ASTEC: adaptive segmentation and tracking of embryonic cells

the ASTEC group

July, 2019

Contents

Chapter 1

Installation

ASTEC (acronym of "adaptive segmentation and tracking of embryonic cells" [Gui15]) has been designed for unix-like systems (e.g. Linux, or MacOS). It has been developed with `python2.7` and was not tested for `python3.0`. It is a set of `python` scripts, built over a set of `C` commands.

There are two distributions. The first one can be retrieved from `github.com/astec-segmentation/astec-2019-published` and includes both the `python` and the `C` codes. The installation procedure is dedicated to this distribution.

The second one is devoluted to more advanced users that may want to benefit from future developements of the ASTEC distribution:

- `python` scripts can be retrieved from `github.com/astec-segmentation/astec`
- `C` code can be retrieved from `gitlab.inria.fr/morpheme/vt`
- optional third-party librairies can be retrieved from `gitlab.inria.fr/morpheme/vt-third-party`

Both `github.com/astec-segmentation/astec-2019-published` and `github.com/astec-segmentation/astec` contains the following 4 sub-directories

```
astec[-2019-published/]
├── documentation/
├── parameter-file-examples/
├── src/
└── tutorial/
```

- `documentation/` contains this documentation.
- `parameter-file-examples/` contains templates of parameter files for the `python` scripts. See chapter 3 for further details.
- `src/` contains the `python` scripts and files (as well as the `C` codes for the `astec-2019-published` distribution).
- `tutorial/` contains a toy data set and the associated parameter files. See chapter 2.

1.1 `github.com/astec-segmentation/astec-2019-published`

1.1.1 Linux system

This section describes the required command to install the ASTEC distribution on a Linux system (was tested on a Ubuntu system (18.04.2, 64 bits) installed on a virtual machine[1]) so the tutorial (chapter 2) can be run.

[1] `virtualbox.org`

1. Get the distribution. It is recommended (but not necessary) to use `git`, so keeping up to date with the distribution will be easier. `git` can be installed with

   ```
   $ sudo apt install git
   ```

   Then, choose the directory where to install the ASTEC distribution, and download it

   ```
   $ cd /wherever/one/wants/
   $ git clone https://github.com/astec-segmentation/astec-2019-published.git
   ```

   It creates the directory `/wherever/one/wants/astec-2019-published/` that will be denoted `/path/to/astec/` from now on.

2. Prepare the compilation of the `C` code. Compilation is done within the `cmake`[2] framework. The standard Ubuntu distribution comes with a `C` compiler but not with a `C++` one. Last a development version of the `zlib` is required. The next few lines allow to install the required components.

   ```
   $ sudo apt install cmake
   $ sudo apt install cmake-curses-gui
   $ sudo apt install g++
   $ sudo apt install zlib1g-dev
   ```

3. Compile the `C` code.

   ```
   $ cd /path/to/astec/
   $ cd src/ASTEC/CommunFunctions/cpp/vt/
   $ mkdir build
   $ cd build
   $ cmake ../
   $ make
   ```

4. Install the required `python` libraries. As mentioned, ASTEC has been developed with `python2.7`. `pip` is here used for the installation of the `python` libraries. Required libraries are `numpy` , `scipy` , `libtiff` , and `h5py` .

   ```
   $ sudo apt install python2.7
   $ sudo apt install python-pip
   $ sudo pip install numpy
   $ sudo pip install scipy
   $ sudo pip install libtiff
   $ sudo pip install h5py
   ```

5. Make the ASTEC scripts/commands available as on-line commands. It can be done in a terminal (but will be valid only for this terminal)

   ```
   $ export PATH=$PATH:/path/to/astec/src
   ```

   or by adding the above line in the right setup file (e.g. `.bashrc`, `.profile`, ...).

[2] `cmake.org`

Chapter 2

Tutorial

Before starting

It is advised to add to your `PATH` environment variable the paths to both the python and the C executable commands (the latter is important in case of non-standard installation). So, Astec commands can be launched without specifying the complete path to the command.

It can be done in a terminal (and will be valid only for this terminal)

```
$ export PATH=$PATH:/path/to/astec/src
$ export PATH=$PATH:/path/to/astec/src/ASTEC/CommunFunctions/cpp/vt/build/bin
```

or by modifying a setup file (e.g. `bashrc`, `.profile`, ...).

2.1 Tutorial data

The directory `/path/to/astec/tutorial/tuto-astec1/`, also denoted by `path/to/experiment/` or `<EXPERIMENT>`, contains the `RAWDATA/` and `parameters/` sub-directories and a `README` file

```
path/to/tuto-astec1/
├── RAWDATA/
├── README
└── parameters/
```

The `RAWDATA/` contains 21 time points (indexed from 0 to 20) of subsampled (for file size consideration) raw data from a 3D+t movie acquired by a MuViSPIM microscope.

```
RAWDATA/
├── LC/
│   ├── Stack0000/
│   │   ├── Time000000_00.mha.gz
│   │   ├── ...
│   │   └── Time000020_00.mha.gz
│   └── Stack0001/
│       ├── Time000000_00.mha.gz
│       ├── ...
│       └── Time000020_00.mha.gz
└── RC/
    └── Stack0000/
```

```
 1 PATH_EMBRYO = '.'
 2
 3 EN = '2019-Tutorial100'
 4
 5 begin = 0
 6 end = 20
 7
 8 acquisition_orientation = 'right'
 9 acquisition_mirrors = False
10 acquisition_resolution = (1., 1., 1.)
11
12 target_resolution = 1.0
```

Figure 2.1: Tutorial parameter file for the fusion step (lines are numbered).

```
    ├── Time000000_00.mha.gz
    ├── ...
    └── Time000020_00.mha.gz
└── Stack0001/
    ├── Time000000_00.mha.gz
    ├── ...
    └── Time000020_00.mha.gz
```

where `LC/` and `LC/` stand respectively for the left and the right cameras.

2.2 Fusion

We assume that we are located in the directory `/path/to/astec/tutorial/tuto-astec1/`. Running the fusion is done with

```
$ 1-fuse.py -p parameters/1-fuse-tutorial-parameters.py
```

`1-fuse-tutorial-parameters.py` being the dedicated parameter file (figure 2.1).

- The variable `PATH_EMBRYO` is the path to the directory where the directory `RAWDATA/` is located. It can be either relative (as in the above example) or global (it could have been `/path/to/astec/tutorial/tuto-astec1/`).
- The variable `EN` is the prefix after which the result fusion images will be named.
- The variables `begin` and `end` set respectively the first and the last index of the input time points to be processed.
- The variables `acquisition_orientation` and `acquisition_mirrors` are parameters describing the acquisition geometry.
- The variable `acquisition_resolution` is the voxel size (along the 3 dimensions X, Y and Z).
- The variable `target_resolution` is the desired isotropic (the same along the 3 dimensions) voxel size for the result fusion images.

After processing, a `FUSE/` directory has been created

```
path/to/tuto-astec1/
└── FUSE/
```

```
1 PATH_EMBRYO = '.'
2
3 EN = '2019-Tutorial100'
4
5 begin = 0
6 end = 20
```

Figure 2.2: Tutorial parameter file for the sequence intra-registration step.

```
├── RAWDATA/
├── README
└── parameters/
```

The FUSE/ directory contains

```
FUSE/
└── FUSE_RELEASE/
    ├── 2019-Tutorial100_fuse_t000.inr
    ├── ...
    ├── 2019-Tutorial100_fuse_t020.inr
    └── LOGS/
```

The fused images are named after <EN>_fuse<XXX>.inr (where <XXX> denotes the value of the variable XXX) and indexed from <begin> to <end> (as the input data).

The directory LOGS/ contains a copy of the parameter file (stamped with date and hour) as well as a log file (also stamped with date and hour) reporting information about the processing.

2.3 Sequence intra-registration (or drift compensation) [1]

We assume that we are located in the directory /path/to/astec/tutorial/tuto-astec1/. Running the sequence intra-registration is done with

```
$ 1.5-intraregistration.py -p parameters/1.5-intraregistration-tutorial-parameters-fuse.py
```

1.5-intraregistration-tutorial-parameters-fuse.py being the dedicated parameter file (figure 2.2).

- The variable PATH_EMBRYO is the path to the directory where the directory FUSE/ is located. It can be either relative (as in the above example) or global (it could have been /path/to/astec/tutorial/tuto-astec1/).
- The variable EN is the prefix after which the images are named.
- The variables begin and end set respectively the first and the last index of the input time points to be processed.

