i3assist Documentation

Release 0.0.1

Dustin Reed Morado

Jan 31, 2017

CONTENTS

CHAPTER

ONE

I3ASSIST

1.1 A python library to facilitate using I3

i3assist is a python library designed to make using I3 slightly easier. It is inspired by my work using pytom and enjoying the flexibilty provided by a clean and object oriented API around averaging and classification jobs.

Contents:

1.1.1 i3assist

This page contains the i3assist API and a list of module's classes

i3assist.Euler([phi, theta, psi, unit])	Describes a particle's orientation using ZXZ extrinsic euler angles.
i3assist.RotationMatrix([angles, unit])	Describes a particle's orientation using ZXZ passive rotation matrix.
i3assist.GridSearch([theta_max, theta_step, ...])	Describes the local grid search implemented in new I3.
i3assist.Position(position_line)	A single particle transform in old I3.
i3assist.PositionList([filename, positions])	Describes a full position file as a list of Positions.
i3assist.Transform(transform_line)	A single particle transform in new I3.
i3assist.TransformList([filename, transforms])	Describes a full transform file with as a list of Transforms.

i3assist.Euler

class i3assist.**Euler**(*phi=0.0*, *theta=0.0*, *psi=0.0*, *unit='deg'*)

Describes a particle's orientation using ZXZ extrinsic euler angles.

The first rotation (phi) is about the z-axis (z). The second rotation (theta) is about the new x-axis (x'). The third rotation (psi) is about the new z-axis (z'') after the second rotation.

All rotations are of the coordinate axes and not of point coordinates.

Parameters

- **phi** (float, optional) – First rotation about the z-axis (z).
- **theta** (float, optional) – Second rotation about the new x-axis (x').
- **psi** (float, optional) – Third rotation about the new z-axis (z'').
- **unit** (str, optional) – Whether rotations are in degrees "deg" or radians "rad".

__init__(*phi=0.0*, *theta=0.0*, *psi=0.0*, *unit='deg'*)

Methods

`__init__`([phi, theta, psi, unit])	
`copy`()	Returns a copy of the euler object.
`degrees`([inplace])	Converts Euler object to describe rotations in units of degrees.
`invert`([inplace])	Returns a copy of euler object with inverted rotations.
`normalize`([inplace])	Returns a copy of euler object with rotations in old I3 bounds.
`pos_string`()	Returns string of angles in old I3 pos format.
`radians`([inplace])	Converts Euler object to describe rotations in units of radians.
`to_matrix`()	Returns the RotationMatrix object equivalent of Euler object.
`trf_string`()	Returns string of angles in new I3 trf format.

Attributes

`angles`	`list` of `float` – Array of rotations in order.
`phi`	`float` – First rotation about the z-axis (z).
`psi`	`float` – Third rotation about the new z-axis (z”).
`theta`	`float` – Second rotation about the new x-axis (x’).
`unit`	`str` – Describes the unit of rotations as degrees or radians.

i3assist.RotationMatrix

class `i3assist.`**`RotationMatrix`**(*angles=None*, *unit='deg'*)

Describes a particle’s orientation using ZXZ passive rotation matrix.

The rotation matrix is the composition of three rotations matrices with the first and third being about the Z-axis and the second about the X-axis.

All rotations are passive (alias) transformations of the coordinate axes and not of point coordinates (active / alibi).

Parameters

- **`angles`** (`list` of `float`, optional) – Euler angles describing the rotation.
- **`unit`** (`str`, optional) – Whether rotations are in degrees “deg” or radians “rad”

`__init__`(*angles=None*, *unit='deg'*)

Methods

`__init__`([angles, unit])	
`copy`()	Returns a copy of the rotation matrix.
`invert`([inplace])	Returns rotation matrix describing opposite rotation.
`pos_string`()	Returns string of angles in old I3 pos format.

