicate the first column the right number
↳of times
268         im_f2 = concatenate((tmp.T, im_f2), axis=1) # Concatenate tmp before the
↳image
269
270     else:
271         im_f1 = im_f1[:, abs(offset):]
272
273         tmp = im_f1[:, im_f1.shape[1] - 1] # Get last column
274         tmp = tile(tmp, (abs(offset),1)) # Replicate the last column the right
↳number of times
275         im_f1 = concatenate((im_f1, tmp.T), axis=1) # Concatenate tmp after the
↳image
276

```

```

277         im_f2 = im_f2[:,abs(offset):]
278
279         tmp = im_f2[:,im_f2.shape[1] - 1] # Get last column
280         tmp = tile(tmp, (abs(offset),1)) # Replicate the last column the right_
↪number of times
281         im_f2 = concatenate((im_f2,tmp.T), axis=1) # Concatenate tmp after the_
↪image
282
283     # Downscale projections (without pixel averaging):
284     if downsc_factor > 1:
285         im_f1 = im_f1[:,::downsc_factor]
286         im_f2 = im_f2[:,::downsc_factor]
287
288     # Sinogram rolling (if required). It doesn't make sense in limited angle_
↪tomography, so check if 180 or 360:
289     if ((rolling == True) and (roll_shift > 0)):
290         if ( (angles - pi) < finfo(float32).eps ):
291             # Flip the last rows:
292             im_f1[-roll_shift:,:] = im_f1[-roll_shift::-1]
293             im_f2[-roll_shift:,:] = im_f2[-roll_shift::-1]
294             # Now roll the sinogram:
295             im_f1 = roll(im_f1, roll_shift, axis=0)
296             im_f2 = roll(im_f2, roll_shift, axis=0)
297         elif ((angles - pi*2.0) < finfo(float32).eps):
298             # Only roll the sinogram:
299             im_f1 = roll(im_f1, roll_shift, axis=0)
300             im_f2 = roll(im_f2, roll_shift, axis=0)
301
302     # Scale image to [0,1] range (if required):
303     if (zerone_mode):
304
305         #print dset_min
306         #print dset_max
307         #print numpy.amin(im_f[:])
308         #print numpy.amax(im_f[:])
309         #im_f = (im_f - dset_min) / (dset_max - dset_min)
310
311         # Cheating the whole process:
312         im_f1 = (im_f1 - numpy.amin(im_f1[:])) / (numpy.amax(im_f1[:]) - numpy.
↪amin(im_f1[:]))
313         im_f2 = (im_f2 - numpy.amin(im_f2[:])) / (numpy.amax(im_f2[:]) - numpy.
↪amin(im_f2[:]))
314
315
316     # Apply log transform:
317     if (logtransform == True):
318         im_f1[im_f1 <= finfo(float32).eps] = finfo(float32).eps
319         im_f1 = -nplog(im_f1 + corr_offset)
320
321         im_f2[im_f2 <= finfo(float32).eps] = finfo(float32).eps
322         im_f2 = -nplog(im_f2 + corr_offset)
323
324     # Replicate pad image to double the width:
325     if (pad):
326
327         dim_o = im_f1.shape[1]
328         n_pad = im_f1.shape[1] + im_f1.shape[1] / 2
329         marg = (n_pad - dim_o) / 2

```

```

330
331     # Pad image:
332     im_f1 = padSmoothWidth(im_f1, n_pad)
333     im_f2 = padSmoothWidth(im_f2, n_pad)
334
335     # Perform the actual reconstruction:
336     [im_f1, im_f2] = recon_gridrec(im_f1, im_f2, angles, param1)
337
338
339     # Crop:
340     if (pad):
341         im_f1 = im_f1[marg:dim_o + marg, marg:dim_o + marg]
342         im_f2 = im_f2[marg:dim_o + marg, marg:dim_o + marg]
343
344     # Resize (if necessary):
345     if (abs(scale - 1.0) > finfo(float32).eps):
346         im_f1 = imresize(im_f1, (siz_orig1, siz_orig1), interp='nearest', mode='F')
347         im_f2 = imresize(im_f2, (siz_orig1, siz_orig1), interp='nearest', mode='F')
348
349     # Return output:
350     return [im_f1.astype(float32), im_f2.astype(float32)]
351
352 def write_log(lock, fname, logfilename, cputime, iotime):
353     """To do...
354
355     """
356     lock.acquire()
357     try:
358         # Print out execution time:
359         log = open(logfilename, "a")
360         log.write(linesep + "\t%s reconstructed (CPU: %0.3f sec - I/O: %0.3f sec)." %
↪ (basename(fname), cputime, iotime))
361         log.close()
362
363     finally:
364         lock.release()
365
366 def write_log_gridrec(lock, fname1, fname2, logfilename, cputime, iotime):
367     """To do...
368
369     """
370     lock.acquire()
371     try:
372         # Print out execution time:
373         log = open(logfilename, "a")
374         log.write(linesep + "\t%s reconstructed (CPU: %0.3f sec - I/O: %0.3f sec)." %
↪ (basename(fname1), cputime/2, iotime/2))
375         log.write(linesep + "\t%s reconstructed (CPU: %0.3f sec - I/O: %0.3f sec)." %
↪ (basename(fname2), cputime/2, iotime/2))
376         log.close()
377
378     finally:
379         lock.release()
380
381 def process_gridrec(lock, int_from, int_to, num_sinos, infile, outpath, preprocessing_
↪ required, skipflat, corr_plan,
382                 norm_sx, norm_dx, flat_end, half_half,
383                 half_half_line, ext_fov, ext_fov_rot_right, ext_fov_overlap, ringrem,
↪ angles, angles_projfrom, angles_projto,

```

```

384     offset, logtransform, param1, circle, scale, pad, rolling, roll_shift,
↪zerone_mode, dset_min, dset_max, decim_factor,
385     downsc_factor, corr_offset, postprocess_required, convert_opt, crop_opt,
↪dynamic_ff, EFF, filtEFF, im_dark,
386     outprefix, logfilename):
387     """To do...
388
389     """
390     # Process the required subset of images:
391     for i in range(int_from, int_to + 1, 2):
392
393         # Read two sinograms:
394         t0 = time()
395         f_in = getHDF5(infile, 'r')
396         if "/tomo" in f_in:
397             dset = f_in['tomo']
398         else:
399             dset = f_in['exchange/data']
400         im1 = tdf.read_sino(dset,i).astype(float32)
401         if (i + 1) <= (int_to + 1) ):
402             im2 = tdf.read_sino(dset,i + 1).astype(float32)
403         else:
404             im2 = im1
405         f_in.close()
406         t1 = time()
407
408
409         # Apply projection removal (if required):
410         im1 = im1[angles_projfrom:angles_projto, :]
411         im2 = im2[angles_projfrom:angles_projto, :]
412
413         # Perform the preprocessing of the sinograms (if required):
414         if (preprocessing_required):
415             if not skipflat:
416                 if dynamic_ff:
417                     # Dynamic flat fielding with downsampling = 2:
418                     im1 = dynamic_flat_fielding(im1, i, EFF, filtEFF, 2, im_dark,
↪norm_sx, norm_dx)
419                 else:
420                     im1 = flat_fielding (im1, i, corr_plan, flat_end, half_half, half_
↪half_line, norm_sx, norm_dx).astype(float32)
421                     im1 = extfov_correction (im1, ext_fov, ext_fov_rot_right, ext_fov_overlap)
422                     if not skipflat:
423                         im1 = ring_correction (im1, ringrem, flat_end, corr_plan['skip_flat_
↪after'], half_half, half_half_line, ext_fov)
424                     else:
425                         im1 = ring_correction (im1, ringrem, False, False, half_half, half_
↪half_line, ext_fov)
426
427                 if not skipflat:
428                     if dynamic_ff:
429                         # Dynamic flat fielding with downsampling = 2:
430                         im2 = dynamic_flat_fielding(im2, i, EFF, filtEFF, 2, im_dark,
↪norm_sx, norm_dx)
431                     else:
432                         im2 = flat_fielding (im2, i + 1, corr_plan, flat_end, half_half,
↪half_half_line, norm_sx, norm_dx).astype(float32)
433                         im2 = extfov_correction (im2, ext_fov, ext_fov_rot_right, ext_fov_overlap)

```

```

434         if not skipflat and not dynamic_ff:
435             im2 = ring_correction (im2, ringrem, flat_end, corr_plan['skip_flat_
↳after'], half_half, half_half_line, ext_fov)
436         else:
437             im2 = ring_correction (im2, ringrem, False, False, half_half, half_
↳half_line, ext_fov)
438
439
440         # Actual reconstruction:
441         [im1, im2] = reconstruct_gridrec(im1, im2, angles, offset, logtransform,
↳param1, circle, scale, pad, rolling, roll_shift,
442             zerone_mode, dset_min, dset_max, decim_factor, downsc_factor,
↳corr_offset)
443
444         # Appy post-processing (if required):
445         if postprocess_required:
446             im1 = postprocess(im1, convert_opt, crop_opt, circle)
447             im2 = postprocess(im2, convert_opt, crop_opt, circle)
448         else:
449             # Create the circle mask for fancy output:
450             if (circle == True):
451                 siz = im1.shape[1]
452                 if siz % 2:
453                     rang = arange(-siz / 2 + 1, siz / 2 + 1)
454                 else:
455                     rang = arange(-siz / 2, siz / 2)
456                 x,y = meshgrid(rang,rang)
457                 z = x ** 2 + y ** 2
458                 a = (z < (siz / 2 - int(round(abs(offset)/downsc_factor)) ) ** 2)
459
460                 im1 = im1 * a
461                 im2 = im2 * a
462
463         # Write down reconstructed slices:
464         t2 = time()
465
466         fname1 = outpath + outprefix + '_' + str(i).zfill(4) + '.tif'
467         imsave(fname1, im1)
468
469         fname2 = outpath + outprefix + '_' + str(i + 1).zfill(4) + '.tif'
470         imsave(fname2, im2)
471
472         t3 = time()
473
474         # Write log (atomic procedure - lock used):
475         write_log_gridrec(lock, fname1, fname2, logfilename, t2 - t1, (t3 - t2) + (t1
↳- t0) )
476
477
478 def process(lock, int_from, int_to, num_sinos, infile, outpath, preprocessing_
↳required, skipflat, corr_plan, norm_sx, norm_dx,
479     flat_end, half_half,
480     half_half_line, ext_fov, ext_fov_rot_right, ext_fov_overlap, ringrem,
↳
481     angles, angles_projfrom, angles_projto,
482     offset, logtransform, param1, circle, scale, pad, method, rolling, roll_
↳shift, zerone_mode, dset_min, dset_max, decim_factor,
483     downsc_factor, corr_offset, postprocess_required, convert_opt, crop_opt,
↳
484     dynamic_ff, EFF, filtEFF, im_dark,

```

```

483     outprefix, logfilename):
484     """To do...
485
486     """
487     # Process the required subset of images:
488     for i in range(int_from, int_to + 1):
489
490         # Perform reconstruction (on-the-fly preprocessing and phase retrieval, if_
↳required):
491         #if (phaseretrieval_required):
492
493         # # Load into memory a bunch of sinograms:
494         # t0 = time()
495
496         # # Open the TDF file for reading:
497         # f_in = getHDF5(infile, 'r')
498         # if "/tomo" in f_in:
499         #     dset = f_in['tomo']
500         # else:
501         #     dset = f_in['exchange/data']
502
503         # # Prepare the data structure according to the approximation window:
504         # tmp_im = numpy.empty((tdf.get_nr_projs(dset),tdf.get_det_size(dset),
↳approx_win), dtype=float32)
505
506         # # Load the temporary data structure reading the input TDF file:
507         # # (It can be parallelized Open-MP style)
508         # ct = 0
509         # for j in range(i - approx_win/2, i + approx_win/2 + 1):
510         #     if (j < 0):
511         #         j = 0
512         #     if (j >= num_sinos):
513         #         j = num_sinos - 1
514         #     a = tdf.read_sino(dset,j).astype(float32)
515         #     tmp_im[:, :, ct] = a
516         #     ct = ct + 1
517
518         # # Close the TDF file:
519         # f_in.close()
520         # t1 = time()
521
522         # # Perform the processing:
523         # if (preprocessing_required):
524         #     ct = 0
525         #     # (It can be parallelized Open-MP style)
526         #     for j in range(i - approx_win/2, i + approx_win/2 + 1):
527         #         if (j < 0):
528         #             j = 0
529         #         if (j >= num_sinos):
530         #             j = num_sinos - 1
531
532         #         tmp_im[:, :, ct] = flat_fielding (tmp_im[:, :, ct], j, corr_plan,
↳
↳flat_end, half_half, half_half_line, norm_sx, norm_dx).astype(float32)
533         #         tmp_im[:, :, ct] = extfov_correction (tmp_im[:, :, ct], ext_fov, ext_
↳
↳fov_rot_right, ext_fov_overlap).astype(float32)
534         #         tmp_im[:, :, ct] = ring_correction (tmp_im[:, :, ct], ringrem, flat_
↳
↳end, corr_plan['skip_flat_after'], half_half, half_half_line, ext_fov).
↳
↳astype(float32)

```



```

535         #             ct = ct + 1
536
537         # # Perform phase retrieval:
538         # # (It can be parallelized Open-MP style)
539         # for ct in range(0, tmp_im.shape[0]):
540
541         #         tmp_im[ct,:,:] = phase_retrieval(tmp_im[ct,:,:].T, phrt_plan).
↪astype(float32).T
542         #             ct = ct + 1
543
544         # # Extract the central processed sinogram:
545         # im = tmp_im[:, :, approx_win/2]
546
547         #else:
548
549         # Read only one sinogram:
550         t0 = time()
551         f_in = getHDF5(infile, 'r')
552         if "/tomo" in f_in:
553             dset = f_in['tomo']
554         else:
555             dset = f_in['exchange/data']
556         im = tdf.read_sino(dset, i).astype(float32)
557         f_in.close()
558         t1 = time()
559
560         # Apply projection removal (if required):
561         im = im[angles_projfrom:angles_projto, :]
562
563         # Perform the preprocessing of the sinogram (if required):
564         if (preprocessing_required):
565             if not skipflat:
566                 if dynamic_ff:
567                     # Dynamic flat fielding with downsampling = 2:
568                     im = dynamic_flat_fielding(im, i, EFF, filtEFF, 2, im_dark, norm_
↪sx, norm_dx).astype(float32)
569                 else:
570                     im = flat_fielding(im, i, corr_plan, flat_end, half_half, half_
↪half_line, norm_sx, norm_dx).astype(float32)
571                     im = extfov_correction(im, ext_fov, ext_fov_rot_right, ext_fov_overlap)
572                     if not skipflat and not dynamic_ff:
573                         im = ring_correction(im, ringrem, flat_end, corr_plan['skip_flat_
↪after'], half_half, half_half_line, ext_fov)
574                     else:
575                         im = ring_correction(im, ringrem, False, False, half_half, half_half_
↪line, ext_fov)
576
577
578         # Actual reconstruction:
579         im = reconstruct(im, angles, offset, logtransform, param1, circle, scale, pad,
↪method, rolling, roll_shift,
580                          zerone_mode, dset_min, dset_max, decim_factor, downsc_factor,
↪corr_offset).astype(float32)
581
582         # Apply post-processing (if required):
583         if postprocess_required:
584             im = postprocess(im, convert_opt, crop_opt)
585         else:

```

```

586     # Create the circle mask for fancy output:
587     if (circle == True):
588         siz = im.shape[1]
589         if siz % 2:
590             rang = arange(-siz / 2 + 1, siz / 2 + 1)
591         else:
592             rang = arange(-siz / 2, siz / 2)
593         x,y = meshgrid(rang,rang)
594         z = x ** 2 + y ** 2
595         a = (z < (siz / 2 - abs(offset) ) ** 2)
596         im = im * a
597
598     # Write down reconstructed slice:
599     t2 = time()
600     fname = outpath + outprefix + '_' + str(i).zfill(4) + '.tif'
601     imsave(fname, im)
602     t3 = time()
603
604     # Write log (atomic procedure - lock used):
605     write_log(lock, fname, logfilename, t2 - t1, (t3 - t2) + (t1 - t0) )
606
607
608 def main(argv):
609     """To do...
610
611     Usage
612     ----
613
614     Parameters
615     -----
616
617     Example
618     -----
619
620     The following line processes the first ten TIFF files of input path
621     "/home/in" and saves the processed files to "/home/out" with the
622     application of the Boin and Haibel filter with smoothing via a Butterworth
623     filter of order 4 and cutoff frequency 0.01:
624
625     reconstruct 0 4 C:\Temp\Dullin_Aug_2012\sino_noflat C:\Temp\Dullin_Aug_2012\sino_
626     ↪noflat\output
627     9.0 10.0 0.0 0.0 0.0 true sino slice C:\Temp\Dullin_Aug_2012\sino_noflat\tomo_
628     ↪conv flat dark
629
630     """
631     lock = Lock()
632     skip_flat = False
633     skip_flat_after = True
634
635     # Get the from and to number of files to process:
636     int_from = int(argv[0])
637     int_to = int(argv[1])
638
639     # Get paths:
640     infile = argv[2]
641     outpath = argv[3]
642
643     # Essential reconstruction parameters:

```

```

642 angles = float(argv[4])
643 offset = float(argv[5])
644 param1 = argv[6]
645 scale = int(float(argv[7]))
646
647 overpad = True if argv[8] == "True" else False
648 logtrsf = True if argv[9] == "True" else False
649 circle = True if argv[10] == "True" else False
650
651 outprefix = argv[11]
652
653 # Parameters for on-the-fly pre-processing:
654 preprocessing_required = True if argv[12] == "True" else False
655 flat_end = True if argv[13] == "True" else False
656 half_half = True if argv[14] == "True" else False
657
658 half_half_line = int(argv[15])
659
660 ext_fov = True if argv[16] == "True" else False
661
662 norm_sx = int(argv[19])
663 norm_dx = int(argv[20])
664
665 ext_fov_rot_right = argv[17]
666 if ext_fov_rot_right == "True":
667     ext_fov_rot_right = True
668     if (ext_fov):
669         norm_sx = 0
670 else:
671     ext_fov_rot_right = False
672     if (ext_fov):
673         norm_dx = 0
674
675 ext_fov_overlap = int(argv[18])
676
677 skip_ringrem = True if argv[21] == "True" else False
678 ringrem = argv[22]
679
680 # Extra reconstruction parameters:
681 zerone_mode = True if argv[23] == "True" else False
682 corr_offset = float(argv[24])
683
684 reconmethod = argv[25]
685
686 decim_factor = int(argv[26])
687 downsc_factor = int(argv[27])
688
689 # Parameters for postprocessing:
690 postprocess_required = True if argv[28] == "True" else False
691 convert_opt = argv[29]
692 crop_opt = argv[30]
693
694 angles_projfrom = int(argv[31])
695 angles_projto = int(argv[32])
696
697 rolling = True if argv[33] == "True" else False
698 roll_shift = int(argv[34])
699

```

```

700     dynamic_ff = True if argv[35] == "True" else False
701
702     nr_threads = int(argv[36])
703     logfilename = argv[37]
704     process_id = int(logfilename[-6:-4])
705
706     # Check prefixes and path:
707     #if not infile.endswith(sep): infile += sep
708     if not exists(outputpath):
709         makedirs(outputpath)
710
711     if not outputpath.endswith(sep): outputpath += sep
712
713     # Open the HDF5 file:
714     f_in = getHDF5(infile, 'r')
715     if "/tomo" in f_in:
716         dset = f_in['tomo']
717
718         tomoprefix = 'tomo'
719         flatprefix = 'flat'
720         darkprefix = 'dark'
721     else:
722         dset = f_in['exchange/data']
723         if "/provenance/detector_output" in f_in:
724             prov_dset = f_in['provenance/detector_output']
725
726             tomoprefix = prov_dset.attrs['tomo_prefix']
727             flatprefix = prov_dset.attrs['flat_prefix']
728             darkprefix = prov_dset.attrs['dark_prefix']
729
730     dset_min = -1
731     dset_max = -1
732     if (zerone_mode):
733         if ('min' in dset.attrs):
734             dset_min = float(dset.attrs['min'])
735         else:
736             zerone_mode = False
737
738         if ('max' in dset.attrs):
739             dset_max = float(dset.attrs['max'])
740         else:
741             zerone_mode = False
742
743     num_sinos = tdf.get_nr_sinos(dset) # Pay attention to the downscale factor
744
745     if (num_sinos == 0):
746         log = open(logfilename, "a")
747         log.write(linesep + "\tNo projections found. Process will end.")
748         log.close()
749         exit()
750
751     # Check extrema (int_to == -1 means all files):
752     if ((int_to >= num_sinos) or (int_to == -1)):
753         int_to = num_sinos - 1
754
755     # Log info:
756     log = open(logfilename, "w")
757     log.write(linesep + "\tInput file: %s" % (infile))

```

```

758 log.write(lineseq + "\tOutput path: %s" % (outpath))
759 log.write(lineseq + "\t-----")
760 log.write(lineseq + "\tPreparing the work plan...")
761 log.close()
762
763 # Get correction plan and phase retrieval plan (if required):
764 corrplan = -1
765 phrtplan = -1
766
767 skipflat = False
768
769 im_dark = -1
770 EFF = -1
771 filtEFF = -1
772 if (preprocessing_required):
773     if not dynamic_ff:
774         # Load flat fielding plan either from cache (if required) or from TDF_
↪file and cache it for faster re-use:
775         corrplan = extract_flatdark(f_in, flat_end, logfilename)
776         if (isscalar(corrplan['im_flat']) and isscalar(corrplan['im_flat_after']))_
↪):
777             skipflat = True
778
779 # Downscale flat and dark images if necessary:
780 if isinstance(corrplan['im_flat'], ndarray):
781     corrplan['im_flat'] = corrplan['im_flat'][:,::downsc_factor,::downsc_
↪factor]
782 if isinstance(corrplan['im_dark'], ndarray):
783     corrplan['im_dark'] = corrplan['im_dark'][:,::downsc_factor,::downsc_
↪factor]
784 if isinstance(corrplan['im_flat_after'], ndarray):
785     corrplan['im_flat_after'] = corrplan['im_flat_after'][:,::downsc_factor,
↪::downsc_factor]
786 if isinstance(corrplan['im_dark_after'], ndarray):
787     corrplan['im_dark_after'] = corrplan['im_dark_after'][:,::downsc_factor,
↪::downsc_factor]
788
789 else:
790     # Dynamic flat fielding:
791     if "/tomo" in f_in:
792         if "/flat" in f_in:
793             flat_dset = f_in['flat']
794             if "/dark" in f_in:
795                 im_dark = _medianize(f_in['dark'])
796             else:
797                 skipdark = True
798         else:
799             skipflat = True # Nothing to do in this case
800     else:
801         if "/exchange/data_white" in f_in:
802             flat_dset = f_in['/exchange/data_white']
803             if "/exchange/data_dark" in f_in:
804                 im_dark = _medianize(f_in['/exchange/data_dark'])
805             else:
806                 skipdark = True
807         else:
808             skipflat = True # Nothing to do in this case
809

```