After processing, a INTRAREG/ directory has been created

```
path/to/tuto-astec1/
├── FUSE/
├── INTRAREG/
├── RAWDATA/
├── README
└── parameters/
```

The INTRAREG/ directory contains

```
INTRAREG/
└── INTRAREG_RELEASE/
    ├── CO-TRSFS/
    │   ├── 2019-Tutorial100_intrareg_flo000_ref001.trsf
    │   ├── ...
    │   └── 2019-Tutorial100_intrareg_flo019_ref020.trsf
    ├── FUSE/
    │   ├── 2019-Tutorial100_intrareg_fuse_t000.inr
    │   ├── ...
    │   └── 2019-Tutorial100_intrareg_fuse_t020.inr
    ├── LOGS/
    ├── MOVIES/
    │   └── 2019-Tutorial100_intrareg_fuse_t0-20_xy205.inr
    └── TRSFS_t0-20/
        ├── 2019-Tutorial100_intrareg_t000.trsf
        ├── ...
        ├── 2019-Tutorial100_intrareg_t020.trsf
        └── template_t0-20.inr
```

- The directory CO-TRSF/ contains the co-registration transformations.
- The directory FUSE/ contains the resampled fused images in the same geometry (images have the same dimensions along X, Y and Z), with drift compensation (the eventual motion of the sample under the microscope has been compensated).
- The directory MOVIES/ contains a 3D (which is a 2D+t) image, here 2019-Tutorial100_intrareg_fuse_t0-20_xy205.ir which the #205 XY-section of the resampled fused images for all the time points.
- The directory TRSFS/ contains the transformation of every fused image towards the reference one as well as the template image (an image large enough to including each fused images after resampling).

  The template image template_t0-20.inr is of size $422 \times 365 \times 410$ with a voxel size of 0.6 (the voxel size can be set by the variable intra_registration_resolution)

2.4 Segmentation of the first time point

We assume that we are located in the directory /path/to/astec/tutorial/tuto-astec1/. Segmenting the first time point is done with

```
$ 2-mars.py -p parameters/2-mars-tutorial-parameters.py
```

2-mars-tutorial-parameters.py being the dedicated parameter file (figure 2.3).

- The variable PATH_EMBRYO is the path to the directory where the directory FUSE/ is located. It can be either relative (as in the above example) or global (it could have been /path/to/astec/tutorial/tuto-astec1/).
- The variable EN is the prefix after which the images are named.
- The variable begin sets the index of the first input time point (to be processed).

After processing, a SEG/ directory has been created

```
path/to/tuto-astec1/
├── FUSE/
├── INTRAREG/
└── RAWDATA/
```

```
1 PATH_EMBRYO = '.'
2
3 EN = '2019-Tutorial100'
4
5 begin = 0
```

Figure 2.3: Tutorial parameter file for the segmentation of the first time point.

```
1 PATH_EMBRYO = '.'
2
3 EN = '2019-Tutorial100'
4
5 begin = 0
6
7 mancor_mapping_file='parameters/3-manualcorrection-tutorial.txt'
```

Figure 2.4: Tutorial parameter file for the segmentation correction of the first time point. See figure 2.5 for the `<mancor_mapping_file>` file.

```
README
SEG/
parameters/
```

The `SEG/` directory contains

```
SEG/
└── SEG_RELEASE/
    ├── 2019-Tutorial100_mars_t000.inr
    ├── LOGS/
    └── RECONSTRUCTION/
```

`2019-Tutorial100_mars_t000.inr` is the segmented first time point of the sequence.

2.5 Correction of the first time point segmentation

We assume that we are located in the directory `/path/to/astec/tutorial/tuto-astec1/`. Segmenting the first time point is done with

```
$ 3-manualcorrection.py -p parameters/3-manualcorrection-tutorial-parameters.py
```

`3-manualcorrection-tutorial-parameters.py` being the dedicated parameter file (figure 2.4).

- The variable `PATH_EMBRYO` is the path to the directory where the directory `SEG/` is located. It can be either relative (as in the above example) or global (it could have been `/path/to/astec/tutorial/tuto-astec1/`).
- The variable `EN` is the prefix after which the images are named.
- The variable `begin` set the index of the first input time point (to be processed).
- The variable `mancor_mapping_file` gives the file name containing the correction to be applied.

```
10 6
20 13
9 4
26 11
21 11
27 15
32 23
39 31
35 20
38 43
46 45
52 42
58 62
63 60
78 67
74 66
68 66
83 75
82 77
```

Figure 2.5: The segmentation correction file `3-manualcorrection-tutorial.txt` for the first time point. The first number is the line index (lines are numbered).

After processing, the `SEG/` directory contains

```
SEG/
└── SEG_RELEASE/
    ├── 2019-Tutorial100_mars_t000.inr
    ├── 2019-Tutorial100_seg_t000.inr
    ├── LOGS/
    └── RECONSTRUCTION/
```

`2019-Tutorial100_seg_t000.inr` is the corrected version of the segmentation obtained at the previous step.

2.6 Segmentation propagation

We assume that we are located in the directory `/path/to/astec/tutorial/tuto-astec1/`. Correcting the first time point is done with

```
$ 4-astec.py -p parameters/4-astec-tutorial-parameters.py
```

`4-astec-tutorial-parameters.py` being the dedicated parameter file (figure 2.6).

After processing, the `SEG/` directory contains

```
SEG/
└── SEG_RELEASE/
    ├── 2019-Tutorial100_mars_t000.inr
    ├── 2019-Tutorial100_seg_lineage.test
```

```
PATH_EMBRYO = '.'

EN = '2019-Tutorial100'

begin = 0
end = 20
delta = 1
raw_delay = 0

## General parameters for segmentation propagation

astec_sigma1 = 0.6
astec_sigma2 = 0.15
astec_h_min_min = 4
astec_h_min_max = 18

## Glace Parameters (if astec_membrane_reconstruction_method is set to 1 or 2):
## membrane_renforcement

astec_sigma_membrane = 0.9
astec_sensitivity = 0.99
astec_manual = False
astec_manual_sigma = 15
astec_hard_thresholding = False
astec_hard_threshold = 1.0

## Tensor voting framework

astec_sigma_TV = 3.6
astec_sigma_LF = 0.9
astec_sample = 0.2

## Default parameters (for classical use, default values should not be changed)

astec_RadiusOpening = 20
astec_Thau = 25
astec_MinVolume = 1000
astec_VolumeRatioBigger = 0.5
astec_VolumeRatioSmaller = 0.1
astec_MorphosnakeIterations = 10
astec_NIterations = 200
astec_DeltaVoxels = 10**3
astec_nb_proc = 10
```

Figure 2.6: Tutorial parameter file for the segmentation propagation.

```
├── 2019-Tutorial100_seg_t000.inr
├── 2019-Tutorial100_seg_t001.inr
├── ...
├── 2019-Tutorial100_seg_t020.inr
├── 4-astec-tutorial-parameters.py
├── LOGS/
└── RECONSTRUCTION/
```

while a `2019-Tutorial100_seg_lineage.pkl` file has been created in the `path/to/tuto-astec1/` directory.

```
path/to/tuto-astec1/
├── 2019-Tutorial100_seg_lineage.pkl
├── FUSE/
├── INTRAREG/
├── RAWDATA/
├── README
├── SEG/
└── parameters/
```

`2019-Tutorial100_seg_lineage.pkl` is a pickle python file containing a dictionary (in the python sense). It can be read by

```
$ python
...
>>> import cPickle as pkl
>>> f=open('2019-Tutorial100_seg_lineage.pkl')
>>> d=pkl.load(f)
>>> f.close()
>>> d.keys()
['h_mins_information', 'lin_tree', 'volumes_information',
'sigmas_information']
```

In this pickle file, cells have an unique identifier $i * 1000 + c$, which is made of both the image index i and the cell identifier c within a segmentation image (recall that, within an image, cells are numbered from 2, 1 being the background label).

2.7 Sequence intra-registration (or drift compensation) [2]

We assume that we are located in the directory `/path/to/astec/tutorial/tuto-astec1/`. Running the sequence intra-registration is done with

```
$ 1.5-intraregistration.py -p parameters/1.5-intraregistration-tutorial-parameters-seg.py
```

`1.5-intraregistration-tutorial-parameters-seg.py` being the dedicated parameter file (figure 2.7).

