ETSimulations User Manual

Kyung Min Shin

Ver 0.2.3 - May 2020

Contents

1 Getting started

The ETSimulations package is designed to be a tool to facilitate generation and processing of simulated cryo-electron tomography data. As of this preliminary version, only the generation of simulated tilt stacks is well-supported and covered by this manual.

External dependencies

The following external software is used by ETSimulations and should thus be installed first:

- Chimera - https://www.cgl.ucsf.edu/chimera/
- TEM-simulator - http://tem-simulator.sourceforge.net/
- Python 3.3 or greater

Setting up ETSimulations

To begin, navigate to a directory in which you want to install the package and run the following commands:

git clone https://github.com/kmshin1397/ETSimulations.git

cd ETSimulations

python3 -m venv env

source env/bin/activate

pip install -r requirements.txt

This will have downloaded the ETSimulations code and set up the necessary Python packages and environment.

Running a simulation

Most all general parameter set up is done through a YAML file which is passed in as an argument to the main **ets_generate_data.py** program. An example such YAML file is provided in the ETSimulations directory as **configs.yaml**.

Detailed set-up with regards to the characteristics of the particles simulated are controlled through custom **"Assembler"** Python classes which can define a series of Chimera commands to open, manipulate, and combine one or more source maps, i.e. various proteins, into a fake particle source. (This is done for the T4SS simulations in the src/assemblers/t4ss_assembler.py file) Custom arguments to pass

along to your custom particle Assemblers can also be defined in the configuration YAML file, as shown in the example.

To actually run a set of simulations, run (assuming you've activated the virtual environment as indicated above - the **source ...** command):

python src/ets_generate_data.py -i configs.yaml

More details on the parameters available can be found in the next section.

Processing a simulated dataset

Another program called **ets_process_data.py** is provided to facilitate processing of the raw datasets generated by the simulations. Here as well, we provide input via a configuration YAML file. This time parameters are given to modules called **"Processors"**. Each Processor module implements a function to take the inputted configurations and automatically set up tomogram reconstruction and sub-tomogram averaging. For example, the EMAN2 processor (the only one implemented so far) will automatically create an EMAN2 project and generate a Python script which will run all desired EMAN2 processing steps (i.e. e2import.py, e2tomogram.py, etc.) in order.

The YAML file provided to the ets_process_data.py run should have:

- root: The path to the folder which contains the raw_data and sim_metadata.json files created by ets_generate_data.py
- name: The project name - should be the same one used for the simulations
- processors: A list of processor objects which have a "name" and "args" arguments. The specifics to what goes into these fields can be found in more details in the section of the manual particular to that processor.

To run the processor set up, run:

python src/ets_process_data.py -i processor_configs.yaml

2 Simulation configuration parameters

- tem_simulator_executable : The file path to the TEM-simulator executable
- chimera_exec_path : The file path to your Chimera installation
- model : The path to the main particle source file (Doesn't actually matter for T4SS simulations because I'm bypassing this and putting together a bunch of different source maps)

- root: The project root directory in which to generate simulations
- config: The TEM-simulator configuration text file to apply to each simulation. An example is provided in the templates folder.
- coord: TheTEM-simulator particle coordinates text file to use as a reference for placing particles in each generated tiltseries. An example for this is also provided in the templates folder.
- num_stacks: The number of tilt stacks to generate
- name: A name for the project
- num_cores: The number of parallel cores to utilize
- apix: The pixel size to give to generated stacks, in nm
- num_chimera_windows: The number of Chimera instances to spawn to drive particle assembly. Creating more windows will clutter your display more, but can make simulations run faster. If your particle assembly does not use a lot of Chimera commands/spend a lot of time running Chimera commands, then having multiple Chimera windows may not be necessary
- defocus_values: A list of defocus values to assign to simulations. The values will be evenly assigned across the dataset, i.e. if 2 defocus values are provided half of the simulated stacks in a dataset will be given one and the other given the other.
- bead_map : The MRC map representing fake gold beads to scatter throughout the tilt stacks. An example is provided in the templates folder
- email : An email address to send completion notifications to

3 Dataset generation outputs

Running the **ets_generate_data.py** program as discussed in section 1.3 will result in a **raw_data** folder being created in the project directory specified in the configurations. In the **raw_data** folder, each tiltseries will get its own sub-directory titled {name}_{stack number}. In each sub-directory, you will find a no-noise version of the stack and a normal noisy version.