```

810     # Prepare plan for dynamic flat fielding with 16 repetitions:
811     if not skipflat:
812         EFF, filtEFF = dff_prepare_plan(flat_dset, 16, im_dark)
813
814         # Downscale images if necessary:
815         im_dark = im_dark[:,::downsc_factor,::downsc_factor]
816         EFF = EFF[:,::downsc_factor,::downsc_factor,:]
817         filtEFF = filtEFF[:,::downsc_factor,::downsc_factor,:]
818
819     f_in.close()
820
821     # Log infos:
822     log = open(logfilename,"a")
823     log.write(linesep + "\tWork plan prepared correctly.")
824     log.write(linesep + "\t-----")
825     log.write(linesep + "\tPerforming reconstruction...")
826     log.close()
827
828     # Run several threads for independent computation without waiting for threads_
829     ↪completion:
830     for num in range(nr_threads):
831         start = ( (int_to - int_from + 1) / nr_threads)*num + int_from
832         if (num == nr_threads - 1):
833             end = int_to
834         else:
835             end = ( (int_to - int_from + 1) / nr_threads)*(num + 1) + int_from - 1
836         if (reconmethod == 'GRIDREC'):
837             ↪Process(target=process_gridrec, args=(lock, start, end, num_sinos, infile,
838             ↪outpath, preprocessing_required, skipflat,
839             ↪corrplan, norm_sx, norm_dx, flat_end, half_half, half_half_
840             ↪line, ext_fov, ext_fov_rot_right,
841             ↪ext_fov_overlap, ringrem,
842             ↪angles, angles_projfrom, angles_projto, offset, logtrsf,
843             ↪param1, circle, scale, overpad,
844             ↪rolling, roll_shift,
845             ↪zerone_mode, dset_min, dset_max, decim_factor, downsc_factor,
846             ↪corr_offset,
847             ↪postprocess_required, convert_opt, crop_opt, dynamic_ff, EFF,
848             ↪filtEFF, im_dark, outprefix,
849             ↪logfilename)).start()
850         else:
851             ↪Process(target=process, args=(lock, start, end, num_sinos, infile,
852             ↪outpath, preprocessing_required, skipflat,
853             ↪corrplan, norm_sx,
854             ↪norm_dx, flat_end, half_half, half_half_line, ext_fov, ext_
855             ↪fov_rot_right, ext_fov_overlap, ringrem,
856             ↪angles, angles_projfrom, angles_projto, offset, logtrsf,
857             ↪param1, circle, scale, overpad,
858             ↪reconmethod, rolling, roll_shift,
859             ↪zerone_mode, dset_min, dset_max, decim_factor, downsc_factor,
860             ↪corr_offset,
861             ↪postprocess_required, convert_opt, crop_opt, dynamic_ff, EFF,
862             ↪filtEFF, im_dark, outprefix,
863             ↪logfilename)).start()
864
865     #start = int_from
866     #end = int_to
867     #if (reconmethod == 'GRIDREC'):

```

```

857 # process_gridrec(lock, start, end, num_sinos, infile, outpath, preprocessing_
↳required, skipflat, corrplan, norm_sx,
858 # norm_dx, flat_end, half_half, half_half_line, ext_fov, ext_
↳fov_rot_right, ext_fov_overlap, ringrem,
859 # angles, angles_projfrom, angles_projto, offset, logtrsf,
↳param1, circle, scale, overpad,
860 # rolling, roll_shift,
861 # zerone_mode, dset_min, dset_max, decim_factor, downsc_factor,
↳corr_offset,
862 # postprocess_required, convert_opt, crop_opt, dynamic_ff, EFF,
↳filtEFF, im_dark, outprefix, logfilename)
863 #else:
864 # process(lock, start, end, num_sinos, infile, outpath, preprocessing_required,
↳skipflat, corrplan, norm_sx,
865 # norm_dx, flat_end, half_half, half_half_line, ext_fov, ext_
↳fov_rot_right, ext_fov_overlap, ringrem,
866 # angles, angles_projfrom, angles_projto, offset, logtrsf,
↳param1, circle, scale, overpad,
867 # reconmethod, rolling, roll_shift, zerone_mode, dset_min, dset_
↳max, decim_factor, downsc_factor, corr_offset,
868 # postprocess_required, convert_opt, crop_opt, dynamic_ff, EFF,
↳filtEFF, im_dark, outprefix, logfilename)
869
870 # Example:
871 # 255 255 C:\Temp\BrunGeorgos.tdf C:\Temp\BrunGeorgos 3.1416 -31.0 shepp-logan 1.
↳0 False False True slice True True True 5 False False 100 0 0 False rivers:11;0
↳False 0.0 FBP_CUDA 1 1 False - - 0 1799 False 2 C:\Temp\log_00.txt
872
873
874 if __name__ == "__main__":
875     main(argv[1:])

```

## exec\_postprocessing

This section contains the exec\_postprocessing script.

Download file: [exec\\_postprocessing.py](#)

```

1 #####
2 # (C) 2016 Elettra - Sincrotrone Trieste S.C.p.A.. All rights reserved. #
3 # # #
4 # # #
5 # This file is part of STP-Core, the Python core of SYRMEP Tomo Project, #
6 # a software tool for the reconstruction of experimental CT datasets. #
7 # # #
8 # STP-Core is free software: you can redistribute it and/or modify it #
9 # under the terms of the GNU General Public License as published by the #
10 # Free Software Foundation, either version 3 of the License, or (at your #
11 # option) any later version. #
12 # # #
13 # STP-Core is distributed in the hope that it will be useful, but WITHOUT #
14 # ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or #
15 # FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License #
16 # for more details. #
17 # # #
18 # You should have received a copy of the GNU General Public License #
19 # along with STP-Core. If not, see <http://www.gnu.org/licenses/>. #

```

```

20 #
21 #####
22
23 #
24 # Author: Francesco Brun
25 # Last modified: July, 8th 2016
26 #
27
28 from sys import argv, exit
29 from glob import glob
30 from os import linesep
31 from os.path import sep, basename, exists
32 from time import time
33 from multiprocessing import Process, Lock
34
35 # pystp-specific:
36 from stp_core.postprocess.postprocess import postprocess
37
38 from tiffiff import imread, imsave
39
40 def _write_log(lock, fname, logfilename, cputime, iotime):
41
42     lock.acquire()
43     try:
44         # Print out execution time:
45         log = open(logfilename, "a")
46         log.write(linesep + "\t%s processed (CPU: %0.3f sec - I/O: %0.3f sec)." % (
↳ (basename(fname), cputime, iotime))
47         log.close()
48
49     finally:
50         lock.release()
51
52 def _process(lock, int_from, int_to, files, outpath, convert_opt, crop_opt, outprefix,
↳ logfilename):
53
54     # Process the required subset of images:
55     for i in range(int_from, int_to + 1):
56
57         # Read i-th slice:
58         t0 = time()
59         im = imread(files[i])
60         t1 = time()
61
62         # Post process the image:
63         im = postprocess(im, convert_opt, crop_opt)
64
65         # Write down post-processed slice:
66         t2 = time()
67         fname = outpath + outprefix + '_' + str(i).zfill(4) + '.tif'
68         imsave(fname, im)
69         t3 = time()
70
71         # Write log (atomic procedure - lock used):
72         _write_log(lock, fname, logfilename, t2 - t1, (t3 - t2) + (t1 - t0) )
73
74
75 def main(argv):

```



```

76     """To do...
77
78     Usage
79     -----
80
81
82     Parameters
83     -----
84
85     Example
86     -----
87     The following line processes the first ten TIFF files of input path
88     "/home/in" and saves the processed files to "/home/out" with the
89     application of the Boin and Haibel filter with smoothing via a Butterworth
90     filter of order 4 and cutoff frequency 0.01:
91
92     reconstruct 0 4 C:\Temp\Dullin_Aug_2012\sino_noflat C:\Temp\Dullin_Aug_2012\sino_
↪noflat\output
93     9.0 10.0 0.0 0.0 0.0 true sino slice C:\Temp\Dullin_Aug_2012\sino_noflat\tomo_
↪conv flat dark
94
95     """
96     lock = Lock()
97     # Get the from and to number of files to process:
98     int_from = int(argv[0])
99     int_to = int(argv[1])
100
101     # Get input and output paths:
102     inpath = argv[2]
103     outpath = argv[3]
104
105     if not inpath.endswith(sep): inpath += sep
106     if not outpath.endswith(sep): outpath += sep
107
108     # Get parameters:
109     convert_opt = argv[4]
110     crop_opt = argv[5]
111
112     outprefix = argv[6]
113
114     # Number of threads to use and logfile:
115     nr_threads = int(argv[7])
116     logfilename = argv[8]
117
118     # Get the files in infile:
119     log = open(logfilename, "w")
120     log.write(linesep + "\tInput TIFF folder: %s" % (inpath))
121     log.write(linesep + "\tOutput TIFF folder: %s" % (outpath))
122     log.write(linesep + "\t-----")
123     if (int_to != -1):
124         log.write(linesep + "\tThe subset [%d,%d] of the input files will be_
↪considered." % (int_from, int_to))
125         log.write(linesep + "\tCropping:")
126         crop_opt_num = crop_opt.split(":")
127         log.write(linesep + "\t\tTop: %s pixels" % (crop_opt_num[0]))
128         log.write(linesep + "\t\tBottom: %s pixels" % (crop_opt_num[1]))
129         log.write(linesep + "\t\tLeft: %s pixels" % (crop_opt_num[2]))
130         log.write(linesep + "\t\tRight: %s pixels" % (crop_opt_num[3]))

```

```

131 conv_method, conv_args = convert_opt.split(":", 1)
132 if (conv_method == "linear8"):
133     min, max = conv_args.split(";")
134     log.write(linesep + "\tConversion to 8-bit by remapping range [%s,%s] to [0,
↳255]." % (min, max))
135     elif (conv_method == "linear"):
136         min, max = conv_args.split(";")
137         log.write(linesep + "\tConversion to 16-bit by remapping range [%s,%s] to [0,
↳65535]." % (min, max))
138         log.write(linesep + "\t-----")
139         log.write(linesep + "\tBrowsing input folder...")
140         log.close()
141
142         files = sorted(glob(inpath + '*.tif*'))
143         num_files = len(files)
144
145         if ((int_to >= num_files) or (int_to == -1)):
146             int_to = num_files - 1
147
148         # Log infos:
149         log = open(logfilename, "a")
150         log.write(linesep + "\tInput folder browsed correctly.")
151         log.close()
152
153         # Run several threads for independent computation without waiting for threads_
↳completion:
154         for num in range(nr_threads):
155             start = ( (int_to - int_from + 1) / nr_threads)*num + int_from
156             if (num == nr_threads - 1):
157                 end = int_to
158             else:
159                 end = ( (int_to - int_from + 1) / nr_threads)*(num + 1) + int_from - 1
160             Process(target=_process, args=(lock, start, end, files, outpath, convert_opt,
↳crop_opt, outprefix, logfilename )).start()
161
162             #start = 0
163             #end = num_files - 1
164             #process(lock, start, end, files, outpath, convert_opt, crop_opt, outprefix,
↳logfilename )
165
166             #0 -1 C:\Temp\BrunGeorgos C:\Temp\BrunGeorgos\slice_8 linear8:-0.01;0.01
↳10:10:10:20 slice C:\Temp\log_00_conv.txt
167
168
169 if __name__ == "__main__":
170     main(argv[1:])

```

## exec\_phaseretrieval

This section contains the `exec_phaseretrieval` script.

Download file: `exec_phaseretrieval.py`

```

1 #####
2 # (C) 2016 Elettra - Sincrotrone Trieste S.C.p.A.. All rights reserved. #
3 # #
4 # #

```

```

5 # This file is part of STP-Core, the Python core of SYRMEP Tomo Project, #
6 # a software tool for the reconstruction of experimental CT datasets. #
7 # #
8 # STP-Core is free software: you can redistribute it and/or modify it #
9 # under the terms of the GNU General Public License as published by the #
10 # Free Software Foundation, either version 3 of the License, or (at your #
11 # option) any later version. #
12 # #
13 # STP-Core is distributed in the hope that it will be useful, but WITHOUT #
14 # ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or #
15 # FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License #
16 # for more details. #
17 # #
18 # You should have received a copy of the GNU General Public License #
19 # along with STP-Core. If not, see <http://www.gnu.org/licenses/>. #
20 # #
21 #####
22 #
23 #
24 # Author: Francesco Brun
25 # Last modified: July, 8th 2016
26 #
27
28 from sys import argv, exit
29 from os import remove, sep, linesep
30 from os.path import exists
31 from numpy import float32, double, amin, amax
32 from time import time
33 from multiprocessing import Process, Lock
34 from pyfftw.interfaces.cache import enable as pyfftw_cache_enable, disable as pyfftw_
35 ↪cache_disable
36 from pyfftw.interfaces.cache import set_keepalive_time as pyfftw_set_keepalive_time
37
38 # pystp-specific:
39 from stp_core.phaseretrieval.tiehom import tiehom, tiehom_plan
40 from stp_core.phaseretrieval.phrt import phrt, phrt_plan
41
42 from h5py import File as getHDF5
43
44 # pystp-specific:
45 import stp_core.io.tdf as tdf
46
47 def _write_data(lock, im, index, outfile, outshape, outtype, logfile, cputime,
48 ↪itime):
49     lock.acquire()
50     try:
51         t0 = time()
52         f_out = getHDF5( outfile, 'a' )
53         f_out_dset = f_out.require_dataset('exchange/data', outshape, outtype,
54 ↪chunks=tdf.get_dset_chunks(outshape[0]))
55         tdf.write_tomo(f_out_dset, index, im.astype(float32))
56
57         # Set minimum and maximum:
58         if ( amin(im[:]) < float(f_out_dset.attrs['min']) ):
59             f_out_dset.attrs['min'] = str(amin(im[:]))
60         if ( amax(im[:]) > float(f_out_dset.attrs['max']) ):

```

```

60         f_out_dset.attrs['max'] = str(amax(im[:]))
61         f_out.close()
62         t1 = time()
63
64         # Print out execution time:
65         log = open(logfilename, "a")
66         log.write(linesep + "\ttomo_%s processed (CPU: %0.3f sec - I/O: %0.3f sec)."
↪ % (str(index).zfill(4), cputime, t1 - t0 + itime))
67         log.close()
68
69     finally:
70         lock.release()
71
72
73 def _process(lock, int_from, int_to, infile, outfile, outshape, outtype, method, plan,
↪ logfilename):
74
75     # Process the required subset of images:
76     for i in range(int_from, int_to + 1):
77
78         # Read input image:
79         t0 = time()
80         f_in = getHDF5(infile, 'r')
81         if "/tomo" in f_in:
82             dset = f_in['tomo']
83         else:
84             dset = f_in['exchange/data']
85         im = tdf.read_tomo(dset, i).astype(float32)
86         f_in.close()
87         t1 = time()
88
89         # Perform phase retrieval (first time also PyFFTW prepares a plan):
90         if (method == 0):
91             im = tiehom(im, plan).astype(float32)
92         else:
93             im = phrt(im, plan, method).astype(float32)
94         t2 = time()
95
96         # Save processed image to HDF5 file (atomic procedure - lock used):
97         _write_data(lock, im, i, outfile, outshape, outtype, logfilename, t2 - t1, t1_
↪ - t0)
98
99
100 def main(argv):
101     """To do...
102
103     """
104     lock = Lock()
105
106     skip_flat = True
107     first_done = False
108     pyfftw_cache_disable()
109     pyfftw_cache_enable()
110     pyfftw_set_keepalive_time(1800)
111
112     # Get the from and to number of files to process:
113     int_from = int(argv[0])
114     int_to = int(argv[1])

```

```

115
116 # Get full paths of input TDF and output TDF:
117 infile = argv[2]
118 outfile = argv[3]
119
120 # Get the phase retrieval parameters:
121 method = int(argv[4])
122 param1 = double(argv[5]) # e.g. regParam, or beta
123 param2 = double(argv[6]) # e.g. thresh or delta
124 energy = double(argv[7])
125 distance = double(argv[8])
126 pixsize = double(argv[9]) / 1000.0 # pixsize from micron to mm:
127 pad = True if argv[10] == "True" else False
128
129 # Number of threads (actually processes) to use and logfile:
130 nr_threads = int(argv[11])
131 logfilename = argv[12]
132
133 # Log infos:
134 log = open(logfilename, "w")
135 log.write(linesep + "\tInput TDF file: %s" % (infile))
136 log.write(linesep + "\tOutput TDF file: %s" % (outfile))
137 log.write(linesep + "\t-----")
138 if (method == 0):
139     log.write(linesep + "\tMethod: TIE-Hom (Paganin et al., 2002)")
140     log.write(linesep + "\t-----")
141     log.write(linesep + "\tDelta/Beta: %0.1f" % ((param2/param1)) )
142 #else:
143 # log.write(linesep + "\tMethod: Projected CTF (Moosmann et al., 2011)")
144 # log.write(linesep + "\t-----")
145 # log.write(linesep + "\tDelta/Beta: %0.1f" % ((param2/param1)) )
146 log.write(linesep + "\tEnergy: %0.1f keV" % (energy))
147 log.write(linesep + "\tDistance: %0.1f mm" % (distance))
148 log.write(linesep + "\tPixel size: %0.3f micron" % (pixsize*1000))
149 log.write(linesep + "\t-----")
150 log.write(linesep + "\tBrowsing input files...")
151 log.close()
152
153 # Remove a previous copy of output:
154 if exists(outfile):
155     remove(outfile)
156
157 # Open the HDF5 file:
158 f_in = getHDF5(infile, 'r')
159 if "/tomo" in f_in:
160     dset = f_in['tomo']
161 else:
162     dset = f_in['exchange/data']
163 num_proj = tdf.get_nr_projs(dset)
164 num_sinos = tdf.get_nr_sinos(dset)
165
166 if (num_proj == 0):
167     log = open(logfilename, "a")
168     log.write(linesep + "\tNo projections found. Process will end.")
169     log.close()
170     exit()
171
172 log = open(logfilename, "a")

```

```

173     log.write(linesep + "\tInput files browsed correctly.")
174     log.close()
175
176     # Check extrema (int_to == -1 means all files):
177     if ( (int_to >= num_proj) or (int_to == -1) ):
178         int_to = num_proj - 1
179
180     if ( (int_from < 0) ):
181         int_from = 0
182
183     # Prepare the plan:
184     log = open(logfilename, "a")
185     log.write(linesep + "\tPreparing the work plan..")
186     log.close()
187
188     im = tdf.read_tomo(dset, 0).astype(float32)
189
190
191     outshape = tdf.get_dset_shape(im.shape[1], im.shape[0], num_proj)
192     f_out = getHDF5(outfile, 'w')
193     f_out_dset = f_out.create_dataset('exchange/data', outshape, im.dtype)
194     f_out_dset.attrs['min'] = str(amin(im[:]))
195     f_out_dset.attrs['max'] = str(amax(im[:]))
196
197     f_out_dset.attrs['version'] = '1.0'
198     f_out_dset.attrs['axes'] = "y:theta:x"
199
200     f_in.close()
201     f_out.close()
202
203     if (method == 0):
204         # Paganin's:
205         plan = tiehom_plan (im, param1, param2, energy, distance, pixsize, pad)
206     else:
207         plan = phrt_plan (im, energy, distance, pixsize, param2, param1,
→ method, pad)
208
209     # Run several threads for independent computation without waiting for
→ threads completion:
210     for num in range(nr_threads):
211         start = (num_proj / nr_threads)*num
212         if (num == nr_threads - 1):
213             end = num_proj - 1
214         else:
215             end = (num_proj / nr_threads)*(num + 1) - 1
216         Process(target=_process, args=(lock, start, end, infile, outfile,
→ outshape, im.dtype, method, plan, logfilename)).start()
217
218     #start = 0
219     #end = num_proj - 1
220     #_process(lock, start, end, infile, outfile, outshape, im.dtype, method,
→ plan, logfilename)
221
222     #255 256 C:\Temp\BrunGeorgos_corr.tdf C:\Temp\BrunGeorgos_corr_phrt.tdf
→ 0 1.0 2000.0 22.0 300.0 2.2 False 1 C:\Temp\log_00.txt

```

```

223 #255 256 C:\Temp\BrunGeorgos_corr.tdf C:\Temp\BrunGeorgos_corr_phrt.tdf
→4 2.5 1.0 22.0 300.0 2.2 False 1 C:\Temp\log_00.txt
224
225 if __name__ == "__main__":
226     main(argv[1:])

```

## exec\_tdf2tiff

This section contains the `exec_tdf2tiff` script.

Download file: `exec_tdf2tiff.py`

```

1 #####
2 # (C) 2016 Elettra - Sincrotrone Trieste S.C.p.A.. All rights reserved. #
3 # # #
4 # # #
5 # This file is part of STP-Core, the Python core of SYRMEP Tomo Project, #
6 # a software tool for the reconstruction of experimental CT datasets. #
7 # # #
8 # STP-Core is free software: you can redistribute it and/or modify it #
9 # under the terms of the GNU General Public License as published by the #
10 # Free Software Foundation, either version 3 of the License, or (at your #
11 # option) any later version. #
12 # # #
13 # STP-Core is distributed in the hope that it will be useful, but WITHOUT #
14 # ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or #
15 # FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License #
16 # for more details. #
17 # # #
18 # You should have received a copy of the GNU General Public License #
19 # along with STP-Core. If not, see <http://www.gnu.org/licenses/>. #
20 # # #
21 #####
22
23 #
24 # Author: Francesco Brun
25 # Last modified: July, 8th 2016
26 #
27
28 import datetime
29 import os
30 import os.path
31 import time
32
33 from sys import argv, exit
34 from numpy import float32, float64
35
36 from tiff file import imread, imsave
37 from h5py import File as getHDF5
38
39 # pystp-specific:
40 import stp_core.io.tdf as tdf
41
42 from multiprocessing import Process, Lock
43
44 def _write_log(lock, fname, logfilename, iotime):

```

```

45     """To do...
46
47     """
48     lock.acquire()
49     try:
50         # Print out execution time:
51         log = open(logfilename,"a")
52         log.write(os.linesep + "\t%s converted in %0.3f sec." % (os.path.
↳basename(fname), iotime))
53         log.close()
54
55     finally:
56         lock.release()
57
58 def _process(lock, int_from, int_to, infile, dset_str, TIFFFormat, projorder, outpath,
↳outprefix, logfilename):
59     """To do...
60
61     """
62     try:
63
64         f = getHDF5( infile, 'r' )
65         dset = f[dset_str]
66
67         # Process the required subset of images:
68         for i in range(int_from, int_to + 1):
69
70             # Read input image:
71             t0 = time.time()
72
73             if projorder:
74                 im = tdf.read_tomo( dset, i )
75             else:
76                 im = tdf.read_sino( dset, i )
77
78             if ( TIFFFormat ):
79                 fname = outpath + outprefix + '_' + str(i).zfill(4) + '.tif'
80             else:
81                 fname = outpath + outprefix + '_' + str(i).zfill(4) + '_' + str(im.
↳shape[1]) + \
82                     'x' + str(im.shape[0]) + '_' + str(im.dtype)      + '.raw'
83
84             # Cast type (if required but it should never occur):
85             if ((im.dtype).type is float64):
86                 im = im.astype(float32, copy=False)
87
88             if ( TIFFFormat ):
89                 imsave(fname, im)
90             else:
91                 im.tofile(fname)
92
93             t1 = time.time()
94
95             # Print out execution time:
96             _write_log(lock, fname, logfilename, t1 - t0)
97
98         f.close()
99

```



```

100     except Exception:
101
102         pass
103
104
105 def main(argv):
106     """
107     Converts a TDF file (HDF5 Tomo Data Format) into a sequence of TIFF_
↳(uncompressed) files.
108
109     Parameters
110     -----
111     from : scalar, integer
112         among all the projections (or sinogram) data, a subset of the volume can
113         be specified, ranging from the parameter "from" to the parameter "to"
114         (see next). In most cases, this parameter is 0.
115
116     to : scalar, integer
117         among all the projections (or sinogram) data, a subset of the volume can
118         be specified, ranging from the parameter "from" (see previous parameter) to
119         the parameter "to". If the value -1 is specified, all the projection (or_
↳sinogram)
120         data will be considered.
121
122     in_file : string
123         path with filename of the TDF to read from (e.g. "Z:\\sample1.tdf").
124
125     out_path : string
126         path that will contain the sequence of TIFF files (e.g. "Z:\\sample1\\tomo\\
↳"). WARNING:
127         the program does NOT automatically create non-existing folders and subfolders_
↳specified
128         in the path. Moreover, if files with the same name already exist they will be_
↳automatically
129         overwritten.
130
131     file_prefix : string
132         string to be assumed as the filename prefix of the TIFF files to create for_
↳the projection (or
133         sinogram) data. E.g. "tomo" will create files having name "tomo_0001.tif",
↳"tomo_0002.tif".
134
135     flat_prefix : string
136         string to be assumed as the filename prefix of the TIFF files to create for_
↳the flat (white field)
137         data. E.g. "flat" will create files having name "flat_1.tif", "flat_2.tif"._
↳If dark or flat data have
138         to be skipped the string "-" can be specified.
139
140     dark_prefix : string
141         string to be assumed as the filename prefix of the TIFF files to create for_
↳the dark (dark field)
142         data. E.g. "dark" will create files having name "dark_1.tif", "dark_2.tif"._
↳If dark or flat data have
143         to be skipped the string "-" can be specified.
144
145     projection_order : boolean string
146         specify the string "True" to create TIFF files for projections (the most_
↳common case), "False"

```