- The variable `PATH_EMBRYO` is the path to the directory where the directory `FUSE/` is located. It can be either relative (as in the above example) or global (it could have been `/path/to/astec/tutorial/tuto-astec1/`).
- The variable `EN` is the prefix after which the images are named.
- The variables `begin` and `end` set respectively the first and the last index of the input time points to be processed.
- the variable `EXP_INTRAREG` set the suffix of the sub-directory of the `INTRAREG/` directory to be created.
- the variable `intra_registration_template_type` set the images to be used to build the template. Here, since it is equal to `"SEGMENTATION''`, they are the segmentation images obtained at the previous step.

```
 1 PATH_EMBRYO = '.'
 2
 3 EN = '2019-Tutorial100'
 4
 5 begin = 0
 6 end = 20
 7
 8 EXP_INTRAREG = 'SEG'
 9
10 intra_registration_template_type = "SEGMENTATION"
11 intra_registration_template_threshold = 2
12 intra_registration_margin = 20
13
14 intra_registration_resample_segmentation_images = True
15 intra_registration_movie_segmentation_images = True
```

Figure 2.7: Tutorial parameter file for the sequence intra-registration step, segmentation images being used to build the template.

The variable `intra_registration_template_threshold` set a threshold to be applied to the template images to define the information to be kept: we want all the points with a value equal or greater than 2 to be contained in the template after resampling. Since cells are labeled from 2 and above, the template is designed to contain all labeled cells after resampling, so it is built as small as possible.

The variable `intra_registration_margin` allows to add margins (in the 3 dimensions) to the built template.

- The variable `intra_registration_resample_segmentation_images` indicates whether the segmentation images are to be resampled in the template geometry.
- The variable `intra_registration_movie_segmentation_images` indicates whether 2D+t movies have to be built from the resampled segmentation images.

After processing, a `INTRAREG/INTRAREG_SEG/` directory has been created and the `INTRAREG/` directory now contains

```
INTRAREG/
├── INTRAREG_RELEASE/
│   └── ...
└── INTRAREG_SEG/
    ├── CO-TRSFS/
    │   ├── 2019-Tutorial100_intrareg_flo000_ref001.trsf
    │   ├── ...
    │   └── 2019-Tutorial100_intrareg_flo019_ref020.trsf
    ├── FUSE/
    │   ├── 2019-Tutorial100_intrareg_fuse_t000.inr
    │   ├── ...
    │   └── 2019-Tutorial100_intrareg_fuse_t020.inr
    ├── LOGS/
    └── MOVIES/
        └── 2019-Tutorial100_intrareg_fuse_t0-20_xy174.inr
```

```
1 PATH_EMBRYO = '.'
2
3 EN = '2019-Tutorial100'
4
5 begin = 0
6 end = 20
7
8 EXP_INTRAREG = 'SEG'
```

Figure 2.8: Tutorial parameter file for the sequence properties from the co-registered segmentation images.

```
    2019-Tutorial100_intrareg_seg_t0-20_xy174.inr
  SEG/
    2019-Tutorial100_intrareg_seg_t000.inr
    ...
    2019-Tutorial100_intrareg_seg_t020.inr
  TRSFS_t0-20/
    2019-Tutorial100_intrareg_t000.trsf
    ...
    2019-Tutorial100_intrareg_t020.trsf
    template_t0-20.inr
```

In addition to directories already described in section 2.3, the `INTRAREG_SEG/` directory contains

- The directory `SEG/` contains the resampled segmentation images in the same geometry (images have the same dimensions along X, Y and Z), with drift compensation (the eventual motion of the sample under the microscope has been compensated).
- In addition to a 2D+t movie made from the resampled fusion images, the directory `MOVIES/` contains a 2D+t movie made from the resampled segmentation images.
- The template image `template_t0-20.inr` in the directory `TRSFS/` is now of size $323 \times 265 \times 348$ with a voxel size of 0.6, which is smaller than the one computed in section 2.3, even with the added margins.

  Note that all resampled images (in both the `FUSE/` and the `SEG/` directories have the same geometry than the template image.

2.8 Sequence properties computation [1]

We assume that we are located in the directory `/path/to/astec/tutorial/tuto-astec1/`. Computing cell properties as well as lineage assumes that segmentation or post-corrected segmentation (see section 2.11) images have been co-registered (see sections 2.7 and 2.10). Extracting the sequence properties from the co-registered segmentation images is done with

```
$ X-embryoproperties.py -p parameters/X-embryoproperties-tutorial-parameters-seg.py
```

`X-embryoproperties-tutorial-parameters-seg.py` being the dedicated parameter file (figure 2.8).

- The variable `PATH_EMBRYO` is the path to the directory where the directory `FUSE/` is located. It can be either relative (as in the above example) or global (it could have been `/path/to/astec/tutorial/tuto-astec1/`).
- The variable `EN` is the prefix after which the images are named.
- The variables `begin` and `end` set respectively the first and the last index of the input time points to be processed.

- the variable `EXP_INTRAREG` set the suffix of the sub-directory of the `INTRAREG/` directory where to search post-corrected segmentation or segmentation images.

  Since the directory `INTRAREG/INTRAREG_SEG/` does only contains the co-registered segmentation images (in the `SEG/` sub-directory), properties will be computed from these images.

After processing, some files appears in the `INTRAREG/INTRAREG_SEG/SEG/` sub-directory

```
INTRAREG/
├── INTRAREG_RELEASE/
│   └── ...
└── INTRAREG_SEG/
    ├── CO-TRSFS/
    │   └── ...
    ├── FUSE/
    │   └── ...
    ├── LOGS/
    ├── MOVIES/
    │   └── ...
    ├── SEG/
    │   ├── 2019-Tutorial100_intrareg_seg_lineage.pkl
    │   ├── 2019-Tutorial100_intrareg_seg_lineage.txt
    │   ├── 2019-Tutorial100_intrareg_seg_lineage.xml
    │   ├── 2019-Tutorial100_intrareg_seg_t000.inr
    │   ├── ...
    │   └── 2019-Tutorial100_intrareg_seg_t020.inr
    └── TRSFS_t0-20/
        └── ...
```

`2019-Tutorial100_intrareg_seg_lineage.pkl` is a pickle python file containing a dictionary (in the python sense). It can be read by

```
$ python
...
>>> import cPickle as pkl
>>> f=open('2019-Tutorial100_intrareg_seg_lineage.pkl')
>>> d=pkl.load(f)
>>> f.close()
>>> d.keys()
['all_cells', 'cell_barycenter', 'cell_contact_surface',
'cell_principal_vectors', 'cell_principal_values', 'cell_volume',
'cell_compactness', 'cell_surface', 'cell_lineage']
```

In this pickle file (as in the one computed at section 2.6), cells have an unique identifier $i * 1000 + c$, which is made of both the image index i and the cell identifier c within a segmentation image (recall that cells are numbered from 2, 1 being the background label).

`2019-Tutorial100_intrareg_seg_lineage.xml` contains the same information than the pickle file, but in xml format (see figure 2.9).

`2019-Tutorial100_intrareg_seg_lineage.tst` contains some *diagnosis* information (smallest and largest cells, weird lineages, etc.).

```
<data>
  <cell_volume>
    ...
  </cell_volume>
  <cell_surface>
     ...
  </cell_surface>
  <cell_compactness>
     ...
  </cell_compactness>
  <cell_barycenter>
     ...
  </cell_barycenter>
  <cell_principal_values>
     ...
  </cell_principal_values>
  <cell_principal_vectors>
     ...
  </cell_principal_vectors>
  <cell_contact_surface>
     ...
  </cell_contact_surface>
  <all_cells>[2, 3, 4, 5, 6, 7, 8, 11, 12, 13,
     ...
     200097, 200099, 200100, 200101, 200102]</all_cells>
  <cell_lineage>
     ...
  </cell_lineage>
</data>
```

Figure 2.9: XML output properties file from the co-registered segmentation image.

```
 1 PATH_EMBRYO = '.'
 2
 3 EN = '2019-Tutorial100'
 4
 5 begin = 0
 6 end = 20
 7 delta = 1
 8 raw_delay = 0
 9
10 postcor_Volume_Threshold=10000
11 postcor_Soon=True
```

Figure 2.10: Tutorial parameter file for the segmentation post-correction.

2.9 Segmentation post-correction

We assume that we are located in the directory `/path/to/astec/tutorial/tuto-astec1/`. Segmentation post-correction is done with

```
$ 5-postcorrection.py -p parameters/5-postcorrection-tutorial-parameters.py
```

`5-postcorrection-tutorial-parameters.py` being the dedicated parameter file (figure 2.10).

