TomPost Documentation

Release 0.4

Kenneth Higa

August 04, 2017

CONTENTS

Contents:

CHAPTER

ONE

INTRODUCTION: WHAT DOES TOMPOST DO?

TomPost is Python package for postprocessing of tomography data. It was developed with a particular workflow in mind, which involved postprocessing of a stack of images containing a reconstructed cylindrical volume in which multiple (approximately) flat objects were stacked on top of one another. The fully-automated workflow (included in the data/extract.py and data/avizo.py directories of the installed package) extracts voxel columns passing through the cylinder from top to bottom, detects the flat objects in these columns, rotates and crops them so that their boundaries coincide with slices in the image stack, analyzes intensity distributions in the extracted regions to obtain segmentation threshold values, and finally produces Avizo scripts for processing the results into surface meshes with Avizo 9. Some users might find have a need for only parts of this workflow, so the processing routines were packaged to encourage flexible reuse.

TomPost was developed to process directories full of images representing different x-y slices through tomographic reconstructions, as produced by TomoPy and other tomography reconstruction programs.

CHAPTER

TWO

INSTALLATION

Windows, Mac OS X, GNU/Linux, and many other operating systems provide command-line interfaces. Installation and use of TomPost, as described in this tutorial, will require limited use of your computer's command-line interface (CLI). For Microsoft Windows, this is the Command Prompt. For GNU/Linux, Mac OS X, and similar systems this might be a Terminal or xterm.

2.1 Introduction to command-line interfaces

CLIs allow users to interact with computer systems by typing commands. The system prompts the user for a command, the user responds by typing a command and pressing the "Enter" key, and the computer attempts to accomplish the given task. This may involve doing calculations and displaying results on the screen or writing results into a file. If the computer is unsuccessful, which can occur for a variety of reasons, it typically displays an error message. Whether successful or unsuccessful, once the attempt is complete, the computer returns to the start of the cycle and prompts the user for another command.

In the CLIs provided with many operating systems, simply giving the name of a program as a command causes the system to "run" the named program by starting the program and giving it control of the interface.

As an example, running the Windows 7 "Command Prompt" program causes a window containing a CLI to appear. The system shows a command prompt that typically looks like:

```
C:\Users\UserName>
```

The Python program is typically named `python`, although this will vary among installations. If Python is already installed on your system, entering this in the Windows CLI starts Python in interactive mode:

```
C:\Users\UserName>python
Python 2.7.6 (default, Nov 10 2013, 19:24:18) [MSC v.1500 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Python prompts the user for commands as well, within the same interface. The >>> is Python's command prompt, which is completely distinct from those typed at the Windows command prompt. Commands entered here are instructions to Python, rather than to the Windows Command Prompt. For example, the following commands ask Python to load "libraries" that enhance its capabilities:

```
>>> import scipy
>>> import h5py
```

If these commands do not result in error messages, the required and suggested libraries are already installed on your computer, and you can skip the next section.

To exit from Python and return to the Windows command prompt, one may type `exit()` at the Python prompt:

```
>>> exit()
C:\Users\UserName>
```

The `exit()` command tells Python that its task is complete. When the operating system sees that a task has completed, it once again offers its command prompt to the user.

2.2 Python and required libraries

TomPost is written in Python 2, and is not yet compatible with Python 3. It is known to run on Python 2.6.6, 2.7.3, and 2.7.9. Typing `python -V` (replacing `python` with the name of you Python 2 program, if needed) at the CLI will display the version number of the installed version of Python, if Python is already installed.

For users of various versions of Microsoft Windows and Mac OS X, there are Python distributions such as Python(x,y), Anaconda, and Enthought Canopy that contain Python along with a number of common libraries. Some of these distributions are quite large.

TomPost has been tested on Python 2.7.3 as packaged in Python(x,y) 2.7.3.1 on a computer running Microsoft Windows 7. The Python(x,y) website contains installation details.

However, users of GNU/Linux may prefer to install only the necessary packages through a package manager. For example, on the system on which TomPost was initially developed (running a 64-bit PC installation of Debian GNU/Linux 6.0), the required and suggested Python packages can be installed by running from the command line (with sufficient permissions):

```
apt-get install python-scipy
```

On a CentOS 6 system, these packages can be installed by running from the command line (with sufficient permissions):