```

147     for sinograms.
148
149     TIFF_format : boolean string
150         specify the string "True" to create TIFF files, "False" for RAW files.
151
152     nr_threads : int
153         number of multiple threads (actually processes) to consider to speed up the_
↪whole conversion process.
154
155     log_file : string
156         path with filename of a log file (e.g. "R:\\log.txt") where info about the_
↪conversion is reported.
157
158     Returns
159     -----
160     no return value
161
162     Example
163     -----
164     Example call to convert all the projections data to a sequence of tomo*.tif files:
165
166     python tdf2tiff.py 0 -1 "C:\Temp\wet12T4part2.tdf" "C:\Temp\tomo" tomo flat_
↪dark True True 3 "C:\Temp\log.txt"
167
168     Requirements
169     -----
170     - Python 2.7 with the latest NumPy, SciPy, H5Py.
171     - TIFFFile from C. Gohlke
172     - tdf.py
173
174     Tests
175     -----
176     Tested with WinPython-64bit-2.7.6.3 (Windows) and Anaconda 2.1.0 (Linux 64-bit).
177
178     """
179
180     lock = Lock()
181
182     # To be used without flat fielding (just conversion):
183     first_done = False
184
185     # Get the from and to number of files to process:
186     int_from = int(argv[0])
187     int_to = int(argv[1]) # -1 means "all files"
188
189     # Get paths:
190     infile = argv[2]
191     outpath = argv[3]
192
193     fileprefix = argv[4]
194     flatprefix = argv[5]
195     darkprefix = argv[6]
196
197     if (flatprefix == "-"):
198         skipflat = True
199     else:
200         skipflat = False
201

```

```

202     if (darkprefix == "-"):
203         skipdark = True
204     else:
205         skipdark = False
206
207     if (fileprefix == "-"):
208         skiptomo = True
209     else:
210         skiptomo = False
211
212     projorder = argv[7]
213     if projorder == "True":
214         projorder = True
215     else:
216         projorder = False
217
218     TIFFFormat = argv[8]
219     if TIFFFormat == "True":
220         TIFFFormat = True
221     else:
222         TIFFFormat = False
223
224     nr_threads = int(argv[9])
225     logfilename = argv[10]
226
227     # Check prefixes and path:
228     if not outpath.endswith(os.path.sep): outpath += os.path.sep
229
230     # Init variables:
231     num_files = 0
232     num_flats = 0
233     num_darks = 0
234
235     # Get the files in infile:
236     log = open(logfilename, "w")
237     log.write(os.linesep + "\tInput TDF: %s" % (infile))
238     if ( TIFFFormat ):
239         log.write(os.linesep + "\tOutput path where TIFF files will be created: %s" %
↳(outpath))
240     else:
241         log.write(os.linesep + "\tOutput path where RAW files will be created: %s" %
↳(outpath))
242     log.write(os.linesep + "\t-----")
243     log.write(os.linesep + "\tFile output prefix: %s" % (fileprefix))
244     log.write(os.linesep + "\tFlat images output prefix: %s" % (flatprefix))
245     log.write(os.linesep + "\tDark images output prefix: %s" % (darkprefix))
246     log.write(os.linesep + "\t-----")
247
248     if (not (skiptomo)):
249         if (int_to != -1):
250             log.write(os.linesep + "\tThe subset [%d,%d] of the data will be
↳considered." % (int_from, int_to))
251
252     if (projorder):
253         log.write(os.linesep + "\tProjection order assumed.")
254     else:
255         log.write(os.linesep + "\tSinogram order assumed.")
256

```

```

257     log.write(os.linesep + "\t-----")
258     log.close()
259
260     if not os.path.exists(infile):
261         log = open(logfilename, "a")
262         log.write(os.linesep + "\tError: input TDF file not found. Process will end.")
263         log.close()
264         exit()
265
266     # Open the HDF5 file:
267     f = getHDF5( infile, 'r' )
268
269     oldTDF = False;
270
271     try:
272         dset = f['tomo']
273         oldTDF = True
274
275     except Exception:
276
277         pass
278
279     if not oldTDF:
280
281         #try:
282             dset = f['exchange/data']
283
284         #except Exception:
285
286             # log = open(logfilename, "a")
287             # log.write(os.linesep + "\tError: invalid TDF format. Process will end.")
288             # log.close()
289             # exit()
290
291     if projorder:
292         num_files = tdf.get_nr_projs(dset)
293     else:
294         num_files = tdf.get_nr_sinos(dset)
295     f.close()
296
297
298
299     # Get attributes:
300     try:
301         f = getHDF5( infile, 'r' )
302         if ('version' in f.attrs) and (f.attrs['version'] == 'TDF 1.0'):
303             log = open(logfilename, "a")
304             log.write(os.linesep + "\tTDF version 1.0 found.")
305             log.write(os.linesep + "\t-----")
306             log.close()
307         f.close()
308
309     except:
310         log = open(logfilename, "a")
311         log.write(os.linesep + "\tWarning: TDF version unknown. Some features will_
↪not be available.")
312         log.write(os.linesep + "\t-----")
313         log.close()

```

```

314 # Check extrema (int_to == -1 means all files):
315 if ( (int_to >= num_files) or (int_to == -1) ):
316     int_to = num_files - 1
317
318
319
320
321 # Spawn the process for the conversion of flat images:
322 if not skipflat:
323
324     f = getHDF5( infile, 'r' )
325     if oldTDF:
326         dset_str = 'flat'
327     else:
328         dset_str = 'exchange/data_white'
329     num_flats = tdf.get_nr_projs(f[dset_str])
330     f.close()
331
332     if ( num_flats > 0):
333         Process(target=_process, args=(lock, 0, num_flats - 1, infile, dset_str,
↪TIFFFormat,
334                                     True, outpath, flatprefix, logfilename)).
↪start()
335         #_process(lock, 0, num_flats - 1, infile, dset_str, TIFFFormat, projorder,
↪ outpath, flatprefix, logfilename)
336
337 # Spawn the process for the conversion of dark images:
338 if not skipdark:
339
340     f = getHDF5( infile, 'r' )
341     if oldTDF:
342         dset_str = 'dark'
343     else:
344         dset_str = 'exchange/data_dark'
345     num_darks = tdf.get_nr_projs(f[dset_str])
346     f.close()
347
348     if ( num_darks > 0):
349         Process(target=_process, args=(lock, 0, num_darks - 1, infile, dset_str,
↪TIFFFormat,
350                                     True, outpath, darkprefix, logfilename)).
↪start()
351         #_process(lock, 0, num_darks - 1, infile, dset_str, TIFFFormat, projorder,
↪ outpath, darkprefix, logfilename)
352
353 # Spawn the processes for the conversion of projection or sinogram images:
354 if not skiptomo:
355
356     if oldTDF:
357         dset_str = 'tomo'
358     else:
359         dset_str = 'exchange/data'
360
361     # Start the process for the conversion of the projections (or sinograms) in a
↪multi-threaded way:
362     for num in range(nr_threads):
363         start = ( (int_to - int_from + 1) / nr_threads)*num + int_from
364         if (num == nr_threads - 1):

```

```

365         end = int_to
366     else:
367         end = ( (int_to - int_from + 1) / nr_threads)*(num + 1) + int_from - 1
368
369         Process(target=_process, args=(lock, start, end, infile, dset_str,
↪TIFFFormat,
370                                     projorder, outpath, fileprefix,
↪
↪logfilefilename)).start()
371
372         #_process(lock, start, end, infile, dset_str, TIFFFormat, projorder,
↪
↪outpath, fileprefix, logfilefilename)
373
374
375 if __name__ == "__main__":
376     main(argv[1:])

```

## exec\_tiff2tdf

This section contains the `exec_tiff2tdf` script.

Download file: `exec_tiff2tdf.py`

```

1  #####
2  # (C) 2016 Elettra - Sincrotrone Trieste S.C.p.A.. All rights reserved. #
3  # # #
4  # # #
5  # This file is part of STP-Core, the Python core of SYRMEP Tomo Project, #
6  # a software tool for the reconstruction of experimental CT datasets. #
7  # # #
8  # STP-Core is free software: you can redistribute it and/or modify it #
9  # under the terms of the GNU General Public License as published by the #
10 # Free Software Foundation, either version 3 of the License, or (at your #
11 # option) any later version. #
12 # # #
13 # STP-Core is distributed in the hope that it will be useful, but WITHOUT #
14 # ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or #
15 # FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License #
16 # for more details. #
17 # # #
18 # You should have received a copy of the GNU General Public License #
19 # along with STP-Core. If not, see <http://www.gnu.org/licenses/>. #
20 # # #
21 #####
22
23 #
24 # Author: Francesco Brun
25 # Last modified: July, 8th 2016
26 #
27
28 import datetime
29 import os
30 import os.path
31 import numpy
32 import time
33
34 from time import strftime
35 from sys import argv, exit

```

```

36 from glob import glob
37
38 from tifffile import imread, imsave
39 from h5py import File as getHDF5
40
41 # pystp-specific:
42 import stp_core.io.tdf as tdf
43 from multiprocessing import Process, Lock
44
45
46 def _write_data(lock, im, index, offset, abs_offset, imfilename, timestamp, projorder,
47 ↪ tot_files,
48 ↪ provenance_dt, outfile, dsetname, outshape, outtype, logfilename, ↪
49 ↪ itime):
50     """To do...
51
52     """
53     lock.acquire()
54     try:
55         # Open the HDF5 file to be populated with projections (or sinograms):
56         t0 = time.time()
57         f_out = getHDF5( outfile, 'a' )
58         f_out_dset = f_out.require_dataset(dsetname, outshape, outtype, chunks=tdf.
59 ↪ get_dset_chunks(outshape[0]))
60
61         # Write the projection file or sinogram file:
62         if projorder:
63             tdf.write_tomo(f_out_dset, index - abs_offset, im)
64         else:
65             tdf.write_sino(f_out_dset, index - abs_offset, im)
66
67         # Set minimum and maximum:
68         if ( numpy.amin(im[:]) < float(f_out_dset.attrs['min']) ):
69             f_out_dset.attrs['min'] = str(numpy.amin(im[:]))
70         if ( numpy.amax(im[:]) > float(f_out_dset.attrs['max']) ):
71             f_out_dset.attrs['max'] = str(numpy.amax(im[:]))
72
73         # Save provenance metadata:
74         provenance_dset = f_out.require_dataset('provenance/detector_output', (tot_
75 ↪ files,), dtype=provenance_dt)
76         provenance_dset["filename", offset - abs_offset + index] = numpy.string_(os.
77 ↪ path.basename(imfilename))
78         provenance_dset["timestamp", offset - abs_offset + index] = numpy.string_(
79 ↪ datetime.datetime.fromtimestamp(timestamp).strftime('%Y-%m-%d %H:%M:%S.%f
80 ↪')[:-3])
81
82         # Close the HDF5:
83         f_out.close()
84         t1 = time.time()
85
86         # Print out execution time:
87         log = open(logfilename, "a")
88         log.write(os.linesep + "\t%s processed (I: %0.3f sec - O: %0.3f sec)." % (os.
89 ↪ path.basename(imfilename), itime, t1 - t0))
90         log.close()
91
92     finally:

```

```

87     lock.release()
88
89 def _process(lock, int_from, int_to, offset, abs_offset, files, projorder, outfile,
90 ↪dsetname, outshape, outtype,
91     crop_top, crop_bottom, crop_left, crop_right, tot_files, provenance_dt,
92 ↪logfile):
93     """To do...
94
95     """
96     # Process the required subset of images:
97     for i in range(int_from, int_to + 1):
98
99         # Read input image:
100         t0 = time.time()
101         im = imread(files[i])
102
103         # Crop:
104         im = im[crop_top:im.shape[0]-crop_bottom,crop_left:im.shape[1]-crop_right]
105
106         # Get the timestamp:
107         t = os.path.getmtime(files[i])
108         t1 = time.time()
109
110         # Save processed image to HDF5 file (atomic procedure - lock used):
111         _write_data(lock, im, i, offset, abs_offset, files[i], t, projorder, tot_
112 ↪files, provenance_dt,
113         outfile, dsetname, outshape, outtype, logfile, t1 - t0)
114
115 def main(argv):
116     """
117     Converts a sequence of TIFF files into a TDF file (HDF5 Tomo Data Format).
118
119     Parameters
120     -----
121     from : scalar, integer
122         among all the projections (or sinogram) files, a subset of files can be
123 ↪specified,
124         ranging from the parameter "from" to the parameter "to" (see next). In most
125 cases, this parameter is 0.
126
127     to : scalar, integer
128         among all the projections (or sinogram) files, a subset of files can be
129 ↪specified,
130         ranging from the parameter "from" (see previous parameter) to the parameter
131 "to". If the value -1 is specified, all the projection files will be
132 ↪considered.
133
134     in_path : string
135         path containing the sequence of TIFF files to consider (e.g.
136 ↪"Z:\\sample1\\tomo\\").
137
138     out_file : string
139         path with filename of the TDF to create (e.g. "Z:\\sample1.tdf"). WARNING:
140 ↪the program
141         does NOT automatically create non-existing folders and subfolders specified
142 ↪in the path.
143         Moreover, if a file with the same name already exists it will be
144 ↪automatically deleted and

```



```

136     overwritten.
137
138     crop_top : scalar, integer
139     during the conversion, images can be cropped if required. This parameter
140     ↪ specifies the number
141     of pixels to crop from the top of the image. Leave 0 for no cropping.
142
143     crop_bottom : scalar, integer
144     during the conversion, images can be cropped if required. This parameter
145     ↪ specifies the number
146     of pixels to crop from the bottom of the image. Leave 0 for no cropping.
147
148     crop_left : scalar, integer
149     during the conversion, images can be cropped if required. This parameter
150     ↪ specifies the number
151     of pixels to crop from the left of the image. Leave 0 for no cropping.
152
153     crop_right : scalar, integer
154     during the conversion, images can be cropped if required. This parameter
155     ↪ specifies the number
156     of pixels to crop from the right of the image. Leave 0 for no cropping.
157
158     file_prefix : string
159     string to be assumed as the filename prefix of the TIFF files to consider for
160     ↪ the projection (or
161     sinogram) files. E.g. "tomo" will consider files having name "tomo_0001.tif
162     ↪", "tomo_0002.tif".
163
164     flat_prefix : string
165     string to be assumed as the filename prefix of the TIFF files to consider for
166     ↪ the flat (white field)
167     files. E.g. "flat" will consider files having name "flat_1.tif", "flat_2.tif".
168     ↪ If dark or flat files have
169     to be skipped the string "-" can be specified.
170
171     dark_prefix : string
172     string to be assumed as the filename prefix of the TIFF files to consider for
173     ↪ the dark (dark field)
174     files. E.g. "dark" will consider files having name "dark_1.tif", "dark_2.tif".
175     ↪ If dark or flat files have
176     to be skipped the string "-" can be specified.
177
178     projection_order : boolean string
179     specify the string "True" if the TIFF files represent projections (the most
180     ↪ common case), "False"
181     for sinograms.
182
183     privilege_sino : boolean string
184     specify the string "True" if the TDF will privilege a fast read/write of
185     ↪ sinograms (the most common
186     case), "False" for fast read/write of projections.
187
188     compression : scalar, integer
189     an integer value in the range of [1,9] to be used as GZIP compression factor
190     ↪ in the HDF5 file, where
191     1 is the minimum compression (and maximum speed) and 9 is the maximum (and
192     ↪ slow) compression.
193     The value 0 can be specified with the meaning of no compression.

```

```

180
181     nr_threads : int
182         number of multiple threads (actually processes) to consider to speed up the_
↳whole conversion process.
183
184     log_file : string
185         path with filename of a log file (e.g. "R:\\log.txt") where info about the_
↳conversion is reported.
186
187     Returns
188     -----
189     no return value
190
191     Example
192     -----
193     Example call to convert all the tomo*.tif* projections to a TDF with no cropping_
↳and minimum compression:
194
195     python tiff2tdf.py 0 -1 "Z:\\rawdata\\c_1\\tomo\\" "Z:\\work\\c1_compr9.tdf"
↳0 0 0 0 tomo flat
196     dark True True 1 "S:\\conversion.txt"
197
198     Requirements
199     -----
200     - Python 2.7 with the latest NumPy, SciPy, H5Py.
201     - TIFFFile from C. Gohlke's website http://www.lfd.uci.edu/~gohlke/
202       (consider also to install TIFFFile.c for performances).
203     - tdf.py
204
205     Tests
206     -----
207     Tested with WinPython-64bit-2.7.6.3 (Windows) and Anaconda 2.1.0 (Linux 64-bit).
↳
208     """
209
210     lock = Lock()
211
212     # Get the from and to number of files to process:
213     int_from = int(argv[0])
214     int_to   = int(argv[1]) # -1 means "all files"
215
216     # Get paths:
217     inpath  = argv[2]
218     outfile = argv[3]
219
220     crop_top    = int(argv[4]) # 0 for all means "no cropping"
221     crop_bottom = int(argv[5])
222     crop_left   = int(argv[6])
223     crop_right  = int(argv[7])
224
225     tomoprefix = argv[8]
226     flatprefix = argv[9] # - means "do not consider flat or darks"
227     darkprefix = argv[10] # - means "do not consider flat or darks"
228
229     if (flatprefix == "-") or (darkprefix == "-"):
230         skipflat = True
231     else:
232         skipflat = False

```

```

233 projorder = True if argv[11] == "True" else False
234 privilege_sino = True if argv[12] == "True" else False
235
236
237 # Get compression factor:
238 compr_opts = int(argv[13])
239 compressionFlag = True;
240 if (compr_opts <= 0):
241     compressionFlag = False;
242 elif (compr_opts > 9):
243     compr_opts = 9
244
245 nr_threads = int(argv[14])
246 logfile_name = argv[15]
247
248 # Check prefixes and path:
249 if not inpath.endswith(os.path.sep): inpath += os.path.sep
250
251 # Get the files in inpath:
252 log = open(logfilename, "w")
253 log.write(os.linesep + "\tInput path: %s" % (inpath))
254 log.write(os.linesep + "\tOutput TDF file: %s" % (outfile))
255 log.write(os.linesep + "\t-----")
256 log.write(os.linesep + "\tProjection file prefix: %s" % (tomoprefix))
257 log.write(os.linesep + "\tDark file prefix: %s" % (darkprefix))
258 log.write(os.linesep + "\tFlat file prefix: %s" % (flatprefix))
259 log.write(os.linesep + "\t-----")
260 log.write(os.linesep + "\tCropping:")
261 log.write(os.linesep + "\t\tTop: %d pixels" % (crop_top))
262 log.write(os.linesep + "\t\tBottom: %d pixels" % (crop_bottom))
263 log.write(os.linesep + "\t\tLeft: %d pixels" % (crop_left))
264 log.write(os.linesep + "\t\tRight: %d pixels" % (crop_right))
265 if (int_to != -1):
266     log.write(os.linesep + "\tThe subset [%d,%d] of the input files will be_
↳considered." % (int_from, int_to))
267
268 if (projorder):
269     log.write(os.linesep + "\tProjection order assumed.")
270 else:
271     log.write(os.linesep + "\tSinogram order assumed.")
272
273 if (privilege_sino):
274     log.write(os.linesep + "\tFast I/O for sinograms privileged.")
275 else:
276     log.write(os.linesep + "\tFast I/O for projections privileged.")
277
278 if (compressionFlag):
279     log.write(os.linesep + "\tTDF compression factor: %d" % (compr_opts))
280 else:
281     log.write(os.linesep + "\tTDF compression: none.")
282
283 if (skipflat):
284     log.write(os.linesep + "\tWarning: flat/dark images (if any) will not be_
↳considered.")
285     log.write(os.linesep + "\t-----")
286     log.close()
287
288 # Remove a previous copy of output:

```

```

289     if os.path.exists(outfile):
290         log = open(logfilename, "a")
291         log.write(os.linesep + "\tWarning: an output file with the same name was_
↳ overwritten.")
292         os.remove(outfile)
293         log.close()
294
295     log = open(logfilename, "a")
296     log.write(os.linesep + "\tBrowsing input files...")
297     log.close()
298
299     # Pythonic way to get file list:
300     if os.path.exists(inpath):
301         tomo_files = sorted(glob(inpath + tomaprefix + '*.tif*'))
302         num_files = len(tomo_files)
303     else:
304         log = open(logfilename, "a")
305         log.write(os.linesep + "\tError: input path does not exist. Process will end.
↳ ")
306         log.close()
307         exit()
308
309     if (num_files == 0):
310         log = open(logfilename, "a")
311         log.write(os.linesep + "\tError: no projection files found. Check input path_
↳ and file prefixes.")
312         log.close()
313         exit()
314
315     log = open(logfilename, "a")
316     log.write(os.linesep + "\tInput files browsed correctly.")
317     log.close()
318
319     # Check extrema (int_to == -1 means all files):
320     if ( (int_to >= num_files) or (int_to == -1) ):
321         int_from = 0
322         int_to = num_files - 1
323
324     # In case of subset specified:
325     num_files = int_to - int_from + 1
326
327     # Prepare output HDF5 output (should this be atomic?):
328     im = imread(tomo_files[0])
329
330     # Crop:
331     im = im[crop_top:im.shape[0]-crop_bottom,crop_left:im.shape[1]-crop_right]
332
333     log = open(logfilename, "a")
334     log.write(os.linesep + "\tPreparing the work plan...")
335     log.close()
336
337     #dsetshape = (num_files,) + im.shape
338     if projorder:
339         #dsetshape = tdf.get_dset_shape(privilege_sino, im.shape[1], im.shape[0], num_
↳ files)
340         datashape = tdf.get_dset_shape(im.shape[1], im.shape[0], num_files)
341     else:
342         #dsetshape = tdf.get_dset_shape(privilege_sino, im.shape[1], num_files, im.
↳ shape[0])

```

```

343     datashape = tdf.get_dset_shape(im.shape[1], num_files, im.shape[0])
344
345     if not os.path.isfile(outfile):
346         f = getHDF5( outfile, 'w' )
347
348         f.attrs['version'] = '1.0'
349         f.attrs['implements'] = "exchange:provenance"
350         exchange_group = f.create_group( 'exchange' )
351
352         if (compressionFlag):
353             dset = f.create_dataset('exchange/data', datashape, im.dtype, chunks=tdf.
↪get_dset_chunks(im.shape[1]),
354             compression="gzip", compression_opts=compr_opts, shuffle=True,
↪fletcher32=True)
355         else:
356             dset = f.create_dataset('exchange/data', datashape, im.dtype)
357
358         if privilege_sino:
359             dset.attrs['axes'] = "y:theta:x"
360         else:
361             dset.attrs['axes'] = "theta:y:x"
362
363         dset.attrs['min'] = str(numpy.amin(im[:]))
364         dset.attrs['max'] = str(numpy.amax(im[:]))
365
366         # Get the total number of files to consider:
367         tot_files = num_files
368         if not skipflat:
369             num_flats = len(sorted(glob(inpath + flatprefix + '*.tif*')))
370             num_darks = len(sorted(glob(inpath + darkprefix + '*.tif*')))
371             tot_files = tot_files + num_flats + num_darks
372
373         # Create provenance dataset:
374         provenance_dt = numpy.dtype([("filename", numpy.dtype("S255")), ("timestamp
↪", numpy.dtype("S255"))])
375         metadata_group = f.create_group( 'provenance' )
376         provenance_dset = metadata_group.create_dataset('detector_output', (tot_files,
↪), dtype=provenance_dt)
377
378         provenance_dset.attrs['tomo_prefix'] = tomoprefix;
379         provenance_dset.attrs['dark_prefix'] = darkprefix;
380         provenance_dset.attrs['flat_prefix'] = flatprefix;
381         provenance_dset.attrs['first_index'] = int(tomo_files[0][-8:-4]);
382
383         # Handle the metadata:
384         if (os.path.isfile(inpath + 'logfile.xml')):
385             with open (inpath + 'logfile.xml', "r") as file:
386                 xml_command = file.read()
387                 tdf.parse_metadata(f, xml_command)
388
389         f.close()
390
391         # Print out about plan preparation:
392         log = open(logfilename, "a")
393         log.write(os.linesep + "\tWork plan prepared succesfully.")
394         log.close()
395
396         # Get the files in inpath:

```