After processing, a `POST/` directory has been created

```
path/to/tuto-astec1/
├── 2019-Tutorial100_seg_lineage.pkl
├── FUSE/
├── INTRAREG/
├── POST/
├── RAWDATA/
├── README
├── SEG/
└── parameters/
```

The `POST/` directory contains

```
POST/
└── POST_RELEASE/
    ├── 2019-Tutorial100_post_lineage.pkl
    ├── 2019-Tutorial100_post_lineage.test
    ├── 2019-Tutorial100_post_t000.inr
    ├── ...
    ├── 2019-Tutorial100_post_t020.inr
    ├── 5-postcorrection-tutorial-parameters.py
    └── 5-postcorrection.log
```

```
 1 PATH_EMBRYO = '.'
 2
 3 EN = '2019-Tutorial100'
 4
 5 begin = 0
 6 end = 20
 7
 8 EXP_INTRAREG = 'POST'
 9
10 intra_registration_template_type = "POST-SEGMENTATION"
11 intra_registration_template_threshold = 2
12 intra_registration_margin = 20
13
14 intra_registration_resample_post_segmentation_images = True
15 intra_registration_resample_segmentation_images = True
16 intra_registration_movie_post_segmentation_images = True
17 intra_registration_movie_segmentation_images = True
```

Figure 2.11: Tutorial parameter file for the sequence intra-registration step, post-segmentation images being used to build the template.

2.10 Sequence intra-registration (or drift compensation) [3]

We assume that we are located in the directory `/path/to/astec/tutorial/tuto-astec1/`. Running the sequence intra-registration is done with

```
$ 1.5-intraregistration.py -p parameters/1.5-intraregistration-tutorial-parameters-post.py
```

`1.5-intraregistration-tutorial-parameters-post.py` being the dedicated parameter file (figure 2.11).

- The variable `PATH_EMBRYO` is the path to the directory where the directory `FUSE/` is located. It can be either relative (as in the above example) or global (it could have been `/path/to/astec/tutorial/tuto-astec1/`).
- The variable `EN` is the prefix after which the images are named.
- The variables `begin` and `end` set respectively the first and the last index of the input time points to be processed.
- the variable `EXP_INTRAREG` set the suffix of the sub-directory of the `INTRAREG/` directory to be created.
- the variable `intra_registration_template_type` set the images to be used to build the template. Here, since it is equal to `"POST-SEGMENTATION''`, they are the post-corrected segmentation images obtained at the previous step.

  The variable `intra_registration_template_threshold` set a threshold to be applied to the template images to define the information to be kept: we want all the points with a value equal or greater than 2 to be contained in the template after resampling. Since cells are labeled from 2 and above, the template is designed to contain all labeled cells after resampling, so it is built as small as possible.

  The variable `intra_registration_margin` allows to add margins (in the 3 dimensions) to the built template.
- The variable `intra_registration_resample_post_segmentation_images` indicates whether the post-corrected segmentation images are to be resampled in the template geometry.
- The variable `intra_registration_resample_segmentation_images` indicates whether the segmentation images are to be resampled in the template geometry.

- The variable `intra_registration_movie_post_segmentation_images` indicates whether 2D+t movies have to be built from the resampled post-corrected segmentation images.
- The variable `intra_registration_movie_segmentation_images` indicates whether 2D+t movies have to be built from the resampled segmentation images.

After processing, a `INTRAREG/INTRAREG_POST/` directory has been created and the `INTRAREG/` directory now contains

```
INTRAREG/
├── INTRAREG_POST/
│   ├── CO-TRSFS/
│   │   ├── 2019-Tutorial100_intrareg_flo000_ref001.trsf
│   │   ├── ...
│   │   └── 2019-Tutorial100_intrareg_flo019_ref020.trsf
│   ├── FUSE/
│   │   ├── 2019-Tutorial100_intrareg_fuse_t000.inr
│   │   ├── ...
│   │   └── 2019-Tutorial100_intrareg_fuse_t020.inr
│   ├── LOGS/
│   ├── MOVIES/
│   │   ├── 2019-Tutorial100_intrareg_fuse_t0-20_xy174.inr
│   │   ├── 2019-Tutorial100_intrareg_post_t0-20_xy174.inr
│   │   └── 2019-Tutorial100_intrareg_seg_t0-20_xy174.inr
│   ├── POST/
│   │   ├── 2019-Tutorial100_intrareg_post_t000.inr
│   │   ├── ...
│   │   └── 2019-Tutorial100_intrareg_post_t020.inr
│   ├── SEG/
│   │   ├── 2019-Tutorial100_intrareg_seg_t000.inr
│   │   ├── ...
│   │   └── 2019-Tutorial100_intrareg_seg_t020.inr
│   └── TRSFS_t0-20/
│       ├── 2019-Tutorial100_intrareg_t000.trsf
│       ├── ...
│       ├── 2019-Tutorial100_intrareg_t020.trsf
│       └── template_t0-20.inr
├── INTRAREG_RELEASE/
│   └── ...
└── INTRAREG_SEG/
    └── ...
```

In addition to directories already described in section 2.3, the `INTRAREG_POST/` directory contains

- The directory `POST/` contains the resampled post-corrected segmentation images in the same geometry (images have the same dimensions along X, Y and Z), with drift compensation (the eventual motion of the sample under the microscope has been compensated).
- In addition to a 2D+t movie made from the resampled fusion and the segmentation images, the directory `MOVIES/` contains a 2D+t movie made from the resampled post-corrected segmentation images.
- The template image `template_t0-20.inr` in the directory `TRSFS/` is now of size $323 \times 265 \times 348$ with a voxel size of 0.6, has the same size than the one computed in section 2.7, which is expected since the post-correction does not change the background. Note that all resampled images (in the `FUSE/`, the `POST/`, and the `SEG/` directories have the same geometry than the template image.

```
1 PATH_EMBRYO = '.'
2
3 EN = '2019-Tutorial100'
4
5 begin = 0
6 end = 20
7
8 EXP_INTRAREG = 'POST'
```

Figure 2.12: Tutorial parameter file for the sequence properties from the co-registered post-corrected segmentation images.

2.11 Sequence properties computation [2]

We assume that we are located in the directory `/path/to/astec/tutorial/tuto-astec1/`. Computing cell properties as well as lineage assumes that segmentation or post-corrected segmentation (see section 2.11) images have been co-registered (see sections 2.7 and 2.10). Extracting the sequence properties from the co-registered segmentation images is done with

```
$ X-embryoproperties.py -p parameters/X-embryoproperties-tutorial-parameters-post.py
```

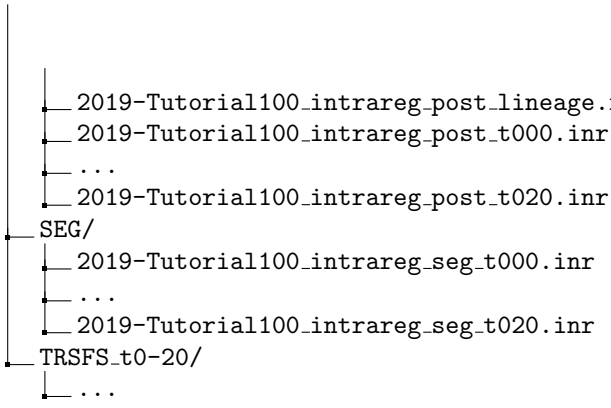
`X-embryoproperties-tutorial-parameters-post.py` being the dedicated parameter file (figure 2.12).

- The variable `PATH_EMBRYO` is the path to the directory where the directory `FUSE/` is located. It can be either relative (as in the above example) or global (it could have been `/path/to/astec/tutorial/tuto-astec1/`).
- The variable `EN` is the prefix after which the images are named.
- The variables `begin` and `end` set respectively the first and the last index of the input time points to be processed.
- the variable `EXP_INTRAREG` set the suffix of the sub-directory of the `INTRAREG/` directory where to search post-corrected segmentation or segmentation images.

  Since the directory `INTRAREG/INTRAREG_POST/` does only contains the co-registered post-corrected segmentation images (in the `POST/` sub-directory), properties will be computed from these images preferably to the co-registered segmentation images (in the `SEG/` sub-directory).

After processing, some files appears in the `INTRAREG/INTRAREG_POST/POST/` sub-directory

```
INTRAREG/
├── INTRAREG_RELEASE/
│   └── ...
└── INTRAREG_SEG/
    ├── CO-TRSFS/
    │   └── ...
    ├── FUSE/
    │   └── ...
    ├── LOGS/
    ├── MOVIES/
    │   └── ...
    └── POST/
        ├── 2019-Tutorial100_intrareg_post_lineage.pkl
        └── 2019-Tutorial100_intrareg_post_lineage.txt
```



Those files have the same meaning than the ones already presented in section 2.8.

Chapter 3

User guide: command line interfaces

The Astec distribution contains 4 sub-directories

```
path/to/astec/
├── documentation/
├── parameter-file-examples/
├── src/
└── tutorial/
```

- `documentation/` contains this documentation
- `parameter-file-examples/` contains examples of parameter files for the command line interfaces (CLIs) that are introduced below. These files are named after the CLI name.
- `src/` contains the command line interfaces (CLIs) and the code.
- `tutorial/` contains a tutorial (see chap. 2) along with a toy example.