The other important output to note is the sim_metadata.json file. This is a JSON file containing metadata for each tiltseries generated, including custom metadata that can be saved from your custom Assembler. For example, the T4SS Assembler saves the random orientations and random shifts/angles away from the centered/perpendicular positions for each component of the simulated particle which were generated during the run. An easy way to interact with and retrieve this information is the Python json module which can load this json as a Python dictionary, i.e.:

```
import json
metadata = json.load(open("sim_metadata.json", "r"))
```

4 The EMAN2 Processor

The EMAN2 Processor, found in processors/eman2_processor.py, is implemented to automatically set up cryo-ET processing of generated simulated data. It works by taking a pre-written Python template script, which lists the EMAN2 programs needed to run to take a series of raw tilt stacks to a subtomogram average, and filling in the desired parameters to those programs based on configurations provided in the YAML input file. This template script can be found at templates/eman2/eman2_process.py. Any argument which can be given to the EMAN2 programs called can be given in the YAML file as part of the parameters object for the program. Note the example configurations below.

```
  processors: [
    {name: "eman2",
     args:
    {
       particle_coordinates_file: "/Users/kshin/Documents/repositories/ETSimulations
/templates/eman2/T4SS_coords_3by3.txt",
       unbinned_boxsize: 128,
       steps_to_run: ["import", "reconstruct"],
       e2import_parameters : {
        "import_tiltseries": "enable",
        "importation": "copy",
        "apix": 2.83,
        "boxsize": 64
      },
      e2tomogram_parameters : {
        "tltstep": 3,
        "tltax": -90,
        "npk": 10,
        "tltkeep": 0.9,
        "outsize": "1k",
        "niter": "2,1,1,1",
        "pkkeep": 0.9,
        "bxsz": 64,
        "pk_mindist": 0.125,
        "filterto": 0.45,
        "rmbeadthr": 10.0,
        "threads": 48,
        "clipz": 350,
        "notmp": "enable"
      }
  }]
```

Like the example above, you can include EMAN2 processing in your processing run by adding a pro-

cessor argument to the "processors" list with the name "eman2" and suitable args.

steps_to_run

In the "args" field, the "steps_to_run" lists the processing steps which will be run when you eventually run the script generated by the processor (Note that the functions will still exist in the script to run steps not included in the list, they just won't be executed unless you manually enable them back). The available steps taken by the steps_to_run are "import", "reconstruct", "estimate_ctf", "extract", "build_set", "generate_initial_model", and "3d_refinement". Note also that the order matters in this list (i.e. reconstructing before importing tiltseries will result errors).

particle_coordinates_parameters

The particle_coordinates_parameters object lists parameters to be used for particle "picking" in preparation for sub-tomogram averaging. Specifically, they tell the EMAN2 Processor what particles to record in the EMAN2 info files for each tomogram as if they were picked using the EMAN2 Boxer tool. These parameters are:

- **mode**: The options available for the particle picking mode can be "single" or "multiple". This refers to the files used to indicate the particle coordinates to transfer into the EMAN2 project. In "single" mode, a single filepath should be provided for the **coordinates_file** argument, pointing to a text file with the 3D particle coordinates to record (an example is provided in templates/eman2). These coordinates will be used for every tomogram in the dataset. In "multiple" mode, each tomogram can be given its own coordinates file, allowing things like artificially induced picking errors. The **coordinates_file** should just be a file name instead of a full path. The EMAN2 Processor will then go into each sub-directory in the dataset's raw_data folder and look for this filename to use as the coordinates for the tomogram reconstructed from that particular stack.
- **coordinates_file**: The text file containing 3D particle coordinates to be transferred to a tomogram's info JSON file. The coordinates in this file should be in pixels, based on the EMAN2 conventions (with the origin being in the center of the axis instead of the lower left, for example). An example such file is provided at templates/eman2/T4SS_coords_3by3.txt. Note: read the **mode** explanation above as the specific usage of this parameter changes based on the mode.
- **unbinned_boxsize**: This defines the boxsize to record for particles when importing the particles from the particles coordinates file into the EMAN2 tomogram info files. This should be in terms of unbinned pixels, i.e. the scale matching the original tiltseries imported.

E2 parameters objects

Each EMAN2 program is given its own *_parameters section in the "args" field, listing all the command line arguments that would be passed in if calling these programs normally. For example, arguments to e2import.py would be listed in the e2import_parameters field as shown. Arguments which are just flags instead of taking a value, such as the "–help" option available in all these programs, should be put in to the configuration section with the special value of "enable" as can be seen in the example

above. The full list of EMAN2 programs exposed by the EMAN2 Processor, and thus able to take their own *_parameters section is:

- e2import.py : e2import_parameters
- e2tomogram.py : e2tomogram_parameters
- e2spt_tomoctf.py : e2spt_tomoctf_parameters
- e2spt_extract.py : e2spt_extract_parameters
- e2spt_buildsets.py : e2spt_buildsets_parameters
- e2spt_sgd.py : e2spt_sgd_parameters
- e2spt_refine.py : e2spt_refine_parameters

Running the generated script

The generated EMAN2 processing script that is outputted by ets_process_data.py will be located in the newly created EMAN2 project directory in the processed_data folder created. This will be a normal Python script you can run, albeit requiring Python 3, using: **python3 eman2_process.py**
Note that Python 3 is only used for the proper file IO and kicking off EMAN2 programs. The EMAN2 programs themselves will be run using Python 2 as EMAN2 is still using Python 2 officially.

Another important thing to note is that the created eman2_process.py script is not meant to be a rigid program. It has been designed to be easily modifiable - all parameters originally passed in are located towards the top of the file. Thus, the script can be easily opened and edited as necessary, such as if the steps_to_process should be modified to pick up from where an error interrupted the processing. For additional clarity and potential modification, a simple text file containing the raw command-line versions of the EMAN2 commands handled by the eman2_process.py will be created as well in a file called eman2_process_commands.txt.

5 The IMOD Processor

The IMOD Processor, found in processors/imod_processor.py, is implemented to facilitate processing of the generated simulation dataset with the IMOD software. Specifically, a project directory is created and set up, along with necessary files, for running the **batchruntomo** program (`https://bio3d.colorado.edu/imod/doc/man/batchruntomo.html`) for processing the simulated stacks in batch. The IMOD Processor as well will take its inputs from the configuration YAML passed into ets_process_data.py. Consider the following example:

```
  processors: [
    {
      name: "imod",
      args: {
```

```
        start_step: 0.0,
        end_step: 8.0,
        num_fiducials: 10,
        tilt_axis: -90,
        apix: 0.283,
        fiducial_method: "autofidseed"
      }
    }
  ]
```

Parameters available

The IMOD Processor, like all others, have the **name** argument ("imod") and an **args** object filled with parameters. The parameters exposed here are abstracted versions of some of the more common ones when processing with batchruntomo, as well as some others specifying processing options for the IMOD processor itself. Specifically, we have:

- start_step: The batchruntomo processing start step (the list of steps can be found on the IMOD batchruntomo documentation page)
- end_step: The batchruntomo processing end step (the list of steps can be found on the IMOD batchruntomo documentation page)
- num_fiducials: The number of fiducials to track with either Raptor or the automatic fiducial seeding/tracking for alignment
- tilt_axis: The tiltseries tilt axis you specified in the TEM-Simulator configs
- apix: The pixel size (in nm)
- fiducial_method: Either "autofidseed" or "raptor", determining how beads are seeded and tracked for alignment
- custom_template: (Optional. Note this is not included in the example above; putting this option in overwrites the others) A filepath to a batchruntomo template file (.adoc) to be used instead of the default one generated with the other options. This can be used to control any parameters open to batchruntomo, but will probably require more experimentation/advanced knowledge to get it to work.
- force_coarse_align: (Optional) Takes a Boolean true or false value. Allows overriding of default behavior which skips the coarse alignment step of the batchruntomo process. Will likely be enabled when using the processor on real data instead of simulated ones.
- real_data_mode: (Optional) When set to true, the IMOD processor will assume you are trying to set up and run batchruntomo for a real dataset not generated by ets_generate_data.py. In real data mode, pre-oriented particle data will not be looked for and extracted from a sim_metadata.json

file. It will be assumed that you have the **force_coarse_align** enabled along with the real data mode, and the fake coarse alignment output files will not be copied over either. Finally, instead of looking for a "raw_data" folder in the provided root, the processor will just use the **root** as the raw data folder.

- data_dirs_start_with: (Optional) By default this will be set to the **name** parameter for the project, and tells the processor what to consider as data subdirectories within the provided root folder. For simulated data, the subdirectories will be in the form **name**_particleNum so the default is fine. This option is exposed for when dealing with real data where this may not be the case.

An important thing to note is that stacks generated by the simulations will most likely not have enough signal to have the IMOD course alignment with cross-correlation step work. Instead, the cross-correlation is likelier to shift tilts extremely out of alignment than to make any improvements. Thus, the IMOD Processor is designed to automatically skip the coarse alignment (steps 2.0 and 3.0) and fake having done it by moving in fake versions of its outputs like the .rawtlt file. It is possible to override this using the **force_coarse_align** option.

Additionally, as mentioned, fake versions of the rough alignment are used in place of doing the cross-correlation. These fake files are located in the templates/imod folder and can be edited if desired. For example, the rawtlt file is for a 2-degree increment, -54 to 54 degrees tilt scheme and should be edited if using a different tilt scheme with your TEM-Simulator.

Using the IMOD Processor on real data

It is possible to use the IMOD Processor to set up and run batchruntomo for real datasets. To do so, something like the processor arguments below should be used:

```
processors: [
  {
    name: "imod",
    args: {
      start_step: 0.0,
      end_step: 8.0,
      num_fiducials: 10,
      tilt_axis: -11,
      apix: 0.52,
      fiducial_method: "autofidseed",
      force_coarse_align: true,
      real_data_mode: true,
      data_dirs_start_with: "dg"
    }
  }
]
```

Detailed descriptions for each parameter can be found above. For this to work, organize your real dataset within your filesystem so that the raw stacks are already split into their own subdirectories within the **root** folder you provide in the YAML configs. Each subdirectory should begin with the string provided to the **data_dirs_start_with** option; should only contain one MRC file inside as this will be taken to be the raw stack to process. It may of course be useful to use the **custom_template** option to provide personalized .adoc files for the batchruntomo directives when dealing with real data.

A note must be made about the IMOD Processor's handling of the fiducial alignment step for batchruntomo (step 6, corresponding to the *tiltalign* IMOD program). By default batchruntomo attempts to dynamically iterate runs of tiltalign, changing parameters such as the tilt AngleOffset and thickness based on past iterations, likely to reduce the need for manual intervention. However, in our experience, the AngleOffset values introduced automatically by batchruntomo for tiltalign throw off the final computed tilt angles significantly. To handle this, we manually run tiltalign for each data sub-directory and skip step 6 of batchruntomo.