```

397     if not skipflat:
398
399         #
400         # Flat part
401         #
402         flat_files = sorted(glob(inpath + flatprefix + '*.tif*'))
403         num_flats = len(flat_files)
404
405         if ( num_flats > 0):
406
407             # Create acquisition group:
408             im = imread(flat_files[0])
409             im = im[crop_top:im.shape[0]-crop_bottom,crop_left:im.shape[1]-crop_right]
410
411             #flatshape = tdf.get_dset_shape(privilege_sino, im.shape[1], im.shape[0],
↳num_flats)
412             flatshape = tdf.get_dset_shape(im.shape[1], im.shape[0], num_flats)
413             f = getHDF5( outfile, 'a' )
414             if (compressionFlag):
415                 dset = f.create_dataset('exchange/data_white', flatshape, im.dtype,
↳chunks=tdf.get_dset_chunks(im.shape[1]),
416                 compression="gzip", compression_opts=compr_opts, shuffle=True,
↳fletcher32=True)
417             else:
418                 dset = f.create_dataset('exchange/data_white', flatshape, im.dtype)
419
420             dset.attrs['min'] = str(numpy.amin(im[:]))
421             dset.attrs['max'] = str(numpy.amax(im[:]))
422
423             if privilege_sino:
424                 dset.attrs['axes'] = "y:theta:x"
425             else:
426                 dset.attrs['axes'] = "theta:y:x"
427             f.close()
428
429             #process(lock, 0, num_flats - 1, 0, flat_files, True, outfile, 'exchange/
↳data_white', dsetshape, im.dtype,
430             # crop_top, crop_bottom, crop_left, crop_right, tot_files, provenance_
↳dt, logfilename )
431
432             else:
433                 log = open(logfilename,"a")
434                 log.write(os.linesep + "\tWarning: flat files not found.")
435                 log.close()
436
437             #
438             # Dark part
439             #
440             dark_files = sorted(glob(inpath + darkprefix + '*.tif*'))
441             num_darks = len(dark_files)
442
443             if ( num_darks > 0):
444                 im = imread(dark_files[0])
445                 im = im[crop_top:im.shape[0]-crop_bottom,crop_left:im.shape[1]-crop_right]
446
447                 #darkshape = tdf.get_dset_shape(privilege_sino, im.shape[1], im.shape[0],
↳num_flats)
448                 darkshape = tdf.get_dset_shape(im.shape[1], im.shape[0], num_darks)

```

```

449     f = getHDF5( outfile, 'a' )
450     if (compressionFlag):
451         dset = f.create_dataset('exchange/data_dark', darkshape, im.dtype,
↳chunks=tdf.get_dset_chunks(im.shape[1]),
452             compression="gzip", compression_opts=compr_opts, shuffle=True,
↳fletcher32=True)
453     else:
454         dset = f.create_dataset('exchange/data_dark', darkshape, im.dtype)
455
456     dset.attrs['min'] = str(numpy.amin(im))
457     dset.attrs['max'] = str(numpy.amax(im))
458
459     if privilege_sino:
460         dset.attrs['axes'] = "y:theta:x"
461     else:
462         dset.attrs['axes'] = "theta:y:x"
463     f.close()
464
465     #process(lock, 0, num_darks - 1, num_flats, dark_files, True, outfile,
↳'exchange/data_dark', dsetshape, im.dtype,
466         #   crop_top, crop_bottom, crop_left, crop_right, tot_files, provenance_
↳dt, logfilename )
467
468     else:
469
470         log = open(logfilename,"a")
471         log.write(os.linesep + "\tWarning: dark files not found.")
472         log.close()
473
474     # Process the required subset of images:
475     if not skipflat:
476         flatdark_offset = num_flats + num_darks
477     else:
478         flatdark_offset = 0
479
480     # Spawn the process for the conversion of flat images:
481     if ( num_flats > 0):
482         Process(target=_process, args=(lock, 0, num_flats - 1, 0, 0, flat_files, True,
↳ outfile, 'exchange/data_white',
483             flatshape, im.dtype, crop_top, crop_bottom, crop_left, crop_right, tot_
↳files, provenance_dt, logfilename )).start()
484
485     # Spawn the process for the conversion of dark images:
486     if ( num_darks > 0):
487         Process(target=_process, args=(lock, 0, num_darks - 1, num_flats, 0, dark_
↳files, True, outfile, 'exchange/data_dark',
488             darkshape, im.dtype, crop_top, crop_bottom, crop_left, crop_right, tot_
↳files, provenance_dt, logfilename )).start()
489
490     # Start the process for the conversion of the projections (or sinograms) in a
↳multi-threaded way:
491     for num in range(nr_threads):
492         start = ( (int_to - int_from + 1) / nr_threads)*num + int_from
493         if (num == nr_threads - 1):
494             end = int_to
495         else:
496             end = ( (int_to - int_from + 1) / nr_threads)*(num + 1) + int_from - 1
497

```

```

498     Process(target=_process, args=(lock, start, end, flatdark_offset, int_from,
↳tomo_files, projorder, outfile, 'exchange/data',
499         datashape, im.dtype, crop_top, crop_bottom, crop_left, crop_right,
↳tot_files, provenance_dt, logfilename)).start()
500
501     #process(lock, start, end, offset, tomo_files, projorder, outfile, 'exchange/
↳data',
502         #         datashape, im.dtype, crop_top, crop_bottom, crop_left, crop_right,
↳tot_files, provenance_dt, logfilename )
503
504 if __name__ == "__main__":
505     main(argv[1:])

```

## tools\_autolimit

This section contains the tools\_autolimit script.

Download file: tools\_autolimit.py

```

1  #####
2  # (C) 2016 Elettra - Sincrotrone Trieste S.C.p.A.. All rights reserved. #
3  # # #
4  # # #
5  # This file is part of STP-Core, the Python core of SYRMEP Tomo Project, #
6  # a software tool for the reconstruction of experimental CT datasets. #
7  # # #
8  # STP-Core is free software: you can redistribute it and/or modify it #
9  # under the terms of the GNU General Public License as published by the #
10 # Free Software Foundation, either version 3 of the License, or (at your #
11 # option) any later version. #
12 # # #
13 # STP-Core is distributed in the hope that it will be useful, but WITHOUT #
14 # ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or #
15 # FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License #
16 # for more details. #
17 # # #
18 # You should have received a copy of the GNU General Public License #
19 # along with STP-Core. If not, see <http://www.gnu.org/licenses/>. #
20 # # #
21 #####
22
23 #
24 # Author: Francesco Brun
25 # Last modified: July, 8th 2016
26 #
27
28 import os
29 import os.path
30 import numpy
31 import time
32
33 from glob import glob
34 from sys import argv, exit
35 from h5py import File as getHDF5
36 from numpy import float32
37 from tiffiffile import imread, imsave
38

```



```

39 # pystp-specific:
40 import stp_core.io.tdf as tdf
41
42 def main(argv):
43     """Computes min/max limits to be used in image degradation to 8-bit or 16-bit.
44
45     Parameters
46     -----
47     argv[0] : string
48         The absolute path of the input folder containing reconstructed TIFF files.
49
50     argv[1] : string
51         The absolute path of the output txt file with the proposed limits as string
52     ↪ "min:max".
53
54     Example
55     -----
56     tools_autolimit "S:\\SampleA\\slices" "R:\\Temp\\autolimit.txt"
57
58     """
59     try:
60         # Get input and output paths:
61         inpath = argv[0]
62         outfile = argv[1] # The txt file with the proposed center
63
64         if not inpath.endswith(os.path.sep): inpath += os.path.sep
65
66         # Get the number of files in folder:
67         files = sorted(glob(inpath + '*.tif*'))
68         num_files = len(files)
69
70         # Read the median slice from disk:
71         im = imread(files[num_files/2])
72
73         # Flat the image and sort it:
74         im_flat = im.flatten()
75         im_flat = numpy.sort(im_flat)
76
77         # Return as minimum the value the skip 0.30% of "black" tail and 0.005% of
78     ↪ "white" tail:
79         low_idx = int(im_flat.shape[0] * 0.0030)
80         high_idx = int(im_flat.shape[0] * 0.9995)
81
82         min = im_flat[low_idx]
83         max = im_flat[high_idx]
84
85         # Print center to output file:
86         text_file = open(outfile, "w")
87         text_file.write( str(min) + ":" + str(max) )
88         text_file.close()
89
90     except:
91
92         exit()
93
94 if __name__ == "__main__":
95     main(argv[1:])

```

## tools\_multiangle

This section contains the tools\_multiangle script.

Download file: tools\_multiangle.py

```

1 #####
2 # (C) 2016 Elettra - Sincrotrone Trieste S.C.p.A.. All rights reserved. #
3 # #
4 # #
5 # This file is part of STP-Core, the Python core of SYRMEP Tomo Project, #
6 # a software tool for the reconstruction of experimental CT datasets. #
7 # #
8 # STP-Core is free software: you can redistribute it and/or modify it #
9 # under the terms of the GNU General Public License as published by the #
10 # Free Software Foundation, either version 3 of the License, or (at your #
11 # option) any later version. #
12 # #
13 # STP-Core is distributed in the hope that it will be useful, but WITHOUT #
14 # ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or #
15 # FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License #
16 # for more details. #
17 # #
18 # You should have received a copy of the GNU General Public License #
19 # along with STP-Core. If not, see <http://www.gnu.org/licenses/>. #
20 # #
21 #####
22
23 #
24 # Author: Francesco Brun
25 # Last modified: Sept, 28th 2016
26 #
27
28 # python:
29 from sys import argv, exit
30 from os import remove, sep, linesep, listdir, makedirs
31 from os.path import exists, dirname, basename, splitext
32 from numpy import array, finfo, copy, float32, double, amin, amax, tile, concatenate,
↳asarray
33 from numpy import empty, reshape, log as nplog, arange, squeeze, fromfile, ndarray,
↳where, meshgrid
34 from time import time
35 from multiprocessing import Process, Lock
36
37 # pystp-specific:
38 from stp_core.preprocess.extfov_correction import extfov_correction
39 from stp_core.preprocess.flat_fielding import flat_fielding
40 from stp_core.preprocess.ring_correction import ring_correction
41 from stp_core.preprocess.extract_flatdark import extract_flatdark
42
43 from stp_core.phaseretrieval.tiehom import tiehom, tiehom_plan
44 from stp_core.phaseretrieval.phrt import phrt, phrt_plan
45
46 from stp_core.reconstruct.rec_astra import recon_astra_fbp, recon_astra_iterative
47 from stp_core.reconstruct.rec_fista_tv import recon_fista_tv
48 from stp_core.reconstruct.rec_mr_fbp import recon_mr_fbp
49 from stp_core.reconstruct.rec_gridrec import recon_gridrec
50
51 from stp_core.postprocess.postprocess import postprocess

```

```

52
53 from stp_core.utils.padding import upperPowerOfTwo, padImage, padSmoothWidth
54 from stp_core.utils.caching import cache2plan, plan2cache
55
56 from tiffiffle import imread, imsave
57 from h5py import File as getHDF5
58 import stp_core.io.tdf as tdf
59
60
61 def write_log(lock, fname, logfilename):
62     """To do...
63
64     """
65     lock.acquire()
66     try:
67         # Print out execution time:
68         log = open(logfilename, "a")
69         log.write(linesep + "\t%s reconstructed." % basename(fname))
70         log.close()
71
72     finally:
73         lock.release()
74
75 def reconstruct(im, angles, offset, logtransform, param1, circle, scale, pad, method,
76               zerone_mode, dset_min, dset_max, corr_offset, postprocess_required,
77               ↪convert_opt,
78               crop_opt, start, end, outpath, sino_idx, downsc_factor, decim_factor,
79               ↪logfilename, lock, slice_prefix):
80     """Reconstruct a sinogram with FBP algorithm (from ASTRA toolbox).
81
82     Parameters
83     -----
84     im1 : array_like
85         Sinogram image data as numpy array.
86     center : float
87         Offset of the center of rotation to use for the tomographic
88         reconstruction with respect to the half of sinogram width
89         (default=0, i.e. half width).
90     logtransform : boolean
91         Apply logarithmic transformation before reconstruction (default=True).
92     filter : string
93         Filter to apply before the application of the reconstruction algorithm.
94     ↪Filter
95         types are: ram-lak, shepp-logan, cosine, hamming, hann, tukey, lanczos,
96     ↪triangular,
97         gaussian, barlett-hann, blackman, nuttall, blackman-harris, blackman-nuttall,
98         flat-top, kaiser, parzen.
99     circle : boolean
100         Create a circle in the reconstructed image and set to zero pixels outside the
101         circle (default=False).
102
103     """
104     # Copy required due to multithreading:
105     im_f = im
106
107     # Decimate projections if required:
108     #if decim_factor > 1:
109     #    im = im[:, :decim_factor, :]

```

```

106
107     # Upscale projections (if required):
108     if (abs(scale - 1.0) > finfo(float32).eps):
109         siz_orig1 = im_f.shape[1]
110         im_f = imresize(im_f, (im_f.shape[0], int(round(scale * im_f.shape[1]))),
↪interp='bicubic', mode='F')
111         offset = int(offset * scale)
112
113     offset = int(round(offset))
114
115     # Apply transformation for changes in the center of rotation:
116     if (offset != 0):
117         if (offset >= 0):
118             im_f = im_f[:, :-offset]
119
120             tmp = im_f[:,0] # Get first column
121             tmp = tile(tmp, (offset,1)) # Replicate the first column the right number
↪of times
122             im_f = concatenate((tmp.T, im_f), axis=1) # Concatenate tmp before the
↪image
123
124         else:
125             im_f = im_f[:, abs(offset):]
126
127             tmp = im_f[:, im_f.shape[1] - 1] # Get last column
128             tmp = tile(tmp, (abs(offset),1)) # Replicate the last column the right
↪number of times
129             im_f = concatenate((im_f, tmp.T), axis=1) # Concatenate tmp after the image
130
131     # Downscale projections (without pixel averaging):
132     #if downsc_factor > 1:
133     #    im = im[:, ::downsc_factor]
134
135     # Scale image to [0,1] range (if required):
136     if (zerone_mode):
137
138         #print dset_min
139         #print dset_max
140         #print numpy.amin(im_f[:])
141         #print numpy.amax(im_f[:])
142         #im_f = (im_f - dset_min) / (dset_max - dset_min)
143
144         # Cheating the whole process:
145         im_f = (im_f - numpy.amin(im_f[:])) / (numpy.amax(im_f[:]) - numpy.amin(im_
↪f[:]))
146
147     # Apply log transform:
148     if (logtransform == True):
149         im_f[im_f <= finfo(float32).eps] = finfo(float32).eps
150         im_f = -nplog(im_f + corr_offset)
151
152     # Replicate pad image to double the width:
153     if (pad):
154
155         dim_o = im_f.shape[1]
156         n_pad = im_f.shape[1] + im_f.shape[1] / 2
157         marg = (n_pad - dim_o) / 2
158

```

```

159     # Pad image:
160     im_f = padSmoothWidth(im_f, n_pad)
161
162     # Loop for all the required angles:
163     for i in range(int(round(start)), int(round(end)) + 1):
164
165         # Save image for next step:
166         im = im_f
167
168         # Apply projection removal (if required):
169         im_f = im_f[0:int(round(i/decim_factor)), :]
170
171         # Perform the actual reconstruction:
172         if (method.startswith('FBP')):
173             im_f = recon_astra_fbp(im_f, angles, method, param1)
174         elif (method == 'MR-FBP_CUDA'):
175             im_f = recon_mr_fbp(im_f, angles)
176         elif (method == 'FISTA-TV_CUDA'):
177             im_f = recon_fista_tv(im_f, angles, param1, param1)
178         elif (method == 'MLEM'):
179             im_f = recon_tomopy_iterative(im_f, angles, method, param1)
180         elif (method == 'GRIDREC'):
181             [im_f, im_f] = recon_gridrec(im_f, im_f, angles, param1)
182         else:
183             im_f = recon_astra_iterative(im_f, angles, method, param1, zerone_mode)
184
185         # Crop:
186         if (pad):
187             im_f = im_f[marg:dim_o + marg, marg:dim_o + marg]
188
189         # Resize (if necessary):
190         if (abs(scale - 1.0) > finfo(float32).eps):
191             im_f = imresize(im_f, (siz_orig1, siz_orig1), interp='nearest', mode='F')
192
193         # Apply post-processing (if required):
194         if postprocess_required:
195             im_f = postprocess(im_f, convert_opt, crop_opt)
196         else:
197             # Create the circle mask for fancy output:
198             if (circle == True):
199                 siz = im_f.shape[1]
200                 if siz % 2:
201                     rang = arange(-siz / 2 + 1, siz / 2 + 1)
202                 else:
203                     rang = arange(-siz / 2, siz / 2)
204                 x,y = meshgrid(rang,rang)
205                 z = x ** 2 + y ** 2
206                 a = (z < (siz / 2 - int(round(abs(offset)/downsc_factor)) ) ** 2)
207                 im_f = im_f * a
208
209             # Write down reconstructed image (file name modified with metadata):
210             fname = outpath + slice_prefix + '_' + str(sino_idx).zfill(4) + '_off=' + ↵
↵str(offset*downsc_factor).zfill(4) + '_proj=' + str(i).zfill(4) + '.tif'
211             imsave(fname, im_f)
212
213         # Restore original image for next step:
214         im_f = im
215

```

```

216     # Write log (atomic procedure - lock used):
217     write_log(lock, fname, logfilename )
218
219
220 def process(sino_idx, num_sinos, infile, outpath, preprocessing_required, corr_plan,
↳norm_sx, norm_dx, flat_end, half_half,
221     half_half_line, ext_fov, ext_fov_rot_right, ext_fov_overlap, ringrem,
↳phaseretrieval_required, phrtmethod,
222     phrt_param1, phrt_param2,
223     energy, distance, pixsize, phrtpad, approx_win, angles, offset,
↳logtransform, param1, circle, scale, pad, method,
224     zerone_mode, dset_min, dset_max, decim_factor, downsc_factor, corr_offset,
↳ postprocess_required, convert_opt,
225     crop_opt, nr_threads, angles_from, angles_to, logfilename, lock, slice_
↳prefix):
226     """To do...
227
228     """
229     slice_nr = sino_idx
230
231     # Perform reconstruction (on-the-fly preprocessing and phase retrieval, if
↳required):
232     if (phaseretrieval_required):
233
234         # In this case a bunch of sinograms is loaded into memory:
235
236         #
237         # Load the temporary data structure reading the input TDF file.
238         # To know the right dimension the first sinogram is pre-processed.
239         #
240
241         # Open the TDF file and get the dataset:
242         f_in = getHDF5(infile, 'r')
243         if "/tomo" in f_in:
244             dset = f_in['tomo']
245         else:
246             dset = f_in['exchange/data']
247
248         # Downscaling and decimation factors considered when determining the
↳approximation window:
249         xrange = arange(sino_idx - approx_win*downsc_factor/2, sino_idx + approx_
↳win*downsc_factor/2, downsc_factor)
250         xrange = xrange[ (xrange >= 0) ]
251         xrange = xrange[ (xrange < num_sinos) ]
252         approx_win = xrange.shape[0]
253
254         # Approximation window cannot be odd:
255         if (approx_win % 2 == 1):
256             approx_win = approx_win-1
257             xrange      = xrange[0:approx_win]
258
259         # Read one sinogram to get the proper dimensions:
260         test_im = tdf.read_sino(dset, xrange[0]).astype(float32)
261         test_im = test_im[:,::decim_factor, ::downsc_factor]
262
263         # Perform the pre-processing of the first sinogram to get the right dimension:
264         if (preprocessing_required):
265             test_im = flat_fielding (test_im, xrange[0]/downsc_factor, corr_plan,
↳flat_end, half_half,

```

```

266             half_half_line/decim_factor, norm_sx, norm_
↳dx).astype(float32)
267         test_im = extfov_correction (test_im, ext_fov, ext_fov_rot_right, ext_fov_
↳overlap/downsc_factor).astype(float32)
268         test_im = ring_correction (test_im, ringrem, flat_end, corr_plan['skip_
↳flat_after'], half_half,
269             half_half_line/decim_factor, ext_fov).
↳astype(float32)
270
271         # Now we can allocate memory for the bunch of slices:
272         tmp_im = empty((approx_win, test_im.shape[0], test_im.shape[1]),
↳dtype=float32)
273         tmp_im[0,:,:] = test_im
274
275         # Reading all the the sinos from TDF file and close:
276         for ct in range(1, approx_win):
277
278             test_im = tdf.read_sino(dset, zrange[ct]).astype(float32)
279             test_im = test_im[:,::decim_factor, ::downsc_factor]
280
281             # Perform the pre-processing for each sinogram of the bunch:
282             if (preprocessing_required):
283                 test_im = flat_fielding (test_im, zrange[ct]/downsc_factor, corr_plan,
↳ flat_end, half_half,
284             half_half_line/decim_factor, norm_sx,
↳norm_dx).astype(float32)
285                 test_im = extfov_correction (test_im, ext_fov, ext_fov_rot_right, ext_
↳fov_overlap/downsc_factor).astype(float32)
286                 test_im = ring_correction (test_im, ringrem, flat_end, corr_plan[
↳'skip_flat_after'], half_half,
287             half_half_line/decim_factor, ext_fov).
↳astype(float32)
288
289             tmp_im[ct,:,:] = test_im
290
291             f_in.close()
292
293             # Now everything has to refer to a downscaled dataset:
294             sino_idx = ((zrange == sino_idx).nonzero())
295
296             #
297             # Perform phase retrieval:
298             #
299
300             # Prepare the plan:
301             if (phrtmethod == 0):
302                 # Paganin's:
303                 phrtplan = tiehom_plan (tmp_im[:,0,:], phrt_param1, phrt_param2, energy,
↳distance, pixsize*downsc_factor, padding=phrtpad)
304             else:
305                 phrtplan = phrt_plan (tmp_im[:,0,:], energy, distance, pixsize*downsc_
↳factor, phrt_param2, phrt_param1, phrtmethod, padding=phrtpad)
306
307             # Process each projection (whose height depends on the size of the bunch):
308             for ct in range(0, tmp_im.shape[1]):
309                 if (phrtmethod == 0):
310                     tmp_im[:,ct,:] = tiehom(tmp_im[:,ct,:], phrtplan).astype(float32)
↳
↳

```

```

311         else:
312             tmp_im[:,ct,:] = phrt(tmp_im[:,ct:], phrtplan, phrtmethod).
↪astype(float32)
313
314         # Extract the requested sinogram:
315         im = tmp_im[sino_idx[0],:,:].squeeze()
316
317     else:
318
319         # Read only one sinogram:
320         f_in = getHDF5(infile, 'r')
321         if "/tomo" in f_in:
322             dset = f_in['tomo']
323         else:
324             dset = f_in['exchange/data']
325         im = tdf.read_sino(dset,sino_idx).astype(float32)
326         f_in.close()
327
328         # Downscale and decimate the sinogram:
329         im = im[:,::decim_factor,::downsc_factor]
330         sino_idx = sino_idx/downsc_factor
331
332         # Perform the preprocessing of the sinogram (if required):
333         if (preprocessing_required):
334             im = flat_fielding(im, sino_idx, corr_plan, flat_end, half_half, half_
↪half_line/decim_factor,
335
336                                     norm_sx, norm_dx).astype(float32)
337             im = extfov_correction(im, ext_fov, ext_fov_rot_right, ext_fov_overlap)
338             im = ring_correction(im, ringrem, flat_end, corr_plan['skip_flat_after'],
↪ half_half,
339
340                                     half_half_line/decim_factor, ext_fov)
341
342         # Log infos:
343         log = open(logfilename,"a")
344         log.write(linesep + "\tPerforming reconstruction with multiple centers of_
↪rotation...")
345         log.write(linesep + "\t-----")
346         log.close()
347
348         # Split the computation into multiple processes:
349         for num in range(nr_threads):
350             start = ( (angles_to - angles_from + 1) / nr_threads)*num + angles_from
351             if (num == nr_threads - 1):
352                 end = angles_to
353             else:
354                 end = ( (angles_to - angles_from + 1) / nr_threads)*(num + 1) + angles_
↪from - 1
355
356             #Process(target=reconstruct, args=(im, angles, offset/downsc_factor,
↪
357             ↪logtransform, param1, circle, scale, pad, method,
358             ↪
359             ↪zerone_mode, dset_min, dset_max, corr_offset, postprocess_
360             ↪required, convert_opt, crop_opt, start, end,
361             ↪
362             ↪outpath, slice_nr, downsc_factor, decim_factor, logfilename,
↪
363             ↪lock, slice_prefix)).start()
364
365         # Actual reconstruction:
366         reconstruct(im, angles, offset/downsc_factor, logtransform, param1, circle,
↪
367         ↪scale, pad, method,

```