Data organization

It is assumed that there will be one directory per experiment. This directory contains the acquired data, but will also contain the result data as depicted below.

```
/path/to/experiment/
├── RAWDATA/
│   └── ...
├── FUSE/
│   └── ...
├── SEG/
│   └── ...
└── POST/
    └── ...
```

`RAWDATA/` is assumed to contain the raw data (ie acquired images from the MuViSPIM microscope), while the other subdirectories will contain processing results.

3.1 `1-fuse.py`

The fusion is made of the following steps.

1. Optionally, a slit line correction. Some Y lines may appear brighter in the acquisition and causes artifacts in the reconstructed (i.e. fused) image. By default, it is not done.
2. A change of resolution in the X and Y directions only (Z remains unchanged). It allows to decrease the data volume (and then the computational cost) if the new pixel size (set by `target_resolution`) is larger than the acquisition one.
3. Optionally, a crop of the resampled acquisitions. It allows to decrease the volume of data, hence the computational cost. The crop is based on the analysis of a MIP view (in the Z direction) of the volume, and thus is sensitive to hyper-intensities if any. By default, it is done.
4. Optionally, a mirroring (along the **X** axis) of the right camera image. It depends on the value of `raw_mirrors` variable.
5. Linear registration of the 3 last images on the first one (considered as the reference). The reference image is resampled again, to get an isotropic voxel (whose size is given by `target_resolution`), i.e. the voxel size is the same along the 3 directions: X, Y, Z.
6. Linear combination of images, weighted by an ad-hoc function.

   Important: The used weighting function assumes that the stacking order in the Z direction is increasing, meaning that high-contrasted images are at the beginning of the stack (small z values) while blurred ones at the end, as in figure 3.1.
7. Optionally, a crop of the fused image, still based on the analysis of a MIP view (in the Z direction). By default, it is done.

3.1.1 `1-fuse.py` options

The following options are available:

`-h` prints a help message
`-p` `file` set the parameter file to be parsed
`-e` `path` set the `path` to the directory where the `RAWDATA/` directory is located
`-k` allows to keep the temporary files
`-f` forces execution, even if (temporary) result files are already existing
`-v` increases verboseness (both at console and in the log file)
`-nv` no verboseness
`-d` increases debug information (in the log file)
`-nd` no debug information

3.1.2 Important parameters in the parameter file

A simple parameter file for fusion is described in the tutorial section 2.2. A more comprehensive parameter file example is provided in the `parameter-file-examples/` directory.

Indicating the right values of the acquisition parameters is crucial; these parameters are

- `raw_ori` is a parameter describing the acquisition orientation of the acquisition of the second pair of images. Its value can be either `'left'` (orientation of 270°) or `'right'` (orientation of 90°), see figures 3.1 and 3.2.
- `raw_mirrors` is a parameter indicating whether the right camera images have been mirrored along the **X** axis or not. Its value is either `False` or `True`. For acquisitions depicted in figures 3.1 and 3.2, it has to be set to `False`, meaning that the mirroring has to be done.
- `raw_resolution` is the voxel size (along the 3 dimensions X, Y and Z) of the acquired images.
- `target_resolution` is the desired isotropic (the same along the 3 dimensions) voxel size for the result fusion images.
- `begin` gives the index of the first time point to be processed.
- `end` gives the index of the last time point to be processed.

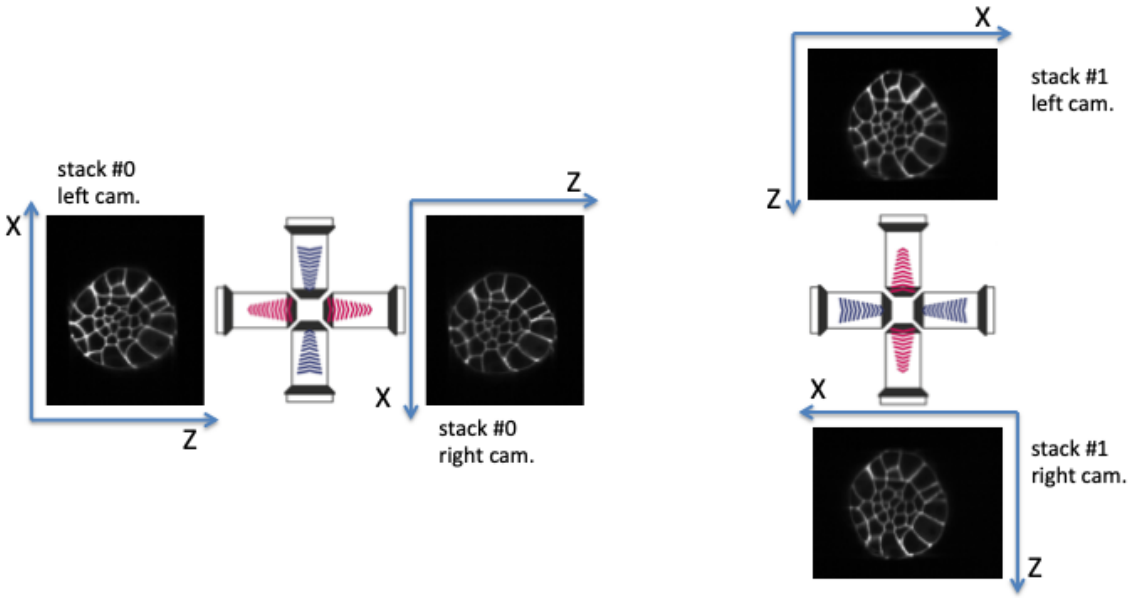


Figure 3.1: Multiview lightsheet microscope acquisition: at a time point, two acquisitions (stack #0 and stack #1) are sequentially performed, the second one orthogonal to the first. For each acquisition, two 3D intensity image stacks are acquired, respectively by the left and the right cameras. It yields four image stacks to be fused. The frame $(\mathbf{X}, \mathbf{Z})$ of the left camera of stack #0 needs to be rotated clockwise (90 degrees along the $\mathbf{Y}$ axis) to correspond to the frame of the left camera of stack #1: `raw_ori` has to be set to `'right'`.

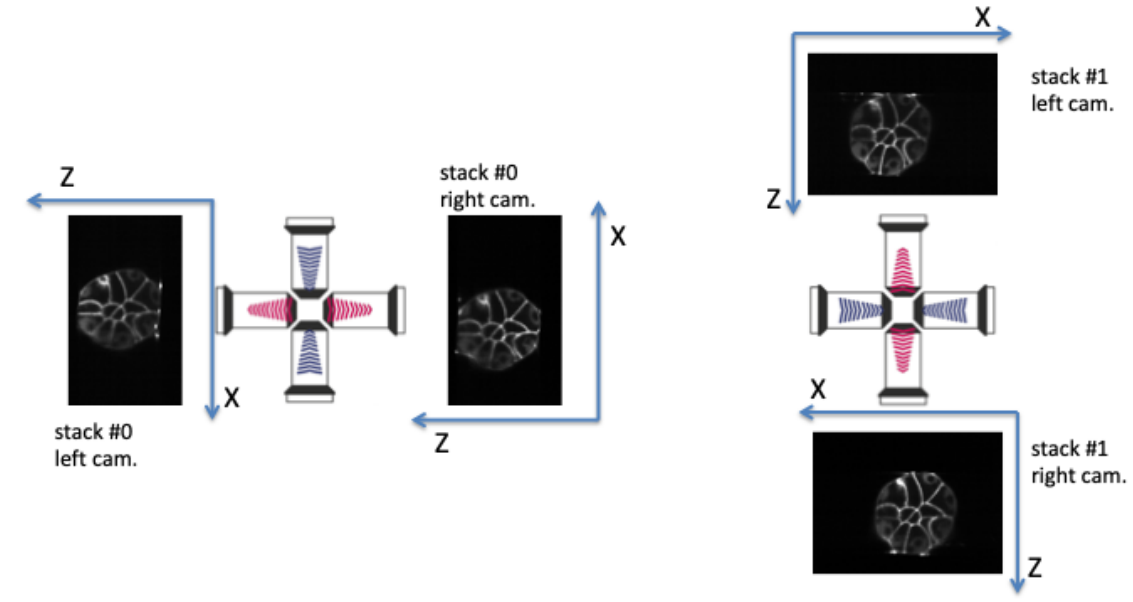


Figure 3.2: The frame $(\mathbf{X}, \mathbf{Z})$ of the left camera of stack #0 needs to be rotated counterclockwise (-90 degrees along the $\mathbf{Y}$ axis) to correspond to the frame of the left camera of stack #1: `raw_ori` has to be set to `'left'`.