Continued on next page

Table 1.4 – continued from previous page

to_euler([unit])	Converts rotation matrix to equivalent euler angles.
transpose([inplace])	Returns rotation matrix describing opposite rotation.
trf_string()	Returns string of angles in new I3 trf format.

Attributes

matrix	numpy.ndarray – (3,3) matrix describing the rotation.

i3assist.GridSearch

class i3assist.**GridSearch**(*theta_max=0.0*, *theta_step=0.0*, *psi_max=0.0*, *psi_step=0.0*, *do_180=False*)

Describes the local grid search implemented in new I3.

See the explanation in MRASRCH and the I3 subvolume tutorial for more information on how the grid search is implemented. But overall the grid is defined by four parameters: Nutation (theta) maximum and step and Spin (psi) maximum and step. Finally there is a do_180 parameter to support the old I3 eulerFG scripts, but this is not available in new I3.

Parameters

- **theta_max** (float, optional) – Maximum half-angle of a cone of nutation about the north pole of the unit sphere.
- **theta_step** (float, optional) – Angular increment of nutation.
- **psi_max** (float, optional) – Maximum absolute angle of spin about the orientation axis of the particle. Searched in both directions.
- **psi_step** (float, optional) – Angular increment of spin.
- **do_180** (boolean, optional) – If True the spins opposite of the current orientation's facing will also be searched.

__init__(*theta_max=0.0*, *theta_step=0.0*, *psi_max=0.0*, *psi_step=0.0*, *do_180=False*)

Methods

__init__([theta_max, theta_step, psi_max, ...])	

Attributes

do_180	boolean Whether to search spins opposite particle facing.
psi_max	float Max absolute angle of spin about the particle z-axis.
psi_step	float Angular increment of spin.
rotations	list of *i3assist.Euler* Rotations searched in i3.
theta_max	float Max half-angle of nutation about the north pole.

Continued on next page

Table 1.7 – continued from previous page

theta_step	float Angular increment of nutation.

i3assist.Position

class i3assist.**Position**(*position_line*)

A single particle transform in old I3.

The fields in the position are broken down as follows:

1. Particle's X coordinate in the tomogram *not used in alignment*
2. Particle's Y coordinate in the tomogram *not used in alignment*
3. Particle's Z coordinate in the tomogram *not used in alignment*
4. Particle's 4D index in the stack of particles
5. Particle's displacement (shift) along X from center of volume.
6. Particle's displacement (shift) along Y from center of volume.
7. Particle's displacement (shift) along Z from center of volume.
8. First euler angle in degrees about Z axis in range [-180, 180].
9. Second euler angle in degrees about X axis in range [0, 180].
10. Third euler angle in degrees about Z axis in range [-180, 180].
11. Class number the particle belongs to.
12. Max correlation coefficient between the particle and reference
13. Unknown reserved value.

Parameters **positionLine** (str) – A string with the transform data.

__init__(*position_line*)

Methods

__init__(position_line)	
copy()	Returns a copy of the position.
to_trf()	Returns the equivalent new I3 transform.

Attributes

class_number	int Class number that the particle belongs to.
coordinates	list of int Particles integer coordinates in tomogram.
rotation	*i3assist.Euler* Euler angles rotating reference to particle.
score	float Correlation score of particle alignment to reference.

Continued on next page

Table 1.9 – continued from previous page

shifts	list of float Particle displacements from box center.
stack_index	int Fourth dimension coordinate of particle in stack.

i3assist.PositionList

class i3assist.**PositionList**(*filename=''*, *positions=None*)

Describes a full position file as a list of Positions.

Refer to the documentation for Position objects

Parameters

- **filename** (str) – Filename of pos file.
- **positions** (list of *i3assist.Position*) – List of positions.

__init__(*filename=''*, *positions=None*)

Methods

__init__([filename, positions])	
from_file(filename)	Loads list of positions from a trf file.
to_file(filename)	Writes out a Position list to a pos file.

Attributes

filename	Filename associated with position list.
positions	List of Position objects.

i3assist.Transform

class i3assist.**Transform**(*transform_line*)

A single particle transform in new I3.