```

361         zerone_mode, dset_min, dset_max, corr_offset, postprocess_
↪required, convert_opt, crop_opt,
362         start, end, outputpath, slice_nr, downsc_factor, decim_factor,
↪logfile, lock, slice_prefix)
363
364
365
366
367 def main(argv):
368     """To do...
369
370
371     """
372     lock = Lock()
373     skip_flat = False
374     skip_flat_after = True
375
376     # Get the from and to number of files to process:
377     sino_idx = int(argv[0])
378
379     # Get paths:
380     infile = argv[1]
381     outputpath = argv[2]
382
383     # Essential reconstruction parameters::
384     angles = float(argv[3])
385     offset = float(argv[4])
386     param1 = argv[5]
387     scale = int(float(argv[6]))
388
389     overpad = True if argv[7] == "True" else False
390     logtrsf = True if argv[8] == "True" else False
391     circle = True if argv[9] == "True" else False
392
393     # Parameters for on-the-fly pre-processing:
394     preprocessing_required = True if argv[10] == "True" else False
395     flat_end = True if argv[11] == "True" else False
396     half_half = True if argv[12] == "True" else False
397
398     half_half_line = int(argv[13])
399
400     ext_fov = True if argv[14] == "True" else False
401
402     norm_sx = int(argv[17])
403     norm_dx = int(argv[18])
404
405     ext_fov_rot_right = argv[15]
406     if ext_fov_rot_right == "True":
407         ext_fov_rot_right = True
408         if (ext_fov):
409             norm_sx = 0
410     else:
411         ext_fov_rot_right = False
412         if (ext_fov):
413             norm_dx = 0
414
415     ext_fov_overlap = int(argv[16])
416

```

```
417 skip_ringrem = True if argv[19] == "True" else False
418 ringrem = argv[20]
419
420 # Extra reconstruction parameters:
421 zerone_mode = True if argv[21] == "True" else False
422 corr_offset = float(argv[22])
423
424 reconmethod = argv[23]
425
426 decim_factor = int(argv[24])
427 downsc_factor = int(argv[25])
428
429 # Parameters for postprocessing:
430 postprocess_required = True if argv[26] == "True" else False
431 convert_opt = argv[27]
432 crop_opt = argv[28]
433
434 # Parameters for on-the-fly phase retrieval:
435 phaseretrieval_required = True if argv[29] == "True" else False
436 phrtmethod = int(argv[30])
437 phrt_param1 = double(argv[31]) # param1( e.g. regParam, or beta)
438 phrt_param2 = double(argv[32]) # param2( e.g. thresh or delta)
439 energy = double(argv[33])
440 distance = double(argv[34])
441 pixsize = double(argv[35]) / 1000.0 # pixsize from micron to mm:
442 phrtpad = True if argv[36] == "True" else False
443 approx_win = int(argv[37])
444
445 preprocessingplan_fromcache = True if argv[38] == "True" else False
446 tmpopath = argv[39]
447 if not tmpopath.endswith(sep): tmpopath += sep
448
449 nr_threads = int(argv[40])
450 angles_from = float(argv[41])
451 angles_to = float(argv[42])
452
453 slice_prefix = argv[43]
454
455 logfilename = argv[44]
456
457 if not exists(outpath):
458     makedirs(outpath)
459
460 if not outpath.endswith(sep): outpath += sep
461
462
463 # Log info:
464 log = open(logfilename, "w")
465 log.write(linesep + "\tInput dataset: %s" % (infile))
466 log.write(linesep + "\tOutput path: %s" % (outpath))
467 log.write(linesep + "\t-----")
468 log.write(linesep + "\tLoading flat and dark images...")
469 log.close()
470
471 # Open the HDF5 file:
472 f_in = getHDF5(infile, 'r')
473 if "/tomo" in f_in:
474     dset = f_in['tomo']
```

```

475     else:
476         dset = f_in['exchange/data']
477         if "/provenance/detector_output" in f_in:
478             prov_dset = f_in['provenance/detector_output']
479
480     dset_min = -1
481     dset_max = -1
482     if (zerone_mode):
483         if ('min' in dset.attrs):
484             dset_min = float(dset.attrs['min'])
485         else:
486             zerone_mode = False
487
488         if ('max' in dset.attrs):
489             dset_max = float(dset.attrs['max'])
490         else:
491             zerone_mode = False
492
493     num_sinos = tdf.get_nr_sinos(dset) # Pay attention to the downscale factor
494
495     if (num_sinos == 0):
496         exit()
497
498     # Check extrema:
499     if (sino_idx >= num_sinos):
500         sino_idx = num_sinos - 1
501
502     # Get correction plan and phase retrieval plan (if required):
503     corrplan = 0
504     if (preprocessing_required):
505         # Load flat fielding plan either from cache (if required) or from TDF file_
506         ↪and cache it for faster re-use:
507         if (preprocessingplan_fromcache):
508             try:
509                 corrplan = cache2plan(infile, tmppath)
510             except Exception as e:
511                 #print "Error(s) when reading from cache"
512                 corrplan = extract_flatdark(f_in, flat_end, logfilename)
513                 plan2cache(corrplan, infile, tmppath)
514         else:
515             corrplan = extract_flatdark(f_in, flat_end, logfilename)
516             plan2cache(corrplan, infile, tmppath)
517
518         # Downscale flat and dark images if necessary:
519         if isinstance(corrplan['im_flat'], ndarray):
520             ↪factor]
521             corrplan['im_flat'] = corrplan['im_flat'][:, :downsc_factor, :downsc_
522             ↪factor]
523         if isinstance(corrplan['im_dark'], ndarray):
524             ↪factor]
525             corrplan['im_dark'] = corrplan['im_dark'][:, :downsc_factor, :downsc_
526             ↪factor]
527         if isinstance(corrplan['im_flat_after'], ndarray):
528             ↪factor]
529             corrplan['im_flat_after'] = corrplan['im_flat_after'][:, :downsc_factor,
530             ↪factor]
531         if isinstance(corrplan['im_dark_after'], ndarray):
532             ↪factor]
533             corrplan['im_dark_after'] = corrplan['im_dark_after'][:, :downsc_factor,
534             ↪factor]
535
536     f_in.close()
537

```

```

528
529     # Log infos:
530     log = open(logfilename,"a")
531     log.write(linesep + "\tPerforming preprocessing...")
532     log.close()
533
534     # Run computation:
535     process( sino_idx, num_sinos, infile, outpath, preprocessing_required, corrplan,
536     ↪norm_sx,
537             norm_dx, flat_end, half_half, half_half_line, ext_fov, ext_fov_rot_right,
538     ↪ext_fov_overlap, ringrem,
539             phaseretrieval_required, phrtmethod, phrt_param1, phrt_param2, energy,
540     ↪distance, pixsize, phrtpad,
541             approx_win, angles, offset, logtrsf, param1, circle, scale, overpad,
542     ↪reconmethod, zerone_mode,
543             dset_min, dset_max, decim_factor, downsc_factor, corr_offset, postprocess_
544     ↪required, convert_opt,
545             crop_opt, nr_threads, angles_from, angles_to, logfilename, lock, slice_
546     ↪prefix )
547
548     # Sample:
549     # 26 C:\Temp\dataset42.tdf C:\Temp\test_angles 3.1416 -72.0 shepp-logan 1.0 False_
550     ↪True True True True False 5 False False 100 0 0 False rivers:11;0 False 0.0 FBP_
551     ↪CUDA 1 4 False - - False 0 1.0 1000.0 22 150 2.2 True 16 False C:\Temp 1 1148 1248_
552     ↪slice C:\Temp\log_angles_00.txt
553
554
555
556
557 if __name__ == "__main__":
558     main(argv[1:])

```

## tools\_extractdata

This section contains the tools\_extractdata script.

Download file: tools\_extractdata.py

```

1 #####
2 # (C) 2016 Elettra - Sincrotrone Trieste S.C.p.A.. All rights reserved. #
3 # # #
4 # # #
5 # This file is part of STP-Core, the Python core of SYRMEP Tomo Project, #
6 # a software tool for the reconstruction of experimental CT datasets. #
7 # # #
8 # STP-Core is free software: you can redistribute it and/or modify it #
9 # under the terms of the GNU General Public License as published by the #
10 # Free Software Foundation, either version 3 of the License, or (at your #
11 # option) any later version. #
12 # # #
13 # STP-Core is distributed in the hope that it will be useful, but WITHOUT #
14 # ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or #
15 # FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License #
16 # for more details. #
17 # # #
18 # You should have received a copy of the GNU General Public License #
19 # along with STP-Core. If not, see <http://www.gnu.org/licenses/>. #
20 # # #

```

```

21 #####
22
23 #
24 # Author: Francesco Brun
25 # Last modified: July, 8th 2016
26 #
27
28
29 import os
30 import os.path
31 import numpy
32 import time
33
34 from sys import argv, exit
35 from h5py import File as getHDF5
36 from numpy import float32
37
38 # pystp-specific:
39 import stp_core.io.tdf as tdf
40
41 def main(argv):
42     """Extract a 2D image (projection or sinogram) from the input TDF file_
↳(DataExchange HDF5) and
43     creates a 32-bit RAW file to disk.
44
45     Parameters
46     -----
47     argv[0] : string
48         The absolute path of the input TDF.
49
50     argv[1] : int
51         The relative position of the image within the dataset.
52
53     argv[2] : string
54         One of the following options: 'tomo', 'sino', 'flat', 'dark'.
55
56     argv[3] : string
57         The absolute path of the output 32-bit RAW image file. Filename will be_
↳modified by adding
58         image width, image height, minimum and maximum value of the input TDF dataset.
59
60     Example
61     -----
62     tools_extractdata "S:\\dataset.tdf" 128 tomo "R:\\proj"
63
64     """
65     try:
66         #
67         # Get input parameters:
68         #
69         infile = argv[0]
70         index = int(argv[1])
71         imtype = argv[2]
72         outfile = argv[3]
73
74         #
75         # Body
76         #

```

```

77
78     # Check if file exists:
79     if not os.path.exists(infile):
80         #log = open(logfilename, "a")
81         #log.write(os.linesep + "\tError: input TDF file not found. Process will_
↪end.")
82
83         #log.close()
84         exit()
85
86     # Open the HDF5 file:
87
88     f = getHDF5( infile, 'r' )
89     if (imtype == 'sino'):
90         if "/tomo" in f:
91             dset = f['tomo']
92         else:
93             dset = f['exchange/data']
94         im = tdf.read_sino( dset, index )
95     elif (imtype == 'dark'):
96         if "/dark" in f:
97             dset = f['dark']
98         else:
99             dset = f['exchange/data_dark']
100        im = tdf.read_tomo( dset, index )
101    elif (imtype == 'flat'):
102        if "/flat" in f:
103            dset = f['flat']
104        else:
105            dset = f['exchange/data_white']
106        im = tdf.read_tomo( dset, index )
107    else:
108        if "/tomo" in f:
109            dset = f['tomo']
110        else:
111            dset = f['exchange/data']
112        im = tdf.read_tomo( dset, index )
113
114    min = float(numpy.nanmin(im[:]))
115    max = float(numpy.nanmax(im[:]))
116
117    # Get global attributes (if any):
118    try:
119        if ('version' in f.attrs):
120            if (f.attrs['version'] == '1.0'):
121                min = float(dset_tomo.attrs['min'])
122                max = float(dset_tomo.attrs['max'])
123    except:
124        pass
125
126    f.close()
127
128    # Cast type:
129    im = im.astype(float32)
130
131    # Modify file name:
132    outfile = outfile + '_' + str(im.shape[1]) + 'x' + str(im.shape[0]) + '_' +_
↪str(min) + '$' + str(max)

```

```

133         # Write RAW data to disk:
134         im.tofile(outfile)
135
136     except:
137
138         exit()
139
140
141 if __name__ == "__main__":
142     main(argv[1:])

```

## tools\_guesscenter

This section contains the `tools_guesscenter` script.

Download file: `tools_guesscenter.py`

```

1 #####
2 # (C) 2016 Elettra - Sincrotrone Trieste S.C.p.A.. All rights reserved. #
3 # # #
4 # # #
5 # This file is part of STP-Core, the Python core of SYRMEP Tomo Project, #
6 # a software tool for the reconstruction of experimental CT datasets. #
7 # # #
8 # STP-Core is free software: you can redistribute it and/or modify it #
9 # under the terms of the GNU General Public License as published by the #
10 # Free Software Foundation, either version 3 of the License, or (at your #
11 # option) any later version. #
12 # # #
13 # STP-Core is distributed in the hope that it will be useful, but WITHOUT #
14 # ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or #
15 # FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License #
16 # for more details. #
17 # # #
18 # You should have received a copy of the GNU General Public License #
19 # along with STP-Core. If not, see <http://www.gnu.org/licenses/>. #
20 # # #
21 #####
22
23 #
24 # Author: Francesco Brun
25 # Last modified: Sept, 28th 2016
26 #
27
28 from math import pi
29 from numpy import float32, double, finfo, ndarray
30 from scipy.misc import imresize #scipy 0.12
31 from os import sep, remove
32 from os.path import basename
33 from sys import argv
34 from h5py import File as getHDF5
35
36 from pyfftw.interfaces.cache import enable as pyfftw_cache_enable, disable as pyfftw_
37 ↪cache_disable
38 from pyfftw.interfaces.cache import set_keepalive_time as pyfftw_set_keepalive_time
39
40 import time

```

```

40
41 # pystp-specific:
42 import stp_core.io.tdf as tdf
43 import stp_core.utils.findcenter as findcenter
44 from stp_core.utils.caching import cache2plan, plan2cache
45 from stp_core.preprocess.extract_flatdark import extract_flatdark
46 from stp_core.preprocess.flat_fielding import flat_fielding
47
48 from tiff file import imread, imsave # only for debug
49
50 def main(argv):
51     """Try to guess the center of rotation of the input CT dataset.
52
53     Parameters
54     -----
55     infile   : array_like
56               HDF5 input dataset
57
58     outfile  : string
59               Full path where the identified center of rotation will be written as output
60
61     scale    : int
62               If sub-pixel precision is interesting, use e.g. 2.0 to get a center of
63     ↪rotation of .5 value. Use 1.0 if sub-pixel precision is not required
64
65     angles   : int
66               Total number of angles of the input dataset
67
68     proj_from : int
69               Initial projections to consider for the assumed angles
70
71     proj_to   : int
72               Final projections to consider for the assumed angles
73
74     method   : string
75               (not implemented yet)
76
77     tmppath  : string
78               Temporary path where look for cached flat/dark files
79
80     """
81     # Get path:
82     infile = argv[0]           # The HDF5 file on the
83     outfile = argv[1]         # The txt file with the proposed center
84     scale = float(argv[2])
85     angles = float(argv[3])
86     proj_from = int(argv[4])
87     proj_to = int(argv[5])
88     method = argv[6]
89     tmppath = argv[7]
90     if not tmppath.endswith(sep): tmppath += sep
91
92     pyfftw_cache_disable()
93     pyfftw_cache_enable()
94     pyfftw_set_keepalive_time(1800)
95
96     # Create a silly temporary log:

```



```

97  tmplog = tmppath + basename(infile) + str(time.time())
98
99  # Open the HDF5 file (take into account also older TDF versions):
100 f_in = getHDF5( infile, 'r' )
101 if "/tomo" in f_in:
102     dset = f_in['tomo']
103 else:
104     dset = f_in['exchange/data']
105 num_proj = tdf.get_nr_projs(dset)
106 num_sinos = tdf.get_nr_sinos(dset)
107
108 # Get flats and darks from cache or from file:
109 try:
110     corrplan = cache2plan(infile, tmppath)
111 except Exception as e:
112     #print "Error(s) when reading from cache"
113     corrplan = extract_flatdark(f_in, True, tmplog)
114     remove(tmplog)
115     plan2cache(corrplan, infile, tmppath)
116
117 # Get first and the 180 deg projections:
118 im1 = tdf.read_tomo(dset,proj_from).astype(float32)
119
120 idx = int(round( (proj_to - proj_from)/angles * pi)) + proj_from
121 im2 = tdf.read_tomo(dset,idx).astype(float32)
122
123 # Apply simple flat fielding (if applicable):
124 if (isinstance(corrplan['im_flat_after'], ndarray) and isinstance(corrplan['im_
↵flat'], ndarray) and
125     isinstance(corrplan['im_dark'], ndarray) and isinstance(corrplan['im_dark_
↵after'], ndarray)) :
126     im1 = ((abs(im1 - corrplan['im_dark'])) / (abs(corrplan['im_flat'] - corrplan[
↵'im_dark']))
127            + finfo(float32).eps)).astype(float32)
128     im2 = ((abs(im2 - corrplan['im_dark_after'])) / (abs(corrplan['im_flat_after
↵'] - corrplan['im_dark_after']))
129            + finfo(float32).eps)).astype(float32)
130
131 # Scale projections (if required) to get subpixel estimation:
132 if ( abs(scale - 1.0) > finfo(float32).eps ) :
133     im1 = imresize(im1, (int(round(scale*im1.shape[0])), int(round(scale*im1.
↵shape[1]))), interp='bicubic', mode='F');
134     im2 = imresize(im2, (int(round(scale*im2.shape[0])), int(round(scale*im2.
↵shape[1]))), interp='bicubic', mode='F');
135
136 # Find the center (flipping left-right im2):
137 cen = findcenter.usecorrelation(im1, im2[ :,:-1])
138 cen = cen / scale
139
140 # Print center to output file:
141 text_file = open(outfile, "w")
142 text_file.write(str(int(cen)))
143 text_file.close()
144
145 # Close input HDF5:
146 f_in.close()
147
148 if __name__ == "__main__":

```

```
149 main(argv[1:])
```

## tools\_raw2tiff32

This section contains the tools\_raw2tiff32 script.

Download file: tools\_raw2tiff32.py

```

1 #####
2 # (C) 2016 Elettra - Sincrotrone Trieste S.C.p.A.. All rights reserved. #
3 # #
4 # #
5 # This file is part of STP-Core, the Python core of SYRMEP Tomo Project, #
6 # a software tool for the reconstruction of experimental CT datasets. #
7 # #
8 # STP-Core is free software: you can redistribute it and/or modify it #
9 # under the terms of the GNU General Public License as published by the #
10 # Free Software Foundation, either version 3 of the License, or (at your #
11 # option) any later version. #
12 # #
13 # STP-Core is distributed in the hope that it will be useful, but WITHOUT #
14 # ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or #
15 # FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License #
16 # for more details. #
17 # #
18 # You should have received a copy of the GNU General Public License #
19 # along with STP-Core. If not, see <http://www.gnu.org/licenses/>. #
20 # #
21 #####
22
23 #
24 # Author: Francesco Brun
25 # Last modified: July, 8th 2016
26 #
27
28 import os
29 import os.path
30
31 from sys import argv, exit
32 from tifffile import *
33 from numpy import zeros, fromfile, float32
34
35 def main(argv):
36     """Convert an input 32-bit RAW image to TIFF format
37
38     Parameters
39     -----
40     argv[0] : string
41         The absolute path of input 32-bit RAW image file.
42
43     argv[1] : string
44         The absolute path of output 32-bit TIFF image file.
45
46     argv[2] : int
47         Width of the input RAW image.
48
49     argv[3] : int

```

```

50     Height of the input RAW image.
51
52     Example
53     -----
54     tools_raw2tiff32 "R:\\slice.raw" "R:\\slice.tiff" 2048 2048
55
56     """
57     #
58     # Get the parameters:
59     #
60     infile = argv[0]
61     outfile = argv[1]
62     width = int(argv[2])
63     height = int(argv[3])
64
65     #
66     # Body
67     #
68
69     # Check if file exists:
70     if not os.path.exists(infile):
71         exit()
72
73     try:
74         # Prepare RAW matrix:
75         im = zeros((width,height), dtype=float32)
76
77         # Read RAW file:
78         im = fromfile(infile, float32).reshape((height,width))
79
80         # Save TIFF 32:
81         imsave(outfile, im)
82
83     except:
84         exit()
85
86 if __name__ == "__main__":
87     main(argv[1:])

```

## tools\_multioffset

This section contains the tools\_multioffset script.

Download file: [tools\\_multioffset.py](#)

```

1  #####
2  # (C) 2016 Elettra - Sincrotrone Trieste S.C.p.A.. All rights reserved. #
3  # # #
4  # # #
5  # This file is part of STP-Core, the Python core of SYRMEP Tomo Project, #
6  # a software tool for the reconstruction of experimental CT datasets. #
7  # # #
8  # STP-Core is free software: you can redistribute it and/or modify it #
9  # under the terms of the GNU General Public License as published by the #
10 # Free Software Foundation, either version 3 of the License, or (at your #
11 # option) any later version. #
12 # # #

```

```

13 # STP-Core is distributed in the hope that it will be useful, but WITHOUT #
14 # ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or #
15 # FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License #
16 # for more details. #
17 # #
18 # You should have received a copy of the GNU General Public License #
19 # along with STP-Core. If not, see <http://www.gnu.org/licenses/>. #
20 # #
21 #####
22
23 #
24 # Author: Francesco Brun
25 # Last modified: Sept, 28th 2016
26 #
27
28 # python:
29 from sys import argv, exit
30 from os import remove, sep, linesep, listdir, makedirs
31 from os.path import exists, dirname, basename, splitext
32 from numpy import array, finfo, copy, float32, double, amin, amax, tile, concatenate,
↳asarray
33 from numpy import empty, reshape, log as nplog, arange, squeeze, fromfile, ndarray,
↳where, meshgrid
34 from time import time
35 from multiprocessing import Process, Lock
36
37 # pystp-specific:
38 from stp_core.preprocess.extfov_correction import extfov_correction
39 from stp_core.preprocess.flat_fielding import flat_fielding
40 from stp_core.preprocess.ring_correction import ring_correction
41 from stp_core.preprocess.extract_flatdark import extract_flatdark
42
43 from stp_core.phaseretrieval.tiehom import tiehom, tiehom_plan
44 from stp_core.phaseretrieval.phrt import phrt, phrt_plan
45
46 from stp_core.reconstruct.rec_astra import recon_astra_fbp, recon_astra_iterative
47 from stp_core.reconstruct.rec_fista_tv import recon_fista_tv
48 from stp_core.reconstruct.rec_mr_fbp import recon_mr_fbp
49 from stp_core.reconstruct.rec_gridrec import recon_gridrec
50
51 from stp_core.postprocess.postprocess import postprocess
52
53 from stp_core.utils.padding import upperPowerOfTwo, padImage, padSmoothWidth
54 from stp_core.utils.caching import cache2plan, plan2cache
55
56 from tiff file import imread, imsave
57 from h5py import File as getHDF5
58 import stp_core.io.tdf as tdf
59
60
61 def write_log(lock, fname, logfilename):
62     """To do...
63
64     """
65     lock.acquire()
66     try:
67         # Print out execution time:
68         log = open(logfilename, "a")

```

```

69     log.write(linsep + "\t%s reconstructed." % basename(fname))
70     log.close()
71
72     finally:
73         lock.release()
74
75 def reconstruct(im, angles, angles_projfrom, angles_projto, offset, logtransform,
76 ↪param1, circle, scale, pad, method,
77 ↪zerone_mode, dset_min, dset_max, corr_offset, postprocess_required,
78 ↪convert_opt,
79 ↪crop_opt, start, end, outpath, sino_idx, downsc_factor, logfilename,
80 ↪lock, slice_prefix):
81     """Reconstruct a sinogram with FBP algorithm (from ASTRA toolbox).
82
83     Parameters
84     -----
85     im1 : array_like
86         Sinogram image data as numpy array.
87     center : float
88         Offset of the center of rotation to use for the tomographic
89         reconstruction with respect to the half of sinogram width
90         (default=0, i.e. half width).
91     logtransform : boolean
92         Apply logarithmic transformation before reconstruction (default=True).
93     filter : string
94         Filter to apply before the application of the reconstruction algorithm.
95 ↪Filter
96     types are: ram-lak, shepp-logan, cosine, hamming, hann, tukey, lanczos,
97 ↪triangular,
98     gaussian, barlett-hann, blackman, nuttall, blackman-harris, blackman-nuttall,
99     flat-top, kaiser, parzen.
100     circle : boolean
101         Create a circle in the reconstructed image and set to zero pixels outside the
102         circle (default=False).
103
104     """
105     # Copy required due to multithreading:
106     im_f = im
107
108     # Decimate projections if required:
109     #if decim_factor > 1:
110     #    im = im[:,::decim_factor,:]
111
112     # Upscale projections (if required):
113     if (abs(scale - 1.0) > finfo(float32).eps):
114         siz_orig1 = im_f.shape[1]
115         im_f = imresize(im_f, (im_f.shape[0], int(round(scale * im_f.shape[1]))),
116 ↪interp='bicubic', mode='F')
117         offset = int(offset * scale)
118
119     # Loop for all the required offsets for the center of rotation:
120     for i in range(int(round(start)), int(round(end)) + 1, downsc_factor):
121
122         offset = int(round(i/downsc_factor))
123
124     # Apply transformation for changes in the center of rotation:
125     if (offset != 0):
126         if (offset >= 0):

```