When one may not be sure of the `raw_ori` and `raw_mirrors` right values, it is advised to perform the fusion on only one time point (by indicating the same index for both `begin` and `end`), with the four possibilities for the variable couple (`raw_ori`, `raw_mirrors`), i.e. (`'left'`, `False`), (`'left'`, `True`), (`'right'`, `False`), and (`'right'`, `True`). It comes to write four parameter files that differ only for the parameters `raw_ori`, `raw_mirrors`, and `EXP_FUSE` (to store the fusion result in different directories, see section 3.1.4). For these first experiments, it is also advised to set `target_resolution` to a large value, in order to speed up the calculations.

3.1.3 Input data

Input data (acquired images from the MuViSPIM microscope) are assumed to be organized in a separate `RAWDATA/` directory in the `/path/to/experiment/` directory as depicted below.

- `RAWDATA/LC/Stack000` contains the images acquired at the first angulation by the left camera.
- `RAWDATA/LC/Stack001` contains the images acquired at the second angulation by the left camera.
- `RAWDATA/RC/Stack000` contains the images acquired at the first angulation by the right camera.
- `RAWDATA/RC/Stack001` contains the images acquired at the second angulation by the right camera.

```
/path/to/experiment/
├── RAWDATA/
│   ├── LC/
│   │   ├── Stack0000/
│   │   │   ├── Time000xxx_00.zip
│   │   │   ├── ...
│   │   │   └── Time000xxx_00.zip
│   │   └── Stack0001/
│   │       ├── Time000xxx_00.zip
│   │       ├── ...
│   │       └── Time000xxx_00.zip
│   └── RC/
│       ├── Stack0000/
│       │   ├── Time000xxx_00.zip
│       │   ├── ...
│       │   └── Time000xxx_00.zip
│       └── Stack0001/
│           ├── Time000xxx_00.zip
│           ├── ...
│           └── Time000xxx_00.zip
└── ...
```

where `xxx` denotes a three digit number (e.g. 000, 001, ...) denoting the time point of each acquisition. The range of time points to be fused are given by the variables `begin` and `end`, while the path `/path/to/experiment/` has to be assigned to the variable `PATH_EMBRYO`

Hence a parameter file containing

```
PATH_EMBRYO = /path/to/experiment/
begin = 0
end = 10
```

indicates that time points in $[0, 10]$ of the `RAWDATA/` subdirectory of `/path/to/experiment/` have to be fused.

3.1.3.1 Input data directory names

However, directories may be named differently. The variables DIR_RAWDATA, DIR_LEFTCAM_STACKZERO, DIR_RIGHTCAM_STACKZER DIR_LEFTCAM_STACKONE, and DIR_RIGHTCAM_STACKONE allow a finer control of the directory names. The images acquired at the first angulation by the left and the right cameras are searched in the directories

<PATH_EMBRYO>/<DIR_RAWDATA>/<DIR_LEFTCAM_STACKZERO>
<PATH_EMBRYO>/<DIR_RAWDATA>/<DIR_RIGHTCAM_STACKZERO>

while the images acquired at the second angulation by the left and the right cameras are searched in the directories

<PATH_EMBRYO>/<DIR_RAWDATA>/<DIR_LEFTCAM_STACKONE>
<PATH_EMBRYO>/<DIR_RAWDATA>/<DIR_RIGHTCAM_STACKONE>

where <XXX> denotes the value of the variable XXX. Then, to parse the following data architecture

```
/path/to/experiment/
├── my_raw_data/
│   ├── LeftCamera/
│   │   ├── FirstStack/
│   │   │   └── ...
│   │   └── SecondStack/
│   │       └── ...
│   └── RightCamera/
│       ├── FirstStack/
│       │   └── ...
│       └── SecondStack/
│           └── ...
└── ...
```

one has to add the following lines in the parameter file

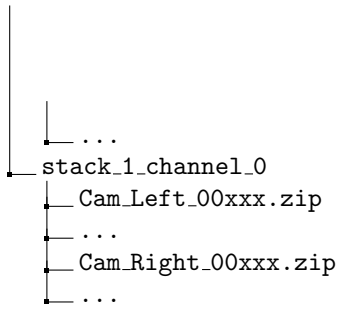
```
DIR_RAWDATA = 'my_raw_data'
DIR_LEFTCAM_STACKZERO = 'LeftCamera/FirstStack'
DIR_RIGHTCAM_STACKZERO = 'RightCamera/FirstStack'
DIR_LEFTCAM_STACKONE = 'LeftCamera/SecondStack'
DIR_RIGHTCAM_STACKONE = 'RightCamera/SecondStack'
```

It has to be noted that, when the stacks of a given time point are in different directories, image file names are tried to be guessed from the directories parsing. It has to be pointed out that indexes have to be encoded with a 3-digit integer with 0 padding (i.e. 000, 001, ...) and that has to be the only variation in the file names (within each directory).

3.1.3.2 Input data image file names

Images acquired from the left and the right cameras may be stored in the same directory, but obviously with different names as in

```
/path/to/experiment/
└── RAWDATA/
    └── stack_0_channel_0
        ├── Cam_Left_00xxx.zip
        ├── ...
        └── Cam_Right_00xxx.zip
```



The parameter file has then to contain the following lines to indicate the directory names.

```
DIR_LEFTCAM_STACKZERO = 'stack_0_channel_0'
DIR_RIGHTCAM_STACKZERO = 'stack_0_channel_0'
DIR_LEFTCAM_STACKONE = 'stack_1_channel_0'
DIR_RIGHTCAM_STACKONE = 'stack_1_channel_0'
```

In addition, to distinguish the images acquired by the left camera to those acquired by the right one, one has to give the image name prefixes, i.e. the common part of the image file names before the 3-digit number that indicates the time point. This is the purpose of the variables `acquisition_leftcam_image_prefix` and `acquisition_rightcam_image_prefix`. The parameter file has then to contain the following lines not only to indicate the directory names but also the image file name prefixes.

```
DIR_LEFTCAM_STACKZERO = 'stack_0_channel_0'
DIR_RIGHTCAM_STACKZERO = 'stack_0_channel_0'
DIR_LEFTCAM_STACKONE = 'stack_1_channel_0'
DIR_RIGHTCAM_STACKONE = 'stack_1_channel_0'
acquisition_leftcam_image_prefix = 'Cam_Left_00'
acquisition_rightcam_image_prefix = 'Cam_Right_00'
```

3.1.3.3 Multichannel acquisition

In case of multichannel acquisition, the fusion is computed for the first channel, and the computed parameters (e.g. transformations, etc.) are also used for the other channels.

For a second channel, the images acquired at the first angulation by the left and the right cameras are searched in the directories

```
<PATH_EMBRYO>/<DIR_RAWDATA>/<DIR_LEFTCAM_STACKZERO_CHANNEL_2>
<PATH_EMBRYO>/<DIR_RAWDATA>/<DIR_RIGHTCAM_STACKZERO_CHANNEL_2>
```

while the images acquired at the second angulation by the left and the right cameras are searched in the directories

```
<PATH_EMBRYO>/<DIR_RAWDATA>/<DIR_LEFTCAM_STACKONE_CHANNEL_2>
<PATH_EMBRYO>/<DIR_RAWDATA>/<DIR_RIGHTCAM_STACKONE_CHANNEL_2>
```

For a third channel, the images acquired at the first angulation by the left and the right cameras are searched in the directories

```
<PATH_EMBRYO>/<DIR_RAWDATA>/<DIR_LEFTCAM_STACKZERO_CHANNEL_3>
<PATH_EMBRYO>/<DIR_RAWDATA>/<DIR_RIGHTCAM_STACKZERO_CHANNEL_3>
```

while the images acquired at the second angulation by the left and the right cameras are searched in the directories

```
<PATH_EMBRYO>/<DIR_RAWDATA>/<DIR_LEFTCAM_STACKONE_CHANNEL_3>
<PATH_EMBRYO>/<DIR_RAWDATA>/<DIR_RIGHTCAM_STACKONE_CHANNEL_3>
```

3.1.4 Output data

The variable `target_resolution` allows to set the desired isotropic (the same along the 3 dimensions) voxel size for the result fusion images.