```
yum install scipy
```

For users who wish to download and install the necessary packages individually, software may be downloaded from the Python and SciPy websites. NumPy is included with SciPy.

2.3 TomPost installation

The TomPost distribution comes in two compressed formats: `tompost-1.0.0.zip` for Windows, and `tompost-1.0.0.tar.gz` for most other systems.

Unpack the downloaded file into a directory (also known as a "folder"). This will create a subdirectory named `tompost-1.0.0`. Make note of the location of this directory. On a Windows computer, this might be `C:\Users\UserName\Downloads\tompost-1.0.0`

Open your computer's CLI. Interactions with the CLI are always understood to be taking place "within" a present directory location, and it is most convenient to interact with files located in the present directory. Commands can be given to the CLI to change the present directory location. On Windows, Mac OS X, and GNU/Linux systems, the `cd` (Change Directory) command can be used to move to another directory. Under Windows, the appropriate command will be similar to:

```
cd C:\Users\UserName\Downloads\tompost-1.0.0
```

while on Mac OS X or GNU/Linux, the appropriate command will be similar to:

```
cd /home/UserName/Downloads/tompost-1.0.0
```

Once in the directory with the installation files, you may use a command such as `dir` (on Windows) or `ls` on Linux or Mac OS X to see a list of files in the directory.

The `setup.py` is used to tell Python about how TomPost should be installed. If you wish to perform a system-wide installation, and you possess sufficient administrative privileges to do so, type the following at the command line to install:

```
python setup.py install
```

Alternatively, you can install TomPost within your own user directories, for which no special administrative access is needed, by running:

```
python setup.py install --user
```

To check that TomPost has been properly installed, start Python as before (by typing `python` at the command line interface), and at the Python prompt, type:

```
import tompost
```

If no errors were encountered, TomPost is ready to use (if ImageJ/Fiji and sed are available). The directory into which the installation files were unpacked is no longer needed and may be removed.

2.4 ImageJ/Fiji installation

ImageJ or Fiji (an ImageJ distribution) must be installed as well. Installation procedures are described on their respective websites. TomPost users must find the full name of the ImageJ executable file (possibly something in the form /home/UserName/Fiji.app/ImageJ-linux64) and supply this to TomPost through function call parameters.

2.5 Avizo and Cygwin installation

Avizo 9 (FEI) must be installed to make use of the mesh generation scripts produced by the pipeline. It may be installed on a separate computer, as the "avizo.py" file can be configured to use different local directory structures.

The Avizo script generation capability of TomPost makes use of sed, the stream editor. This should already be available on Mac OS X and Linux systems. On Windows computers, one can install Cygwin to obtain a Linux-like environment (sed is automatically installed) from which one can use TomPost and easily access sed. It might be necessary to edit data/avizo.py in order for TomPost to find the sed executable.

CHAPTER

THREE

TUTORIAL

TomPost was developed with a particular workflow in mind, but with the awareness that others might want to use a slightly different workflow. It is not object-oriented in design and does not maintain information internally, but rather produces a number of external files. Some of the reliance on external files is needed because of the difficulty of passing information between Python and ImageJ. Future versions might move toward an object-oriented structure or might make use of the ImageJ Python scripting capability.

This tutorial walks through the original workflow to show how the TomPost package is used.

3.1 Extraction

The "extract.py" file in the "data" subdirectory of the TomPost installation describes the process of taking the raw tomography reconstructions, extracting voxel columns, and transforming them into many small voxel blocks that are appropriately rotated and trimmed to remove extraneous "material."

At the top of the file is the import statement needed to get access to the TomPost package:

```
import tompost
```

The "extract.py" file combines two roles: it finds all files to be processed, and directs the processing to be done. This combination of responsibilities is reflected in the rest of the lines at the top of the file, in which there are several variables containing directory names that should be modified according to the local directory structure, file names that can be changed to accomodate user preferences, and finally some information that is specific to the images collected, and which should also be modified according to user needs.

The script then loops over all specified directories of images to be processed. Each directory is understood to contain reconstructed x-y slices from a single scan, and these are expected to show a stack of electrode samples separated by polyimide spacers. The intensities within the electrode regions are relatively high, and those outside (current collectors and spacers) are relatively low. For each of these directories, the script calls "tompost.extract_block_circle" to extract voxel blocks extending through the entire image stack, and located on a circle about the axis of the reconstructed cylinder:

```
tompost.extract_block_circle(imagej_executable_path,first_input_filename,image_side_length,
sampling_radius,sample_side_length,starting_slice,ending_slice,output_filename_root)

Input
-----
imagej_executable_path: string containing full path to ImageJ executable file
first_input_filename: name of first image file in directory, files are assumed to have
sequence numbers understandable by ImageJ
image_side_length: number of pixels along one side of image (assumed to be square)
sampling_radius: radius of circle in image on which samples are extracted
sample_side_length: number of pixels along one side of extracted samples (square cross-
section)
```