```

121         im_f = im_f[:, :-offset]
122
123         tmp = im_f[:,0] # Get first column
124         tmp = tile(tmp, (offset,1)) # Replicate the first column the right_
↪number of times
125         im_f = concatenate((tmp.T,im_f), axis=1) # Concatenate tmp before the_
↪image
126
127     else:
128         im_f = im_f[:,abs(offset):]
129
130         tmp = im_f[:,im_f.shape[1] - 1] # Get last column
131         tmp = tile(tmp, (abs(offset),1)) # Replicate the last column the_
↪right number of times
132         im_f = concatenate((im_f,tmp.T), axis=1) # Concatenate tmp after the_
↪image
133
134     # Downscale projections (without pixel averaging):
135     #if downsc_factor > 1:
136     #    im = im[:,::downsc_factor]
137
138     # Scale image to [0,1] range (if required):
139     if (zerone_mode):
140
141         #print dset_min
142         #print dset_max
143         #print numpy.amin(im_f[:])
144         #print numpy.amax(im_f[:])
145         #im_f = (im_f - dset_min) / (dset_max - dset_min)
146
147         # Cheating the whole process:
148         im_f = (im_f - numpy.amin(im_f[:])) / (numpy.amax(im_f[:]) - numpy.
↪amin(im_f[:]))
149
150     # Apply log transform:
151     if (logtransform == True):
152         im_f[im_f <= finfo(float32).eps] = finfo(float32).eps
153         im_f = -nplog(im_f + corr_offset)
154
155     # Replicate pad image to double the width:
156     if (pad):
157
158         dim_o = im_f.shape[1]
159         n_pad = im_f.shape[1] + im_f.shape[1] / 2
160         marg = (n_pad - dim_o) / 2
161
162         # Pad image:
163         im_f = padSmoothWidth(im_f, n_pad)
164
165     # Perform the actual reconstruction:
166     if (method.startswith('FBP')):
167         im_f = recon_astra_fbp(im_f, angles, method, param1)
168     elif (method == 'MR-FBP_CUDA'):
169         im_f = recon_mr_fbp(im_f, angles)
170     elif (method == 'FISTA-TV_CUDA'):
171         im_f = recon_fista_tv(im_f, angles, param1, param1)
172     elif (method == 'MLEM'):
173         im_f = recon_tomopy_iterative(im_f, angles, method, param1)

```

```

174     elif (method == 'GRIDREC'):
175         [im_f, im_f] = recon_gridrec(im_f, im_f, angles, param1)
176     else:
177         im_f = recon_astra_iterative(im_f, angles, method, param1, zerone_mode)
178
179
180     # Crop:
181     if (pad):
182         im_f = im_f[marg:dim_o + marg, marg:dim_o + marg]
183
184     # Resize (if necessary):
185     if (abs(scale - 1.0) > finfo(float32).eps):
186         im_f = imresize(im_f, (siz_orig1, siz_orig1), interp='nearest', mode='F')
187
188     # Apply post-processing (if required):
189     if postprocess_required:
190         im_f = postprocess(im_f, convert_opt, crop_opt)
191     else:
192         # Create the circle mask for fancy output:
193         if (circle == True):
194             siz = im_f.shape[1]
195             if siz % 2:
196                 rang = arange(-siz / 2 + 1, siz / 2 + 1)
197             else:
198                 rang = arange(-siz / 2, siz / 2)
199             x,y = meshgrid(rang,rang)
200             z = x ** 2 + y ** 2
201             a = (z < (siz / 2 - int(round(abs(offset)/downsc_factor)) ) ** 2)
202             im_f = im_f * a
203
204     # Write down reconstructed image (file name modified with metadata):
205     if ( i >= 0 ):
206         fname = outpath + slice_prefix + '_' + str(sino_idx).zfill(4) + '_proj=' +
↪ str(angles_projto - angles_projfrom) + '_col=' + str((im_f.shape[1] +
↪ offset)*downsc_factor).zfill(4) + '_off=' + str(abs(offset*downsc_factor)).
↪ zfill(4) + '.tif'
207     else:
208         fname = outpath + slice_prefix + '_' + str(sino_idx).zfill(4) + '_proj=' +
↪ str(angles_projto - angles_projfrom) + '_col=' + str((im_f.shape[1] +
↪ offset)*downsc_factor).zfill(4) + '_off=' + str(abs(offset*downsc_factor)).
↪ zfill(4) + '.tif'
209
210     imsave(fname, im_f)
211
212     # Restore original image for next step:
213     im_f = im
214
215     # Write log (atomic procedure - lock used):
216     write_log(lock, fname, logfilename )
217
218
219 def process(sino_idx, num_sinos, infile, outpath, preprocessing_required, corr_plan,
↪ norm_sx, norm_dx, flat_end, half_half,
220           half_half_line, ext_fov, ext_fov_rot_right, ext_fov_overlap, ringrem,
↪ phaseretrieval_required, phrtmethod,
221           phrt_param1, phrt_param2, energy, distance, pixsize, phrtpad, approx_win,
↪ angles, angles_projfrom, angles_projto,
222           offset, logtransform, param1, circle, scale, pad, method,

```

```

223     zerone_mode, dset_min, dset_max, decim_factor, downsc_factor, corr_offset,
↳ postprocess_required, convert_opt,
224     crop_opt, nr_threads, off_from, off_to, logfilename, lock, slice_prefix):
225     """To do...
226
227     """
228     slice_nr = sino_idx
229
230     # Perform reconstruction (on-the-fly preprocessing and phase retrieval, if_
↳required):
231     if (phaseretrieval_required):
232
233         # In this case a bunch of sinograms is loaded into memory:
234
235         #
236         # Load the temporary data structure reading the input TDF file.
237         # To know the right dimension the first sinogram is pre-processed.
238         #
239
240         # Open the TDF file and get the dataset:
241         f_in = getHDF5(infile, 'r')
242         if "/tomo" in f_in:
243             dset = f_in['tomo']
244         else:
245             dset = f_in['exchange/data']
246
247         # Downscaling and decimation factors considered when determining the_
↳approximation window:
248         xrange = arange(sino_idx - approx_win*downsc_factor/2, sino_idx + approx_
↳win*downsc_factor/2, downsc_factor)
249         xrange = xrange[ (xrange >= 0) ]
250         xrange = xrange[ (xrange < num_sinos) ]
251         approx_win = xrange.shape[0]
252
253         # Approximation window cannot be odd:
254         if (approx_win % 2 == 1):
255             approx_win = approx_win-1
256             xrange      = xrange[0:approx_win]
257
258         # Read one sinogram to get the proper dimensions:
259         test_im = tdf.read_sino(dset, xrange[0]).astype(float32)
260         test_im = test_im[:,::decim_factor, ::downsc_factor]
261
262         # Perform the pre-processing of the first sinogram to get the right dimension:
263         if (preprocessing_required):
264             test_im = flat_fielding (test_im, xrange[0]/downsc_factor, corr_plan,
↳flat_end, half_half,
265                                     half_half_line/decim_factor, norm_sx, norm_
↳dx).astype(float32)
266             test_im = extfov_correction (test_im, ext_fov, ext_fov_rot_right, ext_fov_
↳overlap/downsc_factor).astype(float32)
267             test_im = ring_correction (test_im, ringrem, flat_end, corr_plan['skip_
↳flat_after'], half_half,
268                                     half_half_line/decim_factor, ext_fov).
↳astype(float32)
269
270         # Now we can allocate memory for the bunch of slices:
271         tmp_im = empty((approx_win, test_im.shape[0], test_im.shape[1]),
↳dtype=float32)

```



```

272     tmp_im[0,:,:] = test_im
273
274     # Reading all the the sinos from TDF file and close:
275     for ct in range(1, approx_win):
276
277         test_im = tdf.read_sino(dset, zrange[ct]).astype(float32)
278
279         # Apply projection removal (if required):
280         test_im = test_im[angles_projfrom:angles_projto, :]
281
282         # Apply decimation and downscaling (if required):
283         test_im = test_im[:,::decim_factor, ::downsc_factor]
284
285         # Perform the pre-processing for each sinogram of the bunch:
286         if (preprocessing_required):
287             test_im = flat_fielding (test_im, zrange[ct]/downsc_factor, corr_plan,
↪ flat_end, half_half,
288                                     half_half_line/decim_factor, norm_sx,
↪ norm_dx).astype(float32)
289             test_im = extfov_correction (test_im, ext_fov, ext_fov_rot_right, ext_
↪ fov_overlap/downsc_factor).astype(float32)
290             test_im = ring_correction (test_im, ringrem, flat_end, corr_plan[
↪ 'skip_flat_after'], half_half,
291                                     half_half_line/decim_factor, ext_fov).
↪ astype(float32)
292
293             tmp_im[ct,:,:] = test_im
294
295         f_in.close()
296
297         # Now everything has to refer to a downscaled dataset:
298         sino_idx = ((zrange == sino_idx).nonzero())
299
300         #
301         # Perform phase retrieval:
302         #
303
304         # Prepare the plan:
305         if (phrtmethod == 0):
306             # Paganin's:
307             phrtplan = tiehom_plan (tmp_im[:,0,:], phrt_param1, phrt_param2, energy,
↪ distance, pixsize*downsc_factor, padding=phrtpad)
308         else:
309             phrtplan = phrt_plan (tmp_im[:,0,:], energy, distance, pixsize*downsc_
↪ factor, phrt_param2, phrt_param1, phrtmethod, padding=phrtpad)
310
311         # Process each projection (whose height depends on the size of the bunch):
312         for ct in range(0, tmp_im.shape[1]):
313             if (phrtmethod == 0):
314                 tmp_im[:,ct,:] = tiehom(tmp_im[:,ct,:], phrtplan).astype(float32)
↪
315             else:
316                 tmp_im[:,ct,:] = phrt(tmp_im[:,ct,:], phrtplan, phrtmethod).
↪ astype(float32)
317
318         # Extract the requested sinogram:
319         im = tmp_im[sino_idx[0],:,:].squeeze()
320

```

```

321     else:
322
323         # Read only one sinogram:
324         f_in = getHDF5(infile, 'r')
325         if "/tomo" in f_in:
326             dset = f_in['tomo']
327         else:
328             dset = f_in['exchange/data']
329         im = tdf.read_sino(dset, sino_idx).astype(float32)
330         f_in.close()
331
332         # Apply projection removal (if required):
333         im = im[angles_projfrom:angles_projto, :]
334
335         # Downscale and decimate the sinogram:
336         im = im[:,::decim_factor,::downsc_factor]
337         sino_idx = sino_idx/downsc_factor
338
339         # Perform the preprocessing of the sinogram (if required):
340         if (preprocessing_required):
341             im = flat_fielding(im, sino_idx, corr_plan, flat_end, half_half, half_
↪ half_line/decim_factor,
342                               norm_sx, norm_dx).astype(float32)
343             im = extfov_correction(im, ext_fov, ext_fov_rot_right, ext_fov_overlap)
344             im = ring_correction(im, ringrem, flat_end, corr_plan['skip_flat_after'],
↪ half_half,
345                               half_half_line/decim_factor, ext_fov)
346
347         # Log infos:
348         log = open(logfilename, "a")
349         log.write(linesep + "\tPerforming reconstruction with multiple centers of
↪ rotation...")
350         log.write(linesep + "\t-----")
351         log.close()
352
353         # Split the computation into multiple processes:
354         for num in range(nr_threads):
355             start = ( (off_to - off_from + 1) / nr_threads)*num + off_from
356             if (num == nr_threads - 1):
357                 end = off_to
358             else:
359                 end = ( (off_to - off_from + 1) / nr_threads)*(num + 1) + off_from - 1
360
361             #Process(target=reconstruct, args=(im, angles, angles_projfrom, angles_projto,
↪ offset/downsc_factor,
362           #           logtransform, param1, circle, scale, pad, method,
363           #           zerone_mode, dset_min, dset_max, corr_offset, postprocess_
↪ required, convert_opt,
364           #           crop_opt, start, end, outpath, slice_nr, downsc_factor,
↪ logfilename, lock, slice_prefix)).start()
365
366
367         # Actual reconstruction:
368         reconstruct(im, angles, angles_projfrom, angles_projto, offset/downsc_factor,
↪ logtransform,
369           param1, circle, scale, pad, method,
370           zerone_mode, dset_min, dset_max, corr_offset, postprocess_
↪ required, convert_opt,

```

```

371         crop_opt, start, end, outpath, slice_nr, downsc_factor,
↪ logfilename, lock, slice_prefix)
372
373
374
375
376 def main(argv):
377     """To do...
378
379
380
381     """
382     lock = Lock()
383     skip_flat = False
384     skip_flat_after = True
385
386     # Get the from and to number of files to process:
387     sino_idx = int(argv[0])
388
389     # Get paths:
390     infile = argv[1]
391     outpath = argv[2]
392
393     # Essential reconstruction parameters::
394     angles = float(argv[3])
395     off_step = float(argv[4])
396     param1 = argv[5]
397     scale = int(float(argv[6]))
398
399     overpad = True if argv[7] == "True" else False
400     logtrsf = True if argv[8] == "True" else False
401     circle = True if argv[9] == "True" else False
402
403     # Parameters for on-the-fly pre-processing:
404     preprocessing_required = True if argv[10] == "True" else False
405     flat_end = True if argv[11] == "True" else False
406     half_half = True if argv[12] == "True" else False
407
408     half_half_line = int(argv[13])
409
410     ext_fov = True if argv[14] == "True" else False
411
412     norm_sx = int(argv[17])
413     norm_dx = int(argv[18])
414
415     ext_fov_rot_right = argv[15]
416     if ext_fov_rot_right == "True":
417         ext_fov_rot_right = True
418         if (ext_fov):
419             norm_sx = 0
420     else:
421         ext_fov_rot_right = False
422         if (ext_fov):
423             norm_dx = 0
424
425     ext_fov_overlap = int(argv[16])
426
427     skip_ringrem = True if argv[19] == "True" else False

```

```
428 ringrem = argv[20]
429
430 # Extra reconstruction parameters:
431 zerone_mode = True if argv[21] == "True" else False
432 corr_offset = float(argv[22])
433
434 reconmethod = argv[23]
435
436 decim_factor = int(argv[24])
437 downsc_factor = int(argv[25])
438
439 # Parameters for postprocessing:
440 postprocess_required = True if argv[26] == "True" else False
441 convert_opt = argv[27]
442 crop_opt = argv[28]
443
444 # Parameters for on-the-fly phase retrieval:
445 phaseretrieval_required = True if argv[29] == "True" else False
446 phrtmethod = int(argv[30])
447 phrt_param1 = double(argv[31]) # param1( e.g. regParam, or beta)
448 phrt_param2 = double(argv[32]) # param2( e.g. thresh or delta)
449 energy = double(argv[33])
450 distance = double(argv[34])
451 pixsize = double(argv[35]) / 1000.0 # pixsize from micron to mm:
452 phrtpad = True if argv[36] == "True" else False
453 approx_win = int(argv[37])
454
455 angles_projfrom = int(argv[38])
456 angles_projto = int(argv[39])
457
458 preprocessingplan_fromcache = True if argv[40] == "True" else False
459 tmppath = argv[41]
460 if not tmppath.endswith(sep): tmppath += sep
461
462 nr_threads = int(argv[42])
463 off_from = float(argv[43])
464 off_to = float(argv[44])
465
466 slice_prefix = argv[45]
467
468 logfilename = argv[46]
469
470 if not exists(outputpath):
471     makedirs(outputpath)
472
473 if not outputpath.endswith(sep): outputpath += sep
474
475
476 # Log info:
477 log = open(logfilename, "w")
478 log.write(linesep + "\tInput dataset: %s" % (infile))
479 log.write(linesep + "\tOutput path: %s" % (outputpath))
480 log.write(linesep + "\t-----")
481 log.write(linesep + "\tLoading flat and dark images...")
482 log.close()
483
484 # Open the HDF5 file:
485 f_in = getHDF5(infile, 'r')
```

```

486 if "/tomo" in f_in:
487     dset = f_in['tomo']
488 else:
489     dset = f_in['exchange/data']
490     if "/provenance/detector_output" in f_in:
491         prov_dset = f_in['provenance/detector_output']
492
493 dset_min = -1
494 dset_max = -1
495 if (zerone_mode):
496     if ('min' in dset.attrs):
497         dset_min = float(dset.attrs['min'])
498     else:
499         zerone_mode = False
500
501     if ('max' in dset.attrs):
502         dset_max = float(dset.attrs['max'])
503     else:
504         zerone_mode = False
505
506 num_sinos = tdf.get_nr_sinos(dset) # Pay attention to the downscale factor
507
508 if (num_sinos == 0):
509     exit()
510
511 # Check extrema:
512 if (sino_idx >= num_sinos):
513     sino_idx = num_sinos - 1
514
515 # Get correction plan and phase retrieval plan (if required):
516 corrplan = 0
517 if (preprocessing_required):
518     # Load flat fielding plan either from cache (if required) or from TDF file_
↳and cache it for faster re-use:
519     if (preprocessingplan_fromcache):
520         try:
521             corrplan = cache2plan(infile, tmppath)
522         except Exception as e:
523             #print "Error(s) when reading from cache"
524             corrplan = extract_flatdark(f_in, flat_end, logfilename)
525             plan2cache(corrplan, infile, tmppath)
526     else:
527         corrplan = extract_flatdark(f_in, flat_end, logfilename)
528         plan2cache(corrplan, infile, tmppath)
529
530     # Downscale flat and dark images if necessary:
531     if isinstance(corrplan['im_flat'], ndarray):
532         corrplan['im_flat'] = corrplan['im_flat'][:,::downsc_factor,::downsc_
↳factor]
533     if isinstance(corrplan['im_dark'], ndarray):
534         corrplan['im_dark'] = corrplan['im_dark'][:,::downsc_factor,::downsc_
↳factor]
535     if isinstance(corrplan['im_flat_after'], ndarray):
536         corrplan['im_flat_after'] = corrplan['im_flat_after'][:,::downsc_factor,
↳::downsc_factor]
537     if isinstance(corrplan['im_dark_after'], ndarray):
538         corrplan['im_dark_after'] = corrplan['im_dark_after'][:,::downsc_factor,
↳::downsc_factor]

```

```

539     f_in.close()
540
541     # Log infos:
542     log = open(logfilename,"a")
543     log.write(linesep + "\tPerforming preprocessing...")
544     log.close()
545
546     # Run computation:
547     process( sino_idx, num_sinos, infile, outpath, preprocessing_required, corrplan,
548     ↪norm_sx,
549             norm_dx, flat_end, half_half, half_half_line, ext_fov, ext_fov_rot_right,
550     ↪ext_fov_overlap, ringrem,
551             phaseretrieval_required, phrtmethod, phrt_param1, phrt_param2, energy,
552     ↪distance, pixsize, phrtpad,
553             approx_win, angles, angles_projfrom, angles_projto,
554             off_step, logtrsf, param1, circle, scale, overpad, reconmethod, zerone_
555     ↪mode,
556             dset_min, dset_max, decim_factor, downsc_factor, corr_offset, postprocess_
557     ↪required, convert_opt,
558             crop_opt, nr_threads, off_from, off_to, logfilename, lock, slice_prefix )
559
560     # Sample:
561     # 26 C:\Temp\dataset42.tdf C:\Temp\test_offset 3.1416 1.0 shepp-logan 1.0 False_
562     ↪True True True False 5 False False 100 0 0 False rivers:11;0 False 0.0 FBP_
563     ↪CUDA 1 4 False - - False 0 1.0 1000.0 22 150 2.2 True 16 0 1163 False C:\Temp 1 -78_
564     ↪-70 slice C:\Temp\log_angles_00.txt
565
566 if __name__ == "__main__":
567     main(argv[1:])

```

## tools\_guessoverlap

This section contains the tools\_guessoverlap script.

Download file: [tools\\_guessoverlap.py](#)

```

1  #####
2  # (C) 2016 Elettra - Sincrotrone Trieste S.C.p.A.. All rights reserved. #
3  # # #
4  # # #
5  # This file is part of STP-Core, the Python core of SYRMEP Tomo Project, #
6  # a software tool for the reconstruction of experimental CT datasets. #
7  # # #
8  # STP-Core is free software: you can redistribute it and/or modify it #
9  # under the terms of the GNU General Public License as published by the #
10 # Free Software Foundation, either version 3 of the License, or (at your #
11 # option) any later version. #
12 # # #
13 # STP-Core is distributed in the hope that it will be useful, but WITHOUT #
14 # ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or #
15 # FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License #
16 # for more details. #
17 # # #
18 # You should have received a copy of the GNU General Public License #

```

```

19 # along with STP-Core. If not, see <http://www.gnu.org/licenses/>.      #
20 #                                                                    #
21 #####                                                                    #
22 #                                                                    #
23 #                                                                    #
24 # Author: Francesco Brun
25 # Last modified: July, 8th 2016
26 #
27
28 from math import pi
29 from numpy import float32, double, finfo, ndarray
30 from scipy.misc import imresize #scipy 0.12
31 from os import sep, remove
32 from os.path import basename
33 from sys import argv
34 from h5py import File as getHDF5
35
36 import time
37
38 # pystp-specific:
39 import stp_core.io.tdf as tdf
40 import stp_core.utils.findcenter as findcenter
41 from stp_core.utils.caching import cache2plan, plan2cache
42 from stp_core.preprocess.extract_flatdark import extract_flatdark
43
44
45 def main(argv):
46     """Try to guess the amount of overlap in the case of extended FOV CT.
47
48     Parameters
49     -----
50     infile : array_like
51             HDF5 input dataset
52
53     outfile : string
54             Full path where the identified overlap will be written as output
55
56     scale : int
57             If sub-pixel precision is interesting, use e.g. 2.0 to get an overlap
58             of .5 value. Use 1.0 if sub-pixel precision is not required
59
60     tmppath : int
61             Temporary path where look for cached flat/dark files
62
63     """
64
65     # Get path:
66     infile = argv[0] # The HDF5 file on the SSD
67     outfile = argv[1] # The txt file with the proposed center
68     scale = float(argv[2])
69     tmppath = argv[3]
70     if not tmppath.endswith(sep): tmppath += sep
71
72     # Create a silly temporary log:
73     tmplog = tmppath + basename(infile) + str(time.time())
74
75
76     # Open the HDF5 file:

```

```

77 f_in = getHDF5( infile, 'r' )
78 if "/tomo" in f_in:
79     dset = f_in['tomo']
80 else:
81     dset = f_in['exchange/data']
82 num_proj = tdf.get_nr_projs(dset)
83
84
85 # Get first and 180 deg projections:
86 im1 = tdf.read_tomo(dset,0).astype(float32)
87 im2 = tdf.read_tomo(dset,num_proj/2).astype(float32)
88
89
90 # Get flats and darks from cache or from file:
91 try:
92     corrplan = cache2plan(infile, tmpopath)
93 except Exception as e:
94     #print "Error(s) when reading from cache"
95     corrplan = extract_flatdark(f_in, True, tmplog)
96     remove(tmplog)
97     plan2cache(corrplan, infile, tmpopath)
98
99 # Apply simple flat fielding (if applicable):
100 if (isinstance(corrplan['im_flat_after'], ndarray) and isinstance(corrplan['im_
↪flat'], ndarray) and
101     isinstance(corrplan['im_dark'], ndarray) and isinstance(corrplan['im_dark_
↪after'], ndarray)) :
102     im1 = ((abs(im1 - corrplan['im_dark'])) / (abs(corrplan['im_flat'] - corrplan[
↪'im_dark'] + finfo(float32).eps)).astype(float32)
103     im2 = ((abs(im2 - corrplan['im_dark_after'])) / (abs(corrplan['im_flat_after
↪'] - corrplan['im_dark_after'] + finfo(float32).eps)).astype(float32)
104
105
106 # Scale projections (if required) to get subpixel estimation:
107 if ( abs(scale - 1.0) > finfo(float32).eps ):
108     im1 = imresize(im1, (int(round(scale*im1.shape[0])), int(round(scale*im1.
↪shape[1])), interp='bicubic', mode='F');
109     im2 = imresize(im2, (int(round(scale*im2.shape[0])), int(round(scale*im2.
↪shape[1])), interp='bicubic', mode='F');
110
111
112 # Find the center (flipping left-right im2): DISTINGUISH BETWEEN AIR ON THE RIGHT_
↪AND ON THE LEFT??????
113 cen = findcenter.usecorrelation(im1, im2[ :,:-1])
114 cen = (cen / scale)*2.0
115
116 # Print center to output file:
117 text_file = open(outfile, "w")
118 text_file.write(str(int(abs(cen))))
119 text_file.close()
120
121 # Close input HDF5:
122 f_in.close()
123
124 if __name__ == "__main__":
125     main(argv[1:])

```



## preview\_preprocessing

This section contains the preview\_preprocessing script.

Download file: `preview_preprocessing.py`