For more information refer to the subvolume tutorial document for I3.

Parameters **transformLine** (str) – A string with the transform data.

__init__(*transform_line*)

Methods

__init__(transform_line)	
add_rotation(rotation[, inplace])	Adds a rotation in addition to particles current orientation.
add_shift([shift_x, shift_y, shift_z, inplace])	Adjusts the particles defined center by an arbitrary vector.
copy()	Returns a copy of the transform.

Continued on next page

Table 1.12 – continued from previous page

scale(scale_factor[, inplace])	Scale the transform to handle binning.
to_pos()	Return the equivalent old I3 position.

Attributes

class_number	int Class number that the particle belongs to.
coordinates	list of int Particles integer coordinates in tomogram.
rotation	*i3assist.RotationMatrix* Particles rotation matrix to orient.
score	float Correlation score of particle alignment to reference.
shifts	list of float Particle displacements from coordinates.
subset	str Subset identifier for particle.

i3assist.TransformList

class i3assist.**TransformList**(*filename=''*, *transforms=[]*)

Describes a full transform file with as a list of Transforms.

For more information refer to the subvolume tutorial document for I3.

Parameters

- **filename** (str) – Filename of trf file.
- **transforms** (list of *i3assist.Transform*) – List of transforms.

__init__(*filename=''*, *transforms=[]*)

Methods

__init__([filename, transforms])	
from_file(filename)	Loads list of transforms from a trf file.
get_by_class(class_number)	Gets a subset of a transform list based on the class number.
get_by_subset(subset)	Gets a subset of a transform list based on the subset field.
scale(scale_factor[, inplace])	Scales all of the transforms in a list.
sort_by_class([inplace])	Sorts a transform list by class numbers.
sort_by_score([inplace])	Sorts a transform list by correlation coefficient.
to_file(filename)	Writes out a Transform list to a trf file.

Attributes

filename	Filename associated with transform list.
transforms	List of Transform objects.

CHAPTER

TWO

USER GUIDES FOR LEARING I3

2.1 Basic Subtomogram Averaging with I3

Author Dustin Reed Morado

2.1.1 Setting up the I3 Project Directory

The first thing to do is the set your current working directory to the folder containing all of your extracted particles and old I3 position data files () or new I3 transform data files (). Then we will create our I3 project directory, and inside of that directory we will create a folder for our maps, project definitions, tilt angles describing our particle's missing wedges, and transforms:

1. `mkdir i3 # Make our project directory`
2. `mkdir i3/defs # Make our project definitions directory`
3. `mkdir i3/maps # Make our project maps directory`
4. `mkdir i3/tlt # Make our project missing wedge directory`
5. `mkdir i3/trf # Make our project initial transforms directory`
6. ...Or simply: `mkdir -p i3/{defs,maps,tlt,trf}`

Now we change into our newly created project directory with `cd i3`. Now we need to copy a template parameter file (`mraparam.sh`) and a template protomo tilt angle file (`template.tlt`) that describes our data's missing wedge into our project directory. These are included in the example folder of this reference package:

7. `cp ~/Downloads/i3_guides/examples/mraparam.sh .  # Parameter file`
8. `cp ~/Downloads/i3_guides/examples/template.tlt .  # Missing Wedge`

Finally to finish setting the basics of our I3 project directory; edit the parameter file and the tilt angle file to suit the needs of your current project. The parameter file is well documented in explaining what each of the parameters does and while most of the given values must be changed, they provide a meaningfull starting points of the values that you probably want to use for your project.

Filling the project directories

The next step before we start running the program is to fill the maps, definitions, tilt, and transforms directories we created above. You will find it easiest to start with the maps directory and from there we can use loops in the Bash shell to quickly populate the other directories.