```
starting_slice: index of first slice in region of interest, corresponding to position in
directory listing rather than sequence number in filename
ending_slice: index of last slice in region of interest
output_filename_root: full (with directory) root of output filenames, appropriate
extensions will be added automatically

Output
------
Number of samples extracted
```

This routine opens an ImageJ window to show the outlines of the extracted regions, for the user to look through.

Again for each of the image directories, the script then calls the "tompost.determine_separator_slices" function, which identifies slices that comfortably separate slices containing one electrode sample from the next, which works by computing average intensities over slices and using clustering to detect bright (electrode) and dark (surrounding) slices:

```
tompost.determine_separator_slices(image_filename_root,minimum_thickness,
intensity_centroid_guesses,max_sample_count)

Input
-----
image_filename_root: full (with directory) root of extracted voxel column filename
minimum_thickness: threshold in slice count below which regions darker or brighter than
surrouding regions will be ignored
intensity_centroid_guesses: sequence of two intensity values that approximately give the
anticipated average values for the dark and light regions
max_sample_count: maximum number of bright regions that will be detected as potential
electrode regions, known from sample construction

Output
------
List of slice numbers comfortably separating electrode regions
```

The separating slices are then stored in a "pickle" file for ease of later use/reference through Python.

At this point in the script, the user may specify whether the current collector appears at the top or bottom of each individual electrode region. This is easily done by inspecting resliced regions in ImageJ. What this actually specifies is whether the top or bottom boundary of the electrode has an interface with the current collector, which is usually flatter than at the other end and so should be preferentially used to alignment with the x-y plane.

Next, the script again loops over all image directories, recalling the separating slices stored in the pickle file, and calls "tompost.rotate_block" to detect the orientation of each isolated electrode region, rotate it into alignment with the x-y plane, and to then clip edges appropriately to ensure that voxels falling outside of the electrode region are removed:

```
tompost.rotate_block(imagej_executable,image_filename_root,label,global_bounds,sample_counts,
sample_edge_lengths,use_top_surface,minimum_thickness,intensity_centroid_guesses,user_adjustment)

Input
-----
imagej_executable: string containing full path to ImageJ executable file
image_filename_root: full (with directory) root of extracted voxel column filename
label: user-specified identifier for voxel block
global_bounds: a sequence; if the first is smaller that the second, these indicate that
only the slices within this range will be retained for processing
sample_counts: sequence of number of narrow probe columns to take along x and y axes
sample_edge_lengths: sequence of x and y pixel width of probe column cross-sections
use_top_surface: True or False, indicating whether the top or bottom surface of the
electrode region should be used to determine rotations
```

```
minimum_thickness: ignore all regions brighter or darker then their surroundings that are
smaller than this thickness
intensity_centroid_guesses: approximate mean intensity values for dark and light regions
user_adjustment: number of extra slices to include on both ends of electrode block, an
option provided to correct for aggressive cropping

Output
------
Root of filename of rotated and cropped image
```

3.2 Meshing

Once the extracted electrode voxel regions are available, the "avizo.py" script, also in the "data" subdirectory of the TomPost installation, may be used to produce Avizo 9 project files for processing the extracted regions into surface meshes. Again, the tompost package is imported by Python at the top of the file, and then a number of variables specifying local configuration may be modified by the user.

The script loops over the image directories (which should match those in the extraction step), and computes intensity histograms by calling "tompost.histogram":

```
tompost.histogram(raw_image_stack_filename_root,bin_count,scalar_size=4,numpy_dkind='f')

Input
-----
raw_image_stack_filename_root: full (with directory) root of electrode image file
bin_count: number of bins in which to sort intensities
scalar_size: number of bytes representing each voxel in raw image data file
numpy_dkind: string indicating data type representing voxel intensity, as used by NumPy

Output
------
A tuple of two sequences, the first giving the counts in each bin, the second giving the
edges of all bins
```

Next, the histogram information is used by "tompost.pdf_and_cdf_from_histogram" to construct probability distribution functions and cumulative distribution functions:

```
tompost.pdf_and_cdf_from_histogram(bin_edges,bin_counts)

Input
-----
bin_edges: edges of each bin, from tompost.histogram output
bin_counts: counts in each bin, from tompost.histogram output

Output
------
Tuple of discretized probability distribution and cumulative distribution functions.
```

The probability distribution functions are then put through a simple test to ensure that they are effectively unimodal, which is a sometimes very conservative indication that they are unlikely to contain defects such as large cracks:

```
unimodal(bin_edges,pdf,floor_cutoff,smoothing)

Input
-----
bin_edges: histogram bin edges generated from "tompost.histogram"
```

```
pdf: probability distribution function generated from "tompost.pdf_and_cdf_from_histogram"
floor_cutoff: ignore tails beyond the points at which the PDF reaches this value
smoothing: parameter passed to SciPy spline fitting methods to determine how much smoothing
is applied to the data

Output
------
True or False, indicating whether the distribution is effectively unimodal
```

Given relative volumes of different phases, the cumulative distribution functions are then used to calculate intensity thresholds with "tompost.inverse_cdf":

```
inverse_cdf(bin_edges,cdf,integrals)

Input
-----
bin_edges: edges of histogram bins, reported in output from tompost.histogram
cdf: cumulative distribution function values, reported in output from
tompost.pdf_and_cdf_from_histogram
integrals: sequence of cumulative distribution function values for which corresponding
sample values are sought

Output
------
Sequence of sample values corresponding to provided integral values
```

These thresholds are then scaled to make the most of a 16-bit integer range witih "tompost.scale_voxel_values":

```
tompost.scale_voxel_values(lower_bound,upper_bound,values,new_lower_bound,new_upper_bound)

Input
-----
lower_bound: original lower bound of values
upper_bound: original upper bound of values
values: values to be transformed
new_lower_bound: final lower bound
new_upper_bound: final upper bound

Output
------
Sequence of multiplicative factor used in transformation, transformed values
```

This information is finally used to make substitutions in the "avizo_form.tcl" file in the "data" directory of the TomPost installation, producing Avizo project files for each extracted voxel block. These can be used individually as needed when running Avizo, or can be processed in batch form through shell scripting.

CHAPTER

FOUR

CITING TOMPOST

Development and packaging of TomPost required signficant time and effort, and it is hoped that TomPost will save significant time and effort for other researchers. At the present time, if you use TomPost for academic work, please cite the following scientific publication, which describes the work for which the predecessor to TomPost was originally developed:

Kenneth Higa=, Shao-Ling Wu=, Dilworth Y. Parkinson, Yanbao Fu, Steven Ferreira, Vincent Battaglia, and Venkat Srinivasan, *J. Electrochem. Soc.* 2017, 164, 11, E3473-E3488

CHAPTER

FIVE

BACKGROUND AND ACKNOWLEDGEMENTS

This work was supported by the Assistant Secretary for Energy Efficiency and Renewable Energy, Office of Vehicle Technologies of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231 under the CAEBAT3 program.

TomPost was developed in the course of CAEBAT3-funded research by Kenneth F. Higa within the Energy Storage and Distributed Resources Division in the Energy Technologies Area of Berkeley Lab (Lawrence Berkeley National Laboratory, LBNL), a U.S. Department of Energy laboratory operated by the University of California. "Bringing Science Solutions to the World" is Berkeley Lab's mission.

KFH thanks his Computer Aided Battery Simulation (CABS) teammates at Sandia National Laboratories for laying out the initial particle separation process, as well as his CABS teammates at Oak Ridge National Laboratory for fabricating the physical samples for which the tomography data used to test TomPost was generated. KFH thanks his teammates in the Srinivasan group at Argonne National Laboratory (ANL) and the staff of Beamline 2 at the Advanced Photon Source, also at ANL, for helping to collect this tomography data, along with the staff of Beamline 8.3.2 at the Advanced Light Source, also at LBNL, for helping to collect tomography data with which earlier versions of the processing pipeline were tested.

KFH also thanks the developers of TomoPy, which was used to produce test reconstructions, the developers of ImageJ / Fiji, which is used by TomPost for image manipulation, the developers of Python programming language and its standard libraries, the developers of the NumPy and SciPy libraries (and the underlying libraries), the developers of the Sphinx Python Documentation Generator, as well as developers throughout the Open Source community.

While great effort has been made to eliminate bugs, they undoubtedly exist. The TomPost source code is open for all to examine. Users are encouraged to send bug reports and fixes to KFH <KHiga AT LBL DOT gov> for inclusion in the library.

In the spirit of scientific openness and in the interest of strengthening public confidence in the work of the scientific community, KFH respectfully encourages researchers receiving public funding to make their source code publicly available when publishing scientific results. KFH thanks Greg Wilson of Software Carpentry for promoting openness in software development within the scientific community and providing the inspiration to release this work as open-source software.

CHAPTER

SIX

COPYRIGHT NOTICE

CHAPTER

SEVEN

LICENSE