```

1 #####
2 # (C) 2016 Elettra - Sincrotrone Trieste S.C.p.A.. All rights reserved. #
3 # #
4 # #
5 # This file is part of STP-Core, the Python core of SYRMEP Tomo Project, #
6 # a software tool for the reconstruction of experimental CT datasets. #
7 # #
8 # STP-Core is free software: you can redistribute it and/or modify it #
9 # under the terms of the GNU General Public License as published by the #
10 # Free Software Foundation, either version 3 of the License, or (at your #
11 # option) any later version. #
12 # #
13 # STP-Core is distributed in the hope that it will be useful, but WITHOUT #
14 # ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or #
15 # FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License #
16 # for more details. #
17 # #
18 # You should have received a copy of the GNU General Public License #
19 # along with STP-Core. If not, see <http://www.gnu.org/licenses/>. #
20 # #
21 #####
22
23 #
24 # Author: Francesco Brun
25 # Last modified: Sept, 28th 2016
26 #
27
28 from sys import argv, exit
29 from os import remove, sep, linesep
30 from os.path import exists
31 from numpy import float32, nanmin, nanmax, isscalar
32 from time import time
33 from multiprocessing import Process, Lock
34
35 # pystp-specific:
36 from stp_core.preprocess.extfov_correction import extfov_correction
37 from stp_core.preprocess.flat_fielding import flat_fielding
38 from stp_core.preprocess.dynamic_flatfielding import dff_prepare_plan, dynamic_flat_
39 ↪fielding
40 from stp_core.preprocess.ring_correction import ring_correction
41 from stp_core.preprocess.extract_flatdark import extract_flatdark, _medianize
42
43 from h5py import File as getHDF5
44 from stp_core.utils.caching import cache2plan, plan2cache
45 import stp_core.io.tdf as tdf
46
47 def main(argv):
48     """To do...
49
50     """

```

```

51 # Get the zero-order index of the sinogram to pre-process:
52 idx = int(argv[0])
53
54 # Get paths:
55 infile = argv[1]
56 outfile = argv[2]
57
58 # Normalization parameters:
59 norm_sx = int(argv[3])
60 norm_dx = int(argv[4])
61
62 # Params for flat fielding with post flats/darks:
63 flat_end = True if argv[5] == "True" else False
64 half_half = True if argv[6] == "True" else False
65 half_half_line = int(argv[7])
66
67 # Params for extended FOV:
68 ext_fov = True if argv[8] == "True" else False
69 ext_fov_rot_right = argv[9]
70 if ext_fov_rot_right == "True":
71     ext_fov_rot_right = True
72     if (ext_fov):
73         norm_sx = 0
74 else:
75     ext_fov_rot_right = False
76     if (ext_fov):
77         norm_dx = 0
78 ext_fov_overlap = int(argv[10])
79
80 # Method and parameters coded into a string:
81 ringrem = argv[11]
82
83 # Flat fielding method (conventional or dynamic):
84 dynamic_ff = True if argv[12] == "True" else False
85
86 # Tmp path and log file:
87 tmppath = argv[13]
88 if not tmppath.endswith(sep): tmppath += sep
89 logfilename = argv[14]
90
91
92 # Open the HDF5 file:
93 f_in = getHDF5(infile, 'r')
94
95 try:
96     if "/tomo" in f_in:
97         dset = f_in['tomo']
98     else:
99         dset = f_in['exchange/data']
100
101 except:
102     log = open(logfilename, "a")
103     log.write(linsep + "\tError reading input dataset. Process will end.")
104     log.close()
105     exit()
106
107 num_proj = tdf.get_nr_projs(dset)

```

```

108 num_sinos = tdf.get_nr_sinos(dset)
109
110 # Check if the HDF5 makes sense:
111 if (num_sinos == 0):
112     log = open(logfilename, "a")
113     log.write(linesep + "\tNo projections found. Process will end.")
114     log.close()
115     exit()
116
117 # Get flat and darks from cache or from file:
118 skipflat = False
119 skipdark = False
120 if not dynamic_ff:
121     try:
122         corrplan = cache2plan(infile, tmppath)
123     except Exception as e:
124         #print "Error(s) when reading from cache"
125         corrplan = extract_flatdark(f_in, flat_end, logfilename)
126         if (isscalar(corrplan['im_flat']) and isscalar(corrplan['im_flat_after'])):
↪ ):
127             skipflat = True
128         else:
129             plan2cache(corrplan, infile, tmppath)
130     else:
131         # Dynamic flat fielding:
132         if "/tomo" in f_in:
133             if "/flat" in f_in:
134                 flat_dset = f_in['flat']
135                 if "/dark" in f_in:
136                     im_dark = _medianize(f_in['dark'])
137                 else:
138                     skipdark = True
139             else:
140                 skipflat = True # Nothing to do in this case
141         else:
142             if "/exchange/data_white" in f_in:
143                 flat_dset = f_in['/exchange/data_white']
144                 if "/exchange/data_dark" in f_in:
145                     im_dark = _medianize(f_in['/exchange/data_dark'])
146                 else:
147                     skipdark = True
148             else:
149                 skipflat = True # Nothing to do in this case
150
151         # Prepare plan for dynamic flat fielding with 16 repetitions:
152         if not skipflat:
153             EFF, filtEFF = dff_prepare_plan(flat_dset, 16, im_dark)
154
155         # Read input image:
156         im = tdf.read_sino(dset, idx).astype(float32)
157         f_in.close()
158
159         # Perform pre-processing (flat fielding, extended FOV, ring removal):
160         if not skipflat:
161             if dynamic_ff:
162                 # Dynamic flat fielding with downsampling = 2:
163                 im = dynamic_flat_fielding(im, idx, EFF, filtEFF, 2, im_dark, norm_sx,
↪ norm_dx)

```

```

164         else:
165             im = flat_fielding(im, idx, corrplan, flat_end, half_half, half_half_line,
↳ norm_sx, norm_dx)
166
167             im = extfov_correction(im, ext_fov, ext_fov_rot_right, ext_fov_overlap)
168             if not skipflat and not dynamic_ff:
169                 im = ring_correction(im, ringrem, flat_end, corrplan['skip_flat_after'],
↳ half_half, half_half_line, ext_fov)
170             else:
171                 im = ring_correction(im, ringrem, False, False, half_half, half_half_line,
↳ ext_fov)
172
173             # Write down reconstructed preview file (file name modified with metadata):
174             im = im.astype(float32)
175             outfile = outfile + '_' + str(im.shape[1]) + 'x' + str(im.shape[0]) + '_' + str(
↳ nanmin(im)) + '$' + str(nanmax(im))
176             im.tofile(outfile)
177
178             # 255 C:\Temp\Pippo.tdf C:\Temp\Pippo 0 0 True True 900 False False 0 rivers:3;0
↳ False C:\Temp C:\Temp\log_00.txt
179
180
181 if __name__ == "__main__":
182     main(argv[1:])

```

## preview\_postprocessing

This section contains the preview\_postprocessing script.

Download file: [preview\\_postprocessing.py](#)

```

1 #####
2 # (C) 2016 Elettra - Sincrotrone Trieste S.C.p.A.. All rights reserved. #
3 # # #
4 # # #
5 # This file is part of STP-Core, the Python core of SYRMEP Tomo Project, #
6 # a software tool for the reconstruction of experimental CT datasets. #
7 # # #
8 # STP-Core is free software: you can redistribute it and/or modify it #
9 # under the terms of the GNU General Public License as published by the #
10 # Free Software Foundation, either version 3 of the License, or (at your #
11 # option) any later version. #
12 # # #
13 # STP-Core is distributed in the hope that it will be useful, but WITHOUT #
14 # ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or #
15 # FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License #
16 # for more details. #
17 # # #
18 # You should have received a copy of the GNU General Public License #
19 # along with STP-Core. If not, see <http://www.gnu.org/licenses/>. #
20 # # #
21 #####
22
23 #
24 # Author: Francesco Brun
25 # Last modified: July, 8th 2016

```

```

26 #
27
28 from sys import argv, exit
29 from glob import glob
30 from os import linesep
31 from os.path import sep, basename, exists
32 from time import time
33 from numpy import float32, nanmin, nanmax
34 from multiprocessing import Process, Lock
35
36 # pystp-specific:
37 from stp_core.postprocess.postprocess import postprocess
38
39 from tiffiffile import imread, imsave
40
41
42
43 def main(argv):
44     """To do...
45
46     Usage
47     -----
48
49     Parameters
50     -----
51
52     Example
53     -----
54
55     The following line processes the first ten TIFF files of input path
56     "/home/in" and saves the processed files to "/home/out" with the
57     application of the Boin and Haibel filter with smoothing via a Butterworth
58     filter of order 4 and cutoff frequency 0.01:
59
60     reconstruct 0 4 C:\Temp\Dullin_Aug_2012\sino_noflat C:\Temp\Dullin_Aug_2012\sino_
61     ↪noflat\output
62     9.0 10.0 0.0 0.0 0.0 true sino slice C:\Temp\Dullin_Aug_2012\sino_noflat\tomo_
63     ↪conv flat dark
64
65     """
66     lock = Lock()
67     # Get the from and to number of files to process:
68     idx = int(argv[0])
69
70     # Get input and output paths:
71     inpath = argv[1]
72     outfile = argv[2]
73
74     if not inpath.endswith(sep): inpath += sep
75
76     # Get parameters:
77     convert_opt = argv[3]
78     crop_opt = argv[4]
79     crop_opt = '0:0:0:0'
80
81     outprefix = argv[5]
82     logfilename = argv[6]

```

```

82 # Get the files in infile:
83 files = sorted(glob(inpath + '*.tif*'))
84 num_files = len(files)
85
86 if ((idx >= num_files) or (idx == -1)):
87     idx = num_files - 1
88
89 # Read the image:
90 im = imread(files[idx])
91
92 # Process the image:
93 im = postprocess(im, convert_opt, crop_opt)
94
95 # Write down reconstructed preview file (file name modified with metadata):
96 im = im.astype(float32)
97 outfile = outfile + '_' + str(im.shape[1]) + 'x' + str(im.shape[0]) + '_' + str(
↳ nanmin(im)) + '$' + str( nanmax(im) )
98 im.tofile(outfile)
99
100
101 if __name__ == "__main__":
102     main(argv[1:])

```

## preview\_reconstruct

This section contains the `preview_reconstruct` script.

Download file: `preview_reconstruct.py`

```

1 #####
2 # (C) 2016 Elettra - Sincrotrone Trieste S.C.p.A.. All rights reserved. #
3 # # #
4 # # #
5 # This file is part of STP-Core, the Python core of SYRMEP Tomo Project, #
6 # a software tool for the reconstruction of experimental CT datasets. #
7 # # #
8 # STP-Core is free software: you can redistribute it and/or modify it #
9 # under the terms of the GNU General Public License as published by the #
10 # Free Software Foundation, either version 3 of the License, or (at your #
11 # option) any later version. #
12 # # #
13 # STP-Core is distributed in the hope that it will be useful, but WITHOUT #
14 # ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or #
15 # FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License #
16 # for more details. #
17 # # #
18 # You should have received a copy of the GNU General Public License #
19 # along with STP-Core. If not, see <http://www.gnu.org/licenses/>. #
20 # # #
21 #####
22
23 #
24 # Author: Francesco Brun
25 # Last modified: Sept, 28th 2016
26 #
27
28 # python:

```

```

29 from sys import argv, exit
30 from os import remove, sep, linesep, listdir
31 from os.path import exists, dirname, basename, splitext
32 from numpy import array, finfo, copy, float32, double, amin, amax, tile, concatenate,
↳asarray, isscalar, pi
33 from numpy import empty, reshape, log as nplog, arange, squeeze, fromfile, ndarray,
↳where, meshgrid, roll
34 from time import time
35 from multiprocessing import Process, Array
36
37 # pystp-specific:
38 from stp_core.preprocess.extfov_correction import extfov_correction
39 from stp_core.preprocess.flat_fielding import flat_fielding
40 from stp_core.preprocess.dynamic_flatfielding import dff_prepare_plan, dynamic_flat_
↳fielding
41 from stp_core.preprocess.ring_correction import ring_correction
42 from stp_core.preprocess.extract_flatdark import extract_flatdark, _medianize
43
44 from stp_core.phaseretrieval.tiehom import tiehom, tiehom_plan
45 from stp_core.phaseretrieval.phrt import phrt, phrt_plan
46
47 from stp_core.reconstruct.rec_astra import recon_astra_fbp, recon_astra_iterative
48 from stp_core.reconstruct.rec_fista_tv import recon_fista_tv
49 from stp_core.reconstruct.rec_mr_fbp import recon_mr_fbp
50 from stp_core.reconstruct.rec_gridrec import recon_gridrec
51
52 from stp_core.postprocess.postprocess import postprocess
53
54 from stp_core.utils.padding import upperPowerOfTwo, padImage, padSmoothWidth
55 from stp_core.utils.caching import cache2plan, plan2cache
56
57 from tiff file import imread, imsave
58 from h5py import File as getHDF5
59 import stp_core.io.tdf as tdf
60
61
62 def reconstruct(im, angles, offset, logtransform, reparable, circle, scale, pad, method,
63               zerone_mode, dset_min, dset_max, corr_offset, rolling, roll_shift):
64     """Reconstruct a sinogram with FBP algorithm (from ASTRA toolbox).
65
66     Parameters
67     -----
68     im1 : array_like
69         Sinogram image data as numpy array.
70     center : float
71         Offset of the center of rotation to use for the tomographic
72         reconstruction with respect to the half of sinogram width
73         (default=0, i.e. half width).
74     logtransform : boolean
75         Apply logarithmic transformation before reconstruction (default=True).
76     filter : string
77         Filter to apply before the application of the reconstruction algorithm.
↳Filter
78         types are: ram-lak, shepp-logan, cosine, hamming, hann, tukey, lanczos,
↳triangular,
79         gaussian, barlett-hann, blackman, nuttall, blackman-harris, blackman-nuttall,
80         flat-top, kaiser, parzen.
81     circle : boolean

```

```

82         Create a circle in the reconstructed image and set to zero pixels outside the
83         circle (default=False).
84
85         """
86         offset = int(round(offset))
87
88         # Upscale projections (if required):
89         if (abs(scale - 1.0) > finfo(float32).eps):
90             siz_orig1 = im.shape[1]
91             im = imresize(im, (im.shape[0], int(round(scale * im.shape[1]))), interp=
↳'bicubic', mode='F')
92             offset = int(offset * scale)
93
94         # Apply transformation for changes in the center of rotation:
95         if (offset != 0):
96             if (offset >= 0):
97                 im = im[:, :-offset]
98
99                 tmp = im[:, 0] # Get first column
100                tmp = tile(tmp, (offset, 1)) # Replicate the first column the right number
↳of times
101                im = concatenate((tmp.T, im), axis=1) # Concatenate tmp before the image
102
103            else:
104                im = im[:, abs(offset):]
105
106                tmp = im[:, im.shape[1] - 1] # Get last column
107                tmp = tile(tmp, (abs(offset), 1)) # Replicate the last column the right
↳number of times
108                im = concatenate((im, tmp.T), axis=1) # Concatenate tmp after the image
109
110        # Sinogram rolling (if required). It doesn't make sense in limited angle
↳tomography, so check if 180 or 360:
111        if ((rolling == True) and (roll_shift > 0)):
112            if ( (angles - pi) < finfo(float32).eps ):
113                # Flip the last rows:
114                im[-roll_shift:, :] = im[-roll_shift:, ::-1]
115                # Now roll the sinogram:
116                im = roll(im, roll_shift, axis=0)
117            elif ((angles - pi*2.0) < finfo(float32).eps):
118                # Only roll the sinogram:
119                im = roll(im, roll_shift, axis=0)
120
121        # Scale image to [0,1] range (if required):
122        if (zerone_mode):
123
124            #print dset_min
125            #print dset_max
126            #print numpy.amin(im_f[:])
127            #print numpy.amax(im_f[:])
128            #im_f = (im_f - dset_min) / (dset_max - dset_min)
129
130            # Cheating the whole process:
131            im = (im - numpy.amin(im[:])) / (numpy.amax(im[:]) - numpy.amin(im[:]))
132
133        # Apply log transform:
134        if (logtransform == True):
135            im[im <= finfo(float32).eps] = finfo(float32).eps

```



```

136         im = -nplog(im + corr_offset)
137
138     # Replicate pad image to double the width:
139     if (pad):
140
141         dim_o = im.shape[1]
142         n_pad = im.shape[1] + im.shape[1] / 2
143         marg = (n_pad - dim_o) / 2
144
145         # Pad image:
146         im = padSmoothWidth(im, n_pad)
147
148     # Perform the actual reconstruction:
149     if (method.startswith('FBP')):
150         im = recon_astra_fbp(im, angles, method, recpar)
151     elif (method == 'MR-FBP_CUDA'):
152         im = recon_mr_fbp(im, angles)
153     elif (method == 'FISTA-TV_CUDA'):
154         im = recon_fista_tv(im, angles, recpar, recpar)
155     elif (method == 'GRIDREC'):
156         [im, im] = recon_gridrec(im, im, angles, recpar)
157     else:
158         im = recon_astra_iterative(im, angles, method, recpar, zerone_mode)
159
160
161     # Crop:
162     if (pad):
163         im = im[marg:dim_o + marg, marg:dim_o + marg]
164
165     # Resize (if necessary):
166     if (abs(scale - 1.0) > finfo(float32).eps):
167         im = imresize(im, (siz_orig1, siz_orig1), interp='nearest', mode='F')
168
169     # Return output:
170     return im.astype(float32)
171
172
173 #def _testwritedownsino(tmp_im):
174
175 #     for ct in range(0, tmp_im.shape[0]):
176 #         a = tmp_im[ct, :, :].squeeze()
177 #         fname = 'C:\\Temp\\StupidFolder\\sino_' + str(ct).zfill(4) + '.tif'
178 #         imsave(fname, a.astype(float32))
179
180 #def _testwritedownproj(tmp_im):
181
182 #     for ct in range(0, tmp_im.shape[1]):
183 #         a = tmp_im[:, ct, :].squeeze()
184 #         fname = 'C:\\Temp\\StupidFolder\\proj_' + str(ct).zfill(4) + '.tif'
185 #         imsave(fname, a.astype(float32))
186
187 def process(sino_idx, num_sinos, infile, outfile, preprocessing_required, corr_plan,
188 ↪ skipflat, norm_sx, norm_dx, flat_end, half_half,
189 ↪ half_half_line, ext_fov, ext_fov_rot_right, ext_fov_overlap, ringrem,
190 ↪ phaseretrieval_required, phrtmethod, phrt_param1,
191 ↪ phrt_param2, energy, distance, pixsize, phrtpad, approx_win, angles,
192 ↪ angles_projfrom, angles_projto,
193 ↪ offset, logtransform, recpar, circle, scale, pad, method, rolling, roll_
194 ↪ shift,

```

```

191     zerone_mode, dset_min, dset_max, decim_factor, downsc_factor, corr_offset,
↳ postprocess_required, convert_opt,
192     crop_opt, dynamic_ff, EFF, filtEFF, im_dark, nr_threads, logfilename):
193     """To do...
194
195     """
196     # Perform reconstruction (on-the-fly preprocessing and phase retrieval, if
197     # required):
198     if (phaseretrieval_required):
199
200         # In this case a bunch of sinograms is loaded into memory:
201
202         #
203         # Load the temporary data structure reading the input TDF file.
204         # To know the right dimension the first sinogram is pre-processed.
205         #
206
207         # Open the TDF file and get the dataset:
208         f_in = getHDF5(infile, 'r')
209         if "/tomo" in f_in:
210             dset = f_in['tomo']
211         else:
212             dset = f_in['exchange/data']
213
214         # Downscaling and decimation factors considered when determining the
215         # approximation window:
216         xrange = arange(sino_idx - approx_win * downsc_factor / 2, sino_idx + approx_
↳ win * downsc_factor / 2, downsc_factor)
217         xrange = xrange[(xrange >= 0)]
218         xrange = xrange[(xrange < num_sinos)]
219         approx_win = xrange.shape[0]
220
221         # Approximation window cannot be odd:
222         if (approx_win % 2 == 1):
223             approx_win = approx_win - 1
224             xrange = xrange[0:approx_win]
225
226         # Read one sinogram to get the proper dimensions:
227         test_im = tdf.read_sino(dset, xrange[0]).astype(float32)
228
229         # Apply projection removal (if required):
230         test_im = test_im[angles_projfrom:angles_projto, :]
231
232         # Apply decimation and downscaling (if required):
233         test_im = test_im[:,::decim_factor, ::downsc_factor]
234
235         # Perform the pre-processing of the first sinogram to get the right
236         # dimension:
237         if (preprocessing_required):
238             if not skipflat:
239                 if dynamic_ff:
240                     # Dynamic flat fielding with downsampling = 2:
241                     test_im = dynamic_flat_fielding(test_im, xrange[0] / downsc_
↳ factor, EFF, filtEFF, 2, im_dark, norm_sx, norm_dx)
242                 else:
243                     test_im = flat_fielding(test_im, xrange[0] / downsc_factor, corr_
↳ plan, flat_end, half_half,
244                                             half_half_line / decim_factor, norm_sx,
↳ norm_dx).astype(float32)

```

```

245     test_im = extfov_correction(test_im, ext_fov, ext_fov_rot_right, ext_fov_
↳overlap / downsc_factor).astype(float32)
246     if not skipflat and not dynamic_ff:
247         test_im = ring_correction(test_im, ringrem, flat_end, corr_plan['skip_
↳flat_after'], half_half,
248                                     half_half_line / decim_factor, ext_fov).
↳astype(float32)
249     else:
250         test_im = ring_correction(test_im, ringrem, False, False, half_half,
251                                     half_half_line / decim_factor, ext_fov).
↳astype(float32)
252
253     # Now we can allocate memory for the bunch of slices:
254     tmp_im = empty((approx_win, test_im.shape[0], test_im.shape[1]),
↳dtype=float32)
255     tmp_im[0,:,:] = test_im
256
257     # Reading all the the sinos from TDF file and close:
258     for ct in range(1, approx_win):
259
260         # Read the sinogram:
261         test_im = tdf.read_sino(dset, zrange[ct]).astype(float32)
262
263         # Apply projection removal (if required):
264         test_im = test_im[angles_projfrom:angles_projto, :]
265
266         # Apply decimation and downscaling (if required):
267         test_im = test_im[:,::decim_factor, ::downsc_factor]
268
269         # Perform the pre-processing for each sinogram of the bunch:
270         if (preprocessing_required):
271             if not skipflat:
272                 if dynamic_ff:
273                     # Dynamic flat fielding with downsampling = 2:
274                     test_im = dynamic_flat_fielding(test_im, zrange[ct] / downsc_
↳factor, EFF, filtEFF, 2, im_dark, norm_sx, norm_dx)
275                 else:
276                     test_im = flat_fielding(test_im, zrange[ct] / downsc_factor,
↳corr_plan, flat_end, half_half,
277                                             half_half_line / decim_factor, norm_sx,
↳norm_dx).astype(float32)
278                     test_im = extfov_correction(test_im, ext_fov, ext_fov_rot_right, ext_
↳fov_overlap / downsc_factor).astype(float32)
279                     if not skipflat and not dynamic_ff:
280                         test_im = ring_correction(test_im, ringrem, flat_end, corr_plan[
↳'skip_flat_after'], half_half,
281                                                     half_half_line / decim_factor, ext_fov).
↳astype(float32)
282                     else:
283                         test_im = ring_correction(test_im, ringrem, False, False, half_
↳half,
284                                                     half_half_line / decim_factor, ext_fov).
↳astype(float32)
285
286         tmp_im[ct,:,:] = test_im
287
288         f_in.close()
289

```