Maps directory

I3 is very selective when it comes to the names of the maps. Shorter names seem to give the least amount of trouble. Therefore it is useful to create symbolic links to the extracted particles in the directory above our I3 project directory with new names to keep the program happy. First, obviously change into the maps directory and the following Bash shell loop does exactly that:

```
jliu@keemun i3/maps $ i=1; for j in ../../*.mrc
do
    ln -sv ${i} p$(printf "%05d" ${i}).mrc
    i=$((i+1))
done
```

This creates symbolic links from whatever your subtomograms are named to "`p00001.mrc,p00002.mrc, ...`"

Tilt angles directory

Next, the missing wedge for each map is described using our tilt angle files. The most simple and straightforward way to do this is to using the template we copied to our project directory to describe the missing wedge for each map and particle. To do this we again create symbolic links to our template file for each map that we just created in our maps directory. Again, change into the tlt directory and the following loop will accomplish that:

```
jliu@keemun i3/tlt $ for i in ../maps/*.mrc
do
    ln -sv ../template.tlt $(basename ${i} .mrc).tlt
done
```

This creates symbolic links "`p00001.tlt,p00002.tlt, ...`" to `template.tlt`.

Transforms directory

The transforms directory can be the most challenging to fill, there are many possible situations based on your particular project:

1. Particle coordinates as a single point per subtomogram center; no orientation
2. Particle coordinates as two points per subtomogram; describes orientation
3. Old I3 transform as a pos file; describes inverse orientation
4. New I3 transform as a trf file from a previous run; describes orientation

In the first case we can create the most basic transform file for each map. Refer to the I3 tutorial PDF to understand what each field of the transform file describes. After changing into the transforms directory the following Bash shell can be used to create these files:

```
jliu@keemun i3/trf $ i=1; for j in ../../*.mrc
do
    eval $(i3stat -sh -o ${j})
    tx=$(((ox + nx) / 2))
    ty=$(((oy + ny) / 2))
    tz=$(((oz + nz) / 2))
    fmt_i=$(printf "%05d" ${i})
    echo -n "p${fmt_i} ${tx} ${ty} ${tz} " > p${fmt_i}.trf
    echo "0.0 0.0 0.0 1.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 1.0" >> p${fmt_i}.trf
```

```
    i=$((i+1))
done
```

In the second case it is easiest to use the programs that take these positions and convert them into Old I3 transform pos files and go forward from the third case. From here we can convert these to New I3 transforms using the i3assist library. To do this we go to the directory with the original maps and position files and start the IPython console using the command `ipython` from the shell and use the following commands:

```
import i3assist
import glob
import os.path
for posfile in sorted(glob.glob("*.pos")):
    trffile = os.path.splitext(posfile)[0] + '.trf'
    pl = i3assist.PositionList()
    tl = i3assist.TransformList()
    pl.from_file(posfile)
    pos = pl[0]
    pos_trf = pos.to_trf()
    tl.transforms = [pos_trf]
    tl.to_file(trffile)
```

Then we need to change directories back to the I3 project transforms directory and run the following Bash script to correctly insert the proper first four fields into the convert transform files and give these files the correct names:

```
jliu@keemun i3/trf $ i=i; for j in ../../*.mrc
do
    trffile=${j/%mrc/trf}
    fmt_i=$(printf "%05d" ${i})
    eval $(i3stat -sh -o ${j})
    tx=$(((ox + nx) / 2))
    ty=$(((oy + ny) / 2))
    tz=$(((oz + nz) / 2))
    awk -v s="p${fmt_i}" -v tx=${tx} -v ty=${ty} -v tz=${tz} '
        { $1 = s; $2 = tx; $3 = ty; $4 = tz; print }' ${trffile} > p${fmt_i}.trf
    i=$((i+1))
done
```

We can then safely delete the temporary transform files we created with i3assist.

```
jliu@keemun i3/trf $ rm ../../*.trf
```

For the last case we just need to rename the transform files to match the same naming convention as our maps. The loop to do this is simply the one we used for filling the maps directory:

```
jliu@keemun i3/trf $ i=1; for j in ../../*.trf
do
    ln -sv ${i} p$(printf "%05d" ${i}).trf
    i=$((i+1))
done
```