3.1.4.1 Output data directory names

The resulting fused images are stored in sub-directory `FUSE/FUSE_<EXP_FUSE>` under the `/path/to/experiment/` directory

```
/path/to/experiment/
├── RAWDATA/
│   └── ...
└── FUSE/
    ├── FUSE_<EXP_FUSE>/
    │   └── ...
```

where `<EXP_FUSE>` is the value of the variable `EXP_FUSE` (its default value is '`RELEASE`'). Hence, the line

```
EXP_FUSE = 'TEST'
```

in the parameter file will create the directory `FUSE/FUSE_TEST/` in which the fused images are stored. For instance, when testing for the values of the variable couple (`raw_ori`, `raw_mirrors`), a first parameter file may contain

```
raw_ori = 'left'
raw_mirrors = False
begin = 1
end = 1
EXP_FUSE=TEST-LEFT-FALSE
```

a second parameter file may contain

```
raw_ori = 'left'
raw_mirrors = True
begin = 1
end = 1
EXP_FUSE=TEST-LEFT-TRUE
```

etc. The resulting fused images will then be in different directories

```
/path/to/experiment/
├── RAWDATA/
│   └── ...
└── FUSE/
    ├── FUSE_TEST-LEFT-FALSE/
    │   └── ...
    ├── FUSE_TEST-LEFT-TRUE/
    │   └── ...
    └── ...
```

This will ease their visual inspection to decide which values of the variable couple (`raw_ori`, `raw_mirrors`) to use for the fusion.

3.1.4.2 Output data file names

Fused image files are named after the variable `EN`: `<EN>_fuse_t<xxx>.inr` where `<xxx>` is the time point index encoded by a 3-digit integer (with 0 padding).

3.1.4.3 Multichannel acquisition

Variables `EXP_FUSE_CHANNEL_2` and `EXP_FUSE_CHANNEL_3` allows to set the directory names for the resulting fused images of the other channels.

3.1.5 Fusion parameters

3.1.5.1 Step 3: raw data cropping

For computational cost purposes, raw data (images acquired by the MuViSPIM microscope) are cropped (only in X and Y dimensions) before co-registration. A threshold is computed with Otsu's method [Ots79] on the maximum intensity projection (MIP) image. The cropping parameters are computed to keep the above-threshold points in the MIP image, plus some extra margins. Hyper-intense areas may biased the threshold computation, hence the cropping.

To desactivate this cropping, the line

```
raw_crop = False
```

has to be added in the parameter file.

3.1.5.2 Step 5: linear registration

To decrease the computational cost, images are normalized and cast on one byte before registration. While it generally does not degrade the registration quality, it may induce troubles when hyper-intensities areas are present in the image. In such a case, the useful information may then be summarized in only a few intensity values.

Intensity normalization in registration can be deactivated by adding the following line in the parameter file

```
fusion_registration_normalization = False
```

To verify whether a good quality registration can be conducted, the searched transformation type can be changed for a simpler one than affine. Adding the following line in the parameter file.

```
fusion_registration_transformation_type = translation
```

will search for a translation which is supposed to be sufficient, according that only translations relates the 4 acquisitions of the MuViSPIM microscope (in a perfect setting). If the search for an affine transformation (the default behavior) failed (the fusion looks poor) while the search for a translation is successful (the fusion looks good), a two-steps registration may help to refine the found translation by a subsequent affine transformation as explained below.

Hyper-intensities areas may also bias the threshold calculation used for the automatic crop (step 3 of fusion). In such cases, the iterative registration method may find a local minimum that is not the desired one, because the relative positions of the two images to be co-registered are too far apart. To circumvent such a behavior, a two-steps registration can be done. It consists on a first pre-registration with a transformation with fewer degrees of freedom (i.e. a 3D translation).

This pre-registration can be activated by adding the following line in the parameter file.

```
fusion_preregistration_compute_registration = True
```

It may be also preferable to deactivate the image normalization for both registration steps with

```
fusion_preregistration_normalization = False
fusion_registration_normalization = False
```

3.1.5.3 Step 7: fused data cropping

To save disk storage, fused images are cropped at the end of the fusion stage. To desactivate this cropping, the line

```
fusion_crop = False
```

has to be added in the parameter file.

3.1.6 Troubleshooting

- The fused images are obviously wrong.
  1. Are the values of the variable couple (`raw_ori`, `raw_mirrors`) the right ones? Conduct experiments as suggested in section 3.1.2 (see also section 3.1.4) to get the right values.
  2. The registration may have failed.
     (a) Deactivate the 1-byte normalization (see section 3.1.5.2).
     (b) Try to register with a simpler transformation type (i.e. translation) and/or with a two-steps registration (see section 3.1.5.2).
- The imaged sample is cropped by the image border in the fused image.
  1. Check whether the imaged sample was not already cropped in the raw data.
  2. The automated cropping may have failed. It is more likely to happen when cropping the raw data, so deactivate it (see section 3.1.5.1). If it still happens, try to deactivate also the fused image cropping (see section 3.1.5.3).

3.1.7 Parameter list

Please also refer to the file `parameter-file-examples/1-fuse-parameters.py`

- `DIR_LEFTCAM_STACKONE` see section 3.1.3
- `DIR_LEFTCAM_STACKONE_CHANNEL_2` see section 3.1.3
- `DIR_LEFTCAM_STACKONE_CHANNEL_3` see section 3.1.3
- `DIR_LEFTCAM_STACKZERO` see section 3.1.3
- `DIR_LEFTCAM_STACKZERO_CHANNEL_2` see section 3.1.3
- `DIR_LEFTCAM_STACKZERO_CHANNEL_3` see section 3.1.3
- `DIR_RAWDATA` see section 3.1.3
- `DIR_RAWDATA_CHANNEL_2` see section 3.1.3
- `DIR_RAWDATA_CHANNEL_3` see section 3.1.3
- `DIR_RIGHTCAM_STACKONE` see section 3.1.3
- `DIR_RIGHTCAM_STACKONE_CHANNEL_2` see section 3.1.3
- `DIR_RIGHTCAM_STACKONE_CHANNEL_3` see section 3.1.3
- `DIR_RIGHTCAM_STACKZERO` see section 3.1.3
- `DIR_RIGHTCAM_STACKZERO_CHANNEL_2` see section 3.1.3
- `DIR_RIGHTCAM_STACKZERO_CHANNEL_3` see section 3.1.3
- `EN` see section 3.1.4
- `EXP_FUSE` see section 3.1.4
- `EXP_FUSE_CHANNEL_2` see section 3.1.4
- `EXP_FUSE_CHANNEL_3` see section 3.1.4
- `PATH_EMBRYO` see section 3.1.3
- `RESULT_IMAGE_SUFFIX_FUSE`
- `acquisition_leftcam_image_prefix` see section 3.1.3
- `acquisition_mirrors` same as `raw_mirrors`

- acquisition_orientation same as raw_ori
- acquisition_resolution same as raw_resolution
- acquisition_rightcam_image_prefix see section 3.1.3
- acquisition_slit_line_correction
- begin see section 3.1.2
- default_image_suffix
- delta
- end see section 3.1.2
- fusion_crop see section 3.1.5.3
- fusion_margin_x_0
- fusion_margin_x_1
- fusion_margin_y_0
- fusion_margin_y_1
- fusion_preregistration_compute_registration see section 3.1.5.2
- fusion_preregistration_lts_fraction
- fusion_preregistration_normalization see section 3.1.5.2
- fusion_preregistration_pyramid_highest_level
- fusion_preregistration_pyramid_lowest_level
- fusion_preregistration_transformation_estimation_type
- fusion_preregistration_transformation_type
- fusion_registration_compute_registration
- fusion_registration_lts_fraction
- fusion_registration_normalization see section 3.1.5.2
- fusion_registration_pyramid_highest_level
- fusion_registration_pyramid_lowest_level
- fusion_registration_transformation_estimation_type
- fusion_registration_transformation_type see section 3.1.5.2
- raw_crop see section 3.1.5.1
- raw_delay
- raw_margin_x_0
- raw_margin_x_1
- raw_margin_y_0
- raw_margin_y_1
- raw_mirrors see section 3.1.2
- raw_ori see section 3.1.2
- raw_resolution see section 3.1.2
- result_image_suffix
- target_resolution see section 3.1.4

3.2 1.5-intraregistration.py

The sequence intra-registration procedure can be done either after the fusion step, or after the (post-)segmentation step. It aims at

- compensating for the eventual motion of the imaged sample with respect to the microscope
- resampling the fusion and/or the segmentation images into a common frame/geometry, so they can better be compared, and
- building 2D+t images made of 2D sections from either the fusion and/or the segmentation images, so that the quality of the fusion and/of the tracking step can be visually assessed.