```

290     # Now everything has to refer to a downscaled dataset:
291     sino_idx = ((zrange == sino_idx).nonzero())
292
293     #
294     # Perform phase retrieval:
295     #
296
297     # Prepare the plan:
298     if (phrtmethod == 0):
299         # Paganin's:
300         phrtplan = tiehom_plan(tmp_im[:,0,:], phrt_param1, phrt_param2, energy,
↪distance, pixsize * downsc_factor, phrtpad)
301     else:
302         phrtplan = phrt_plan(tmp_im[:,0,:], energy, distance, pixsize * downsc_
↪factor, phrt_param2, phrt_param1, phrtmethod, phrtpad)
303         #phrtplan = prepare_plan (tmp_im[:,0,:], beta, delta, energy, distance,
304         #pixsize*downsc_factor, padding=phrtpad)
305
306     # Process each projection (whose height depends on the size of the bunch):
307     for ct in range(0, tmp_im.shape[1]):
308         #tmp_im[:,ct,:] = phase_retrieval(tmp_im[:,ct,:], phrtplan).
↪astype(float32)
309         if (phrtmethod == 0):
310             tmp_im[:,ct,:] = tiehom(tmp_im[:,ct,:], phrtplan).astype(float32)
311         else:
312             tmp_im[:,ct,:] = phrt(tmp_im[:,ct,:], phrtplan, phrtmethod).
↪astype(float32)
313
314     # Extract the requested sinogram:
315     im = tmp_im[sino_idx[0],:,:].squeeze()
316
317     else:
318
319     # Read only one sinogram:
320     f_in = getHDF5(infile, 'r')
321     if "/tomo" in f_in:
322         dset = f_in['tomo']
323     else:
324         dset = f_in['exchange/data']
325     im = tdf.read_sino(dset,sino_idx).astype(float32)
326     f_in.close()
327
328     # Apply projection removal (if required):
329     im = im[angles_projfrom:angles_projto, :]
330
331     # Apply decimation and downscaling (if required):
332     im = im[:,::decim_factor,::downsc_factor]
333     sino_idx = sino_idx / downsc_factor
334
335     # Perform the preprocessing of the sinogram (if required):
336     if (preprocessing_required):
337         if not skipflat:
338             if dynamic_ff:
339                 # Dynamic flat fielding with downsampling = 2:
340                 im = dynamic_flat_fielding(im, sino_idx, EFF, filtEFF, 2, im_dark,
↪norm_sx, norm_dx)
341             else:
342                 im = flat_fielding(im, sino_idx, corr_plan, flat_end, half_half,
↪half_half_line / decim_factor,

```

```

343         norm_sx, norm_dx).astype(float32)
344     im = extfov_correction(im, ext_fov, ext_fov_rot_right, ext_fov_overlap)
345     if not skipflat and not dynamic_ff:
346         im = ring_correction(im, ringrem, flat_end, corr_plan['skip_flat_after
↪'], half_half,
347                             half_half_line / decim_factor, ext_fov)
348     else:
349         im = ring_correction(im, ringrem, False, False, half_half,
350                             half_half_line / decim_factor, ext_fov)
351
352
353     # Additional ring removal before reconstruction:
354     #im = boinhaibel(im, '11;')
355     #im = munchetal(im, '5;1.8')
356     #im = rivers(im, '13;')
357     #im = raven(im, '11;0.8')
358     #im = oimoen(im, '51;51')
359
360     # Actual reconstruction:
361     im = reconstruct(im, angles, offset / downsc_factor, logtransform, repara, circle,
↪ scale, pad, method,
362                    zerone_mode, dset_min, dset_max, corr_offset, rolling, roll_
↪shift).astype(float32)
363
364     # Apply post-processing (if required):
365     if postprocess_required:
366         im = postprocess(im, convert_opt, crop_opt)
367     else:
368         # Create the circle mask for fancy output:
369         if (circle == True):
370             siz = im.shape[1]
371             if siz % 2:
372                 rang = arange(-siz / 2 + 1, siz / 2 + 1)
373             else:
374                 rang = arange(-siz / 2, siz / 2)
375             x,y = meshgrid(rang,rang)
376             z = x ** 2 + y ** 2
377             a = (z < (siz / 2 - int(round(abs(offset) / downsc_factor))) ** 2)
378             im = im * a
379
380     # Write down reconstructed preview file (file name modified with metadata):
381     im = im.astype(float32)
382     outfile = outfile + '_' + str(im.shape[1]) + 'x' + str(im.shape[0]) + '_' +
↪str(amin(im)) + '$' + str(amax(im))
383     im.tofile(outfile)
384
385     #print "With %d thread(s): [%0.3f sec, %0.3f sec, %0.3f sec]." % (nr_threads,
386     #t1-t0, t2-t1, t3-t2)
387
388 def main(argv):
389     """To do...
390
391     Usage
392     -----
393
394     Parameters
395     -----
396

```

```
397
398 Example
399 -----
400
401
402 """
403 # Get the from and to number of files to process:
404 sino_idx = int(argv[0])
405
406 # Get paths:
407 infile = argv[1]
408 outfile = argv[2]
409
410 # Essential reconstruction parameters:
411 angles = float(argv[3])
412 offset = float(argv[4])
413 recpar = argv[5]
414 scale = int(float(argv[6]))
415
416 overpad = True if argv[7] == "True" else False
417 logtrsf = True if argv[8] == "True" else False
418 circle = True if argv[9] == "True" else False
419
420 # Parameters for on-the-fly pre-processing:
421 preprocessing_required = True if argv[10] == "True" else False
422 flat_end = True if argv[11] == "True" else False
423 half_half = True if argv[12] == "True" else False
424
425 half_half_line = int(argv[13])
426
427 ext_fov = True if argv[14] == "True" else False
428
429 norm_sx = int(argv[17])
430 norm_dx = int(argv[18])
431
432 ext_fov_rot_right = argv[15]
433 if ext_fov_rot_right == "True":
434     ext_fov_rot_right = True
435     if (ext_fov):
436         norm_sx = 0
437 else:
438     ext_fov_rot_right = False
439     if (ext_fov):
440         norm_dx = 0
441
442 ext_fov_overlap = int(argv[16])
443
444 skip_ringrem = True if argv[19] == "True" else False
445 ringrem = argv[20]
446
447 # Extra reconstruction parameters:
448 zerone_mode = True if argv[21] == "True" else False
449 corr_offset = float(argv[22])
450
451 reconmethod = argv[23]
452
453 decim_factor = int(argv[24])
454 downsc_factor = int(argv[25])
```

```

455
456 # Parameters for postprocessing:
457 postprocess_required = True if argv[26] == "True" else False
458 convert_opt = argv[27]
459 crop_opt = argv[28]
460
461 # Parameters for on-the-fly phase retrieval:
462 phaseretrieval_required = True if argv[29] == "True" else False
463 phrtmethod = int(argv[30])
464 phrt_param1 = double(argv[31]) # param1( e.g. regParam, or beta)
465 phrt_param2 = double(argv[32]) # param2( e.g. thresh or delta)
466 energy = double(argv[33])
467 distance = double(argv[34])
468 pixsize = double(argv[35]) / 1000.0 # pixsize from micron to mm:
469 phrtpad = True if argv[36] == "True" else False
470 approx_win = int(argv[37])
471
472 angles_projfrom = int(argv[38])
473 angles_projto = int(argv[39])
474
475 rolling = True if argv[40] == "True" else False
476 roll_shift = int(argv[41])
477
478 preprocessingplan_fromcache = True if argv[42] == "True" else False
479 dynamic_ff = True if argv[43] == "True" else False
480
481 nr_threads = int(argv[44])
482 tmppath = argv[45]
483 if not tmppath.endswith(sep): tmppath += sep
484
485 logfilename = argv[46]
486
487 # Open the HDF5 file:
488 f_in = getHDF5(infile, 'r')
489 if "/tomo" in f_in:
490     dset = f_in['tomo']
491 else:
492     dset = f_in['exchange/data']
493     if "/provenance/detector_output" in f_in:
494         prov_dset = f_in['provenance/detector_output']
495
496 dset_min = -1
497 dset_max = -1
498 if (zerone_mode):
499     if ('min' in dset.attrs):
500         dset_min = float(dset.attrs['min'])
501     else:
502         zerone_mode = False
503
504     if ('max' in dset.attrs):
505         dset_max = float(dset.attrs['max'])
506     else:
507         zerone_mode = False
508
509 num_sinos = tdf.get_nr_sinos(dset) # Pay attention to the downscale factor
510
511 if (num_sinos == 0):
512     exit()

```

```

513
514 # Check extrema:
515 if (sino_idx >= num_sinos):
516     sino_idx = num_sinos - 1
517
518 # Get correction plan and phase retrieval plan (if required):
519 skipflat = False
520
521 corrplan = 0
522 im_dark = 0
523 EFF = 0
524 filtEFF = 0
525 if (preprocessing_required):
526     if not dynamic_ff:
527         # Load flat fielding plan either from cache (if required) or from TDF file
528         # and cache it for faster re-use:
529         if (preprocessingplan_fromcache):
530             try:
531                 corrplan = cache2plan(infile, tmpopath)
532             except Exception as e:
533                 #print "Error(s) when reading from cache"
534                 corrplan = extract_flatdark(f_in, flat_end, logfilename)
535                 if (isscalar(corrplan['im_flat']) and isscalar(corrplan['im_flat_
↪after']))):
536                     skipflat = True
537                 else:
538                     plan2cache(corrplan, infile, tmpopath)
539             else:
540                 corrplan = extract_flatdark(f_in, flat_end, logfilename)
541                 if (isscalar(corrplan['im_flat']) and isscalar(corrplan['im_flat_after
↪']))):
542                     skipflat = True
543                 else:
544                     plan2cache(corrplan, infile, tmpopath)
545
546 # Downscale flat and dark images if necessary:
547 if isinstance(corrplan['im_flat'], ndarray):
548     corrplan['im_flat'] = corrplan['im_flat'][:,::downsc_factor,::downsc_
↪factor]
549 if isinstance(corrplan['im_dark'], ndarray):
550     corrplan['im_dark'] = corrplan['im_dark'][:,::downsc_factor,::downsc_
↪factor]
551 if isinstance(corrplan['im_flat_after'], ndarray):
552     corrplan['im_flat_after'] = corrplan['im_flat_after'][:,::downsc_factor,
↪::downsc_factor]
553 if isinstance(corrplan['im_dark_after'], ndarray):
554     corrplan['im_dark_after'] = corrplan['im_dark_after'][:,::downsc_factor,
↪::downsc_factor]
555
556 else:
557     # Dynamic flat fielding:
558     if "/tomo" in f_in:
559         if "/flat" in f_in:
560             flat_dset = f_in['flat']
561         if "/dark" in f_in:
562             im_dark = _medianize(f_in['dark'])
563         else:
564             skipdark = True

```



```

565         else:
566             skipflat = True # Nothing to do in this case
567     else:
568         if "/exchange/data_white" in f_in:
569             flat_dset = f_in['/exchange/data_white']
570             if "/exchange/data_dark" in f_in:
571                 im_dark = _medianize(f_in['/exchange/data_dark'])
572             else:
573                 skipdark = True
574         else:
575             skipflat = True # Nothing to do in this case
576
577     # Prepare plan for dynamic flat fielding with 16 repetitions:
578     if not skipflat:
579         EFF, filtEFF = dff_prepare_plan(flat_dset, 16, im_dark)
580
581         # Downscale images if necessary:
582         im_dark = im_dark[:, :downsc_factor, :downsc_factor]
583         EFF = EFF[:, :downsc_factor, :downsc_factor, :]
584         filtEFF = filtEFF[:, :downsc_factor, :downsc_factor, :]
585
586     f_in.close()
587
588     # Run computation:
589     process(sino_idx, num_sinos, infile, outfile, preprocessing_required, corrplan,
590 ↪ skipflat, norm_sx,
591 ↪ norm_dx, flat_end, half_half, half_half_line, ext_fov, ext_fov_rot_
592 ↪ right, ext_fov_overlap, ringrem,
593 ↪ phaseretrieval_required, phrtmethod, phrt_param1, phrt_param2, energy,
594 ↪ distance, pixsize, phrtpad, approx_win, angles,
595 ↪ angles_projfrom, angles_projto, offset,
596 ↪ logtrsf, recpar, circle, scale, overpad, reconmethod,
597 ↪ rolling, roll_shift,
598 ↪ zerone_mode, dset_min, dset_max, decim_factor,
599 ↪ downsc_factor, corr_offset, postprocess_required, convert_opt, crop_
600 ↪ opt, dynamic_ff, EFF, filtEFF, im_dark, nr_threads, logfilename)
601
602     # Sample:
603     # 311 C:\Temp\BrunGeorgos.tdf C:\Temp\BrunGeorgos.raw 3.1416 -31.0 shepp-logan
604     # 1.0 False False True True True True 5 False False 100 0 0 False rivers:11;0
605     # False 0.0 FBP_CUDA 1 1 False - - True 5 1.0 1000.0 22 150 2.2 True 16 0 1799
606     # True True 2 C:\Temp\StupidFolder C:\Temp\log_00.txt
607
608 if __name__ == "__main__":
609     main(argv[1:])

```

## preview\_phaseretrieval

This section contains the `preview_phaseretrieval` script.

Download file: `preview_phaseretrieval.py`

```

1 #####
2 # (C) 2016 Elettra - Sincrotrone Trieste S.C.p.A.. All rights reserved. #
3 # #
4 # #

```

```

5 # This file is part of STP-Core, the Python core of SYRMEP Tomo Project, #
6 # a software tool for the reconstruction of experimental CT datasets. #
7 # #
8 # STP-Core is free software: you can redistribute it and/or modify it #
9 # under the terms of the GNU General Public License as published by the #
10 # Free Software Foundation, either version 3 of the License, or (at your #
11 # option) any later version. #
12 # #
13 # STP-Core is distributed in the hope that it will be useful, but WITHOUT #
14 # ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or #
15 # FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License #
16 # for more details. #
17 # #
18 # You should have received a copy of the GNU General Public License #
19 # along with STP-Core. If not, see <http://www.gnu.org/licenses/>. #
20 # #
21 #####
22 #
23 #
24 # Author: Francesco Brun
25 # Last modified: July, 8th 2016
26 #
27
28 from sys import argv, exit
29 from os import remove, sep, linesep
30 from os.path import exists
31 from numpy import float32, double, nanmin, nanmax, finfo, ndarray
32 from time import time
33 from multiprocessing import Process, Lock
34 from pyfftw.interfaces.cache import enable as pyfftw_cache_enable, disable as pyfftw_
35 ↪cache_disable
36 from pyfftw.interfaces.cache import set_keepalive_time as pyfftw_set_keepalive_time
37
38 # pystp-specific:
39 from stp_core.phaseretrieval.tiehom import tiehom, tiehom_plan
40 from stp_core.phaseretrieval.phrt import phrt, phrt_plan
41
42 from h5py import File as getHDF5
43 from stp_core.utils.caching import cache2plan, plan2cache
44 from stp_core.preprocess.extract_flatdark import extract_flatdark
45 import stp_core.io.tdf as tdf
46
47 def main(argv):
48     """To do...
49
50     """
51     lock = Lock()
52
53     skip_flat = True
54     first_done = False
55     pyfftw_cache_disable()
56     pyfftw_cache_enable()
57     pyfftw_set_keepalive_time(1800)
58
59     # Get the from and to number of files to process:
60     idx = int(argv[0])
61

```

```

62 # Get full paths of input TDF and output TDF:
63 infile = argv[1]
64 outfile = argv[2]
65
66 # Get the phase retrieval parameters:
67 method = int(argv[3])
68 param1 = double(argv[4]) # param1( e.g. regParam, or beta)
69 param2 = double(argv[5]) # param2( e.g. thresh or delta)
70 energy = double(argv[6])
71 distance = double(argv[7])
72 pixsize = double(argv[8]) / 1000.0 # pixsize from micron to mm:
73 pad = True if argv[9] == "True" else False
74
75 # Tmp path and log file:
76 tmppath = argv[10]
77 if not tmppath.endswith(sep): tmppath += sep
78 logfilename = argv[11]
79
80
81 # Open the HDF5 file:
82 f_in = getHDF5(infile, 'r')
83 if "/tomo" in f_in:
84     dset = f_in['tomo']
85 else:
86     dset = f_in['exchange/data']
87 num_proj = tdf.get_nr_projs(dset)
88 num_sinos = tdf.get_nr_sinos(dset)
89
90 # Check if the HDF5 makes sense:
91 if (num_proj == 0):
92     log = open(logfilename, "a")
93     log.write(linesep + "\tNo projections found. Process will end.")
94     log.close()
95     exit()
96
97
98 # Get flats and darks from cache or from file:
99 try:
100     corrplan = cache2plan(infile, tmppath)
101 except Exception as e:
102     #print "Error(s) when reading from cache"
103     corrplan = extract_flatdark(f_in, True, logfilename)
104     remove(logfilename)
105     plan2cache(corrplan, infile, tmppath)
106
107 # Read projection:
108 im = tdf.read_tomo(dset, idx).astype(float32)
109 f_in.close()
110
111 # Apply simple flat fielding (if applicable):
112 if (isinstance(corrplan['im_flat_after'], ndarray) and isinstance(corrplan['im_
113 ↪flat'], ndarray) and
114     isinstance(corrplan['im_dark'], ndarray) and isinstance(corrplan['im_dark_
115 ↪after'], ndarray)) :
116     if (idx < num_proj/2):
117         im = (im - corrplan['im_dark']) / ((abs(corrplan['im_flat'] -
118 ↪corrplan['im_dark']) + finfo(float32).eps)
119     else:

```

```

117         im = (im - corrplan['im_dark_after']) /
→ (abs(corrplan['im_flat_after'] - corrplan['im_dark_after'])
118         + finfo(float32).eps)
119
120     # Prepare plan:
121     im = im.astype(float32)
122     if (method == 0):
123         # Paganin's:
124         plan = tiehom_plan (im, param1, param2, energy, distance, pixsize, pad)
125         im = tiehom(im, plan).astype(float32)
126     else:
127         plan = phrt_plan (im, energy, distance, pixsize, param2, param1,
→ method, pad)
128         im = phrt(im, plan, method).astype(float32)
129
130     # Write down reconstructed preview file (file name modified with
→ metadata):
131     im = im.astype(float32)
132     outfile = outfile + '_' + str(im.shape[1]) + 'x' + str(im.shape[0]) + '_' +
→ str( nanmin(im) ) + '$' + str( nanmax(im) )
133     im.tofile(outfile)
134
135 if __name__ == "__main__":
136     main(argv[1:])

```

## Credits

## Citations

We kindly request that you cite the following article [\[A1\]](#) if you use project.

## References

---

## Bibliography

---

- [A1] Francesco Brun, Serena Pacile, Agostino Accardo, George Kourousias, Diego Dreossi, Lucia Mancini, Giuliana Tromba, and Roberto Pugliese. Enhanced and flexible software tools for x-ray computed tomography at the italian synchrotron radiation facility elettra. *Fundamenta Informaticae*, 141(2-3):233–243, Oct 2015. URL: <http://doi.org/10.3233/FI-2015-1273>, doi:10.3233/FI-2015-1273.
- [B1] Francesco Brun, Serena Pacile, Agostino Accardo, George Kourousias, Diego Dreossi, Lucia Mancini, Giuliana Tromba, and Roberto Pugliese. Enhanced and flexible software tools for x-ray computed tomography at the italian synchrotron radiation facility elettra. *Fundamenta Informaticae*, 141(2-3):233–243, Oct 2015. URL: <http://doi.org/10.3233/FI-2015-1273>, doi:10.3233/FI-2015-1273.



## S

stp\_core, 128  
stp\_core.io.tdf, 8  
stp\_core.phaseretrieval.phrt, 10  
stp\_core.phaseretrieval.tiehom, 9  
stp\_core.postprocess.postprocess, 11  
stp\_core.preprocess.extfov\_correction,  
11  
stp\_core.preprocess.extract\_flatdark,  
12  
stp\_core.preprocess.flat\_fielding, 12  
stp\_core.preprocess.ring\_correction, 13  
stp\_core.reconstruct.rec\_astra, 13  
stp\_core.reconstruct.rec\_fista\_tv, 14  
stp\_core.reconstruct.rec\_gridrec, 14  
stp\_core.reconstruct.rec\_mr\_fbp, 15  
stp\_core.utils.caching, 15  
stp\_core.utils.findcenter, 16  
stp\_core.utils.padding, 16





- C**  
 cache2plan() (in module stp\_core.utils.caching), 15
- E**  
 extfov\_correction() (in module stp\_core.preprocess.extfov\_correction), 12  
 extract\_flatdark() (in module stp\_core.preprocess.extract\_flatdark), 12
- F**  
 flat\_fielding() (in module stp\_core.preprocess.flat\_fielding), 12
- G**  
 get\_det\_size() (in module stp\_core.io.tdf), 8  
 get\_dset\_chunks() (in module stp\_core.io.tdf), 8  
 get\_dset\_shape() (in module stp\_core.io.tdf), 8  
 get\_nr\_projs() (in module stp\_core.io.tdf), 8  
 get\_nr\_sinos() (in module stp\_core.io.tdf), 8
- P**  
 padImage() (in module stp\_core.utils.padding), 16  
 padSmoothWidth() (in module stp\_core.utils.padding), 16  
 parse\_metadata() (in module stp\_core.io.tdf), 8  
 phrt() (in module stp\_core.phaseretrieval.phrt), 10  
 phrt\_plan() (in module stp\_core.phaseretrieval.phrt), 10  
 plan2cache() (in module stp\_core.utils.caching), 15  
 postprocess() (in module stp\_core.postprocess.postprocess), 11
- R**  
 read\_sino() (in module stp\_core.io.tdf), 9  
 read\_tomo() (in module stp\_core.io.tdf), 9  
 recon\_astra\_fbp() (in module stp\_core.reconstruct.rec\_astra), 14  
 recon\_astra\_iterative() (in module stp\_core.reconstruct.rec\_astra), 14  
 recon\_fista\_tv() (in module stp\_core.reconstruct.rec\_fista\_tv), 14  
 recon\_gridrec() (in module stp\_core.reconstruct.rec\_gridrec), 15  
 recon\_mr\_fbp() (in module stp\_core.reconstruct.rec\_mr\_fbp), 15  
 replicatePadImage() (in module stp\_core.utils.padding), 17  
 ring\_correction() (in module stp\_core.preprocess.ring\_correction), 13
- S**  
 stp\_core (module), 17, 128  
 stp\_core.io.tdf (module), 8  
 stp\_core.phaseretrieval.phrt (module), 10  
 stp\_core.phaseretrieval.tiehom (module), 9  
 stp\_core.postprocess.postprocess (module), 11  
 stp\_core.preprocess.extfov\_correction (module), 11  
 stp\_core.preprocess.extract\_flatdark (module), 12  
 stp\_core.preprocess.flat\_fielding (module), 12  
 stp\_core.preprocess.ring\_correction (module), 13  
 stp\_core.reconstruct.rec\_astra (module), 13  
 stp\_core.reconstruct.rec\_fista\_tv (module), 14  
 stp\_core.reconstruct.rec\_gridrec (module), 14  
 stp\_core.reconstruct.rec\_mr\_fbp (module), 15  
 stp\_core.utils.caching (module), 15  
 stp\_core.utils.findcenter (module), 16  
 stp\_core.utils.padding (module), 16
- T**  
 tiehom() (in module stp\_core.phaseretrieval.tiehom), 9  
 tiehom\_plan() (in module stp\_core.phaseretrieval.tiehom), 10
- U**  
 upperPowerOfTwo() (in module stp\_core.utils.padding), 17  
 usecorrelation() (in module stp\_core.utils.findcenter), 16
- W**  
 write\_sino() (in module stp\_core.io.tdf), 9

`write_tomo()` (in module `stp_core.io.tdf`), 9

## Z

`zeroPadImage()` (in module `stp_core.utils.padding`), 17