Definitions directory

The last step in filling in our project directories is the definitions directory, which just contains two files `maps` and `sets`. Refer to the I3 tutorial PDF to see the format of these files, but with all of the other directories already setup generating these two files is simple using the following loop:

```
jliu@keemun i3/defs $ touch maps sets; for i in ../maps/*.mrc
do
    echo "../maps $(basename ${i}) ../tlt/$(basename ${i} .mrc).tlt" >> maps
    echo "$(basename ${i}) $(basename ${i} .mrc)" >> sets
done
```

And we are done with setup, and can continue to actually processing our project, which is extremely simple.

2.1.2 Running the First Cycle

With everything in place; the first cycle of processing utilizes four shell scripts that combine and abstract the smaller building block programs of I3 into sensible processing units based on the road map of basic subtomogram averaging and classification.

1. `i3mrainitial.sh # Produces the initial global average,reference and masks`
2. `i3mramsacls.sh 0 # Runs the actual alignment and classification.`
3. `i3cp.sh 0 # Copies selected class averages to select folder for alignemnt`
4. `i3mraselect.sh 0 # Aligns selected class averages to make the final alignment`

i3mrainitial

After running i3mrainitial you will now have a directory `cycle-000` in your project folder. In this folder you will have the initial I3 database, the global average of the subtomograms based on the transforms given by the transform files in the transform directory, the masked and filtered reference that will be used in the subsequent alignment step along with the Fourier transform of this file, and finally the binary mask that will be used in the subsequent classification step. There may also be montages of the reference and versions of all maps that have been rotated about the X-axis to visualize the maps perpendicular to the Z-axis, which while useful in some cases, can also be visualized using IMOD's slicer window. Whether or not these files are created are based on the parameters you set in your parameter file.

Troubleshooting

This is the most likely command to fail in running I3, due to the fact that this is when the I3 database is first created. The error messages are also not particularly helpful but the following suggestions may help.

When restarting a run that failed delete the `cycle-000` directory make whatever corrections necessary and then rerun the command: `rm -rf cycle-000 && i3mrainitial.sh`.

- `i3external` errors are often caused by your maps having too long of a

  filename or if you have followed this guide that should not be the case, and therefore means you have more maps in project than I3 can handle which can be anywhere from 1,000 to 10,000. Try splitting your data into multiple I3 projects, or refer to the intermediate or expert guides for how to manually add maps to the database.

- `i3boximport` errors are due to the second to fourth fields of of your

  transform file being incorrect. Again if you have followed this guide your particle center coordinates should be correct. However, if you created the transform files yourself, double check that your volumes coordinates and origin correspond correctly with the centers in your transforms. You can do this using the I3 command: `i3stat -o <Your subtomogram filename>`.

- `i3dataset` errors are the most difficult to debug. They signal that the

  shifts and rotations describing a subtomograms orientation and position are invalid, duplicated elsewhere in the transform file, or incorrectly formated. Again following this guide should prevent these problems, but if you created your own transform files, make sure that each line in the transform file has 16 fields; that lines have the same transformed coordinates, and that the last nine fields are all values between 0 and 1.

i3mramsacls

After running i3mramsacls you will have many new files in the `cycle-000` directory. However, there are just a few that as a beginner you should look at before starting the next program.

The first is a montage of the calculated factors in the SVD (Singular Value Decomposition) processing of the dataset. These factors reduce the dimension of the dataset to the most variable regions of interest and this variance is used to cluster the data into classes using HAC (Hierarchical Ascendant Clustering).

The second of these are the class averages produced after the clustering. Here you are looking to make sure that the classes correspond to true variation and heterogeneity in the data, and not artifacts such as the missing wedge, simple variations in noise, and junk such as gold and debris nearby particles. You will want to especially focus on the class averages that have been selected for aligning class averages in the last step of the processing of the first cycle.

Troubleshooting

Errors in this stage of processing are uncommon. However if you have any errors, they will almost certainly come from a mistake in the parameter file. Make sure that alignment parameters are sane, and most frequently make sure that the class averages requested were actually calculated.

Rerunning this step to fix specific errors is beyond the scope of a beginner tutorial, and for more information on how to handle these situations efficiently, please refer to the intermediate and expert guides.

i3cp

After running this command the only thing done is copying the selected class averages to a new select folder to be aligned in the next step. There's nothing to check at this step, just move quickly on to the last script.

Troubleshooting

The only error in this stage is if you selected a class for which class averages were not generated. Edit your parameter file making sure that the class selected does exist.

To rerun the command, find the most recent generated directory in the `cycle-000` directory and delete it (it should have the name <...>-000-sel):

```
jliu@keemun i3 $ ls -ltr cycle-000 # Find the most recently created directory
jliu@keemun i3 $ rm -rf cycle-000/<...>-000-sel #replace <...> as appropriate
jliu@keemun i3 $ i3cp.sh 0 # rerun command
```

i3mraselect

After running this last stage of the first cycle, you will finally have an aligned average to inspect. Optionally if you specified FSC (Fourier Shell Correlation) masks in your parameter file, you will also have even and odd half averages

and the corresponding FSC data and graph in postscript format. Note that this resolution reported is not gold-standard and can easily overestimate the true resolution of your data.

You have now finished your first cycle and the next step is to create and run another cycle and we will repeat this until our structure converges by visual inspection or in terms of resolution.

Troubleshooting

Errors in this stage are also uncommon similar to i3mramsacls and if you encounter trouble here refer to the suggestions for that section.

2.1.3 Running the Second and Subsequent Cycles

With our first cycle complete, basically all of the steps are all the same. The only difference is the first step which originally was `i3mrainitial.sh` is now replaced with `i3mranext.sh 0 1` where the 0 represents our old cycle number and 1 represents the new cycle we are now calculating. Here we will not create the global average (since it would be the same as the previous cycle's aligned average), and we will not create the new cycle's database from scratch, but instead copy it from the previous cycle and appended to with new information on the current cycle. Again, we will be left with the reference for this cycle and the classification mask that will be used in the next stage.

i3mranext

Before running this command be sure to edit and update your parameter file to take into account the refinement achieved in the first cycle. Do not worry about losing the parameters used in the first cycle as a copy of the parameter file used for that cycle exists in the `cycle-000` directory.

Troubleshooting

You should also not experience any common errors at this stage. Any errors you do experience should point to errors in your parameter files, specifically the filtering, masking, and location of the reference, as well as the masks created for classification.

The rest of the cycle

Now that you have a directory `cycle-001` you can repeat the last three stages of the first cycle. Namely:

1. `jliu@keemun i3 $ i3mramsacls.sh 1`
2. `jliu@keemun i3 $ i3cp.sh 1`
3. `jliu@keemun i3 $ i3mraselect.sh 1`

2.1.4 Conclusion

Again, we repeat the above steps for as many cycles as desired. If we know beforehand that we want to run multiple cycles in succession, I3 supports an Bash shell environment variable `I3PARAM` that defaults to `mraparam.sh`, but we can set to another value to support using multiple parameter files written at once.

For example if we want to run 10 cycles without manually checking each stage and each cycle we can write a parameter file for each cycle, say

```
mraparam_00.sh, mraparam_01.sh, ... mraparam_09.sh
```

and then we run the following loop:

```
for i in {0..9}
do
   fmt_i=$(printf "%02d" ${i})
   export I3PARAM="mraparam_${fmt_i}.sh"
   if [[ ${i} -eq 0 ]]
   then
       i3mrainitial.sh
   else
       i3mranext.sh $((i-1)) ${i}
   fi
   i3mramsacls.sh ${i}
   i3cp.sh ${i}
   i3mraselect.sh ${i}
done
```

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

Symbols

E

G

P

R

T