The intra-registration procedure is made of the following steps:

1. Co-registration of pairs of successive fused images (section 3.2.3). This yields the transformations $T_{t+1 \leftarrow t}$. Fused images are located in `<EMBRYO>/FUSE/FUSE_<EXP_FUSE>`: the parameter `EXP_FUSE` is either set in the parameter file or is set at `RELEASE`. This step may be long.
2. Composition of transformations issued from the co-registration step. This step computes the transformations $T_{ref \leftarrow t}$. towards a reference image `ref` given by the parameter `intra_registration_reference_index`.
3. Computation of the *template* image (section 3.2.4). This *template* image dimension are computed so that the useful information of all resampled images fits into it. Useful information can be issued from either the fused sequence, the segmentation sequence or the post-segmentation sequence. It is indicated by the `intra_registration_template_type` which value can be either `'FUSION'`, `'SEGMENTATION'`, or `'POST-SEGMENTATION'`. This step may be long.
4. Resampling of either the fused or the segmentation images (section 3.2.4.1). Note that changing the parameters for this step will not require to re-compute the first steps.
5. Extraction of 2D+t images from the resampled sequences (section 3.2.5). Note that changing the parameters for this step (i.e. requiring extra movies) will not require to re-compute the first steps, with an eventual exception for the resampling step.

3.2.1 `1.5-intraregistration.py` options

The following options are available:

`-h` prints a help message
`-p file` set the parameter file to be parsed
`-e path` set the `path` to the directory where the `RAWDATA/` directory is located
`-k` allows to keep the temporary files
`-f` forces execution, even if (temporary) result files are already existing
`-v` increases verboseness (both at console and in the log file)
`-nv` no verboseness
`-d` increases debug information (in the log file)
`-nd` no debug information

3.2.2 Output data

The results are stored in sub-directories `INTRAREG/INTRAREG_<EXP_INTRAREG>` under the `/path/to/experiment/` directory where `<EXP_INTRAREG>` is the value of the variable `EXP_INTRAREG` (its default value is '`RELEASE`').

```
/path/to/experiment/
├── ...
├── INTRAREG/
│   └── INTRAREG_<EXP_INTRAREG>/
│       ├── CO-TRSFS/
│       ├── FUSE/
│       ├── LOGS/
│       ├── MOVIES/
│       ├── [POST/]
│       ├── [SEG/]
│       └── TRSFS_t<begin>-<end>/
└── ...
```

`INTRAREG/INTRAREG_<EXP_INTRAREG>/FUSE` contains the fused images after resampling in a common geometry (i.e. all image have the same X, Y and Z dimensions)

```
/path/to/experiment/
├── ...
├── INTRAREG/
│   └── INTRAREG_<EXP_INTRAREG>/
│       └── ...
```

```
    |
    |
    |   |
    |   |-- FUSE/
    |   |   |-- <EN>_intrareg_fuse_t<xxx>.inr .4 ...
    |-- ...
```

3.2.3 Co-registration parameters

Default registration parameters for the co-registration are set by:

```
# intra_registration_compute_registration = True
# intra_registration_transformation_type = 'rigid'
# intra_registration_transformation_estimation_type = 'wlts'
# intra_registration_lts_fraction = 0.55
# intra_registration_pyramid_highest_level = 6
# intra_registration_pyramid_lowest_level = 3
# intra_registration_normalization = True
```

Computed transformations are stored in `INTRAREG/INTRAREG_<EXP>/CO-TRSFS`. It may be advised to set the pyramid lowest level value to some higher value to speed up the co-registrations (recall that all pairs of successive images will be co-registered, i.e.

```
intra_registration_pyramid_lowest_level = 4
```

3.2.4 Template building parameters

```
# intra_registration_reference_index = None
# intra_registration_template_type = 'FUSION'
# intra_registration_template_threshold = None
# intra_registration_resolution = 0.6
# intra_registration_margin = None
```

The `intra_registration_reference_index` allows to choose the reference image (the one which remains still, i.e. is only displaced by a translation), by default it is the first image image of the series (associated to `begin`).

Depending on `intra_registration_template_type` (`'FUSION'`, `'SEGMENTATION'` or `'POST-SEGMENTATION'`, the two latter assume obviously that the segmentation has been done), the *template* image can be built either after the fusion or the segmentation images. If no threshold is given by `intra_registration_template_threshold`, the built template will be large enough to include all the transformed fields of view (in this case, the template is the same whatever `intra_registration_template_type` is).

If a threshold is given, the built template will be large enough to include all the transformed points above the threshold. E.g., the background is labeled with either '1' or '0' in segmentation images, then a threshold of '2' ensures that all the embryo cells will not be cut by the resampling stage. In this case, adding an additional margin to the template could be a good idea for visualization purpose. Last but not least, using a larger resolution than the `target_resolution` (the resolution of the fused images) allows to decrease the resampled images volume. This can be achieved by setting `intra_registration_resolution` to a larger value than the one of `target_resolution` (default is 0.6).

Thus, building a *template* image after the segmentation images can be done with

```
# intra_registration_reference_index = None
intra_registration_template_type = "SEGMENTATION"
intra_registration_template_threshold = 2
# intra_registration_resolution = 0.6
intra_registration_margin = 10
```

Computed transformations from the *template* image as well as the *template* image itself are stored in `INTRAREG/INTRAREG<EXP>/TRSFS_t<F>-<L>/` where `<F>` and `L` are the first and the last index of the series (specified by `begin` and `end` from the parameter file).

3.2.4.1 Resampling fusion/segmentation images

The resampled fusion and segmentation images will be stored respectively in `INTRAREG/INTRAREG_<EXP>/FUSE`, `INTRAREG/INTRAREG_<EXP>/SEG/` and `INTRAREG/INTRAREG_<EXP>/POST/`. Resampling is done either if the following parameters are set to `True` or if movies are requested to be computed.

```
# intra_registration_resample_fusion_images = True
# intra_registration_resample_segmentation_images = False
# intra_registration_resample_post_segmentation_images = False
```

3.2.5 2D+t movies

For either visual assessment or illustration purposes, 2D+t (i.e. 3D) images can be built from 2D sections extracted from the resampled temporal series. This is controlled by the following parameters:

```
# intra_registration_movie_fusion_images = True
# intra_registration_movie_segmentation_images = False
# intra_registration_movie_post_segmentation_images = False

# intra_registration_xy_movie_fusion_images = [];
# intra_registration_xz_movie_fusion_images = [];
# intra_registration_yz_movie_fusion_images = [];

# intra_registration_xy_movie_segmentation_images = [];
# intra_registration_xz_movie_segmentation_images = [];
# intra_registration_yz_movie_segmentation_images = [];

# intra_registration_xy_movie_post_segmentation_images = [];
# intra_registration_xz_movie_post_segmentation_images = [];
# intra_registration_yz_movie_post_segmentation_images = [];
```

If `intra_registration_movie_fusion_images` is set to `True`, a movie is made with the XY-section located at the middle of each resampled fusion image (recall that, after resampling, all images have the same geometry). Additional XY-movies can be done by specifying the wanted Z values in `intra_registration_xy_movie_fusion_im` E.g.

```
intra_registration_xy_movie_fusion_images = [100, 200];
```

will build two movies with XY-sections located respectively at Z values of 100 and 200. The same stands for the other orientation and for the resampled segmentation images.

3.2.6 Parameter list

Please also refer to the file `parameter-file-examples/1.5-intraregistration-parameters.py`

- `EN`
- `EXP_FUSE`
- `EXP_INTRAREG`
- `EXP_POST`
- `EXP_SEG`
- `PATH_EMBRYO`
- `begin`
- `default_image_suffix`
- `delta`
- `end`

- intra_registration_compute_registration
- intra_registration_lts_fraction
- intra_registration_margin
- intra_registration_movie_fusion_images
- intra_registration_movie_post_segmentation_images
- intra_registration_movie_segmentation_images
- intra_registration_normalization
- intra_registration_pyramid_highest_level
- intra_registration_pyramid_lowest_level
- intra_registration_reference_index
- intra_registration_resample_fusion_images
- intra_registration_resample_post_segmentation_images
- intra_registration_resample_segmentation_images
- intra_registration_resolution
- intra_registration_sigma_segmentation_images
- intra_registration_template_threshold
- intra_registration_template_type
- intra_registration_transformation_estimation_type
- intra_registration_transformation_type
- intra_registration_xy_movie_fusion_images
- intra_registration_xy_movie_post_segmentation_images
- intra_registration_xy_movie_segmentation_images
- intra_registration_xz_movie_fusion_images
- intra_registration_xz_movie_post_segmentation_images
- intra_registration_xz_movie_segmentation_images
- intra_registration_yz_movie_fusion_images
- intra_registration_yz_movie_post_segmentation_images
- intra_registration_yz_movie_segmentation_images
- result_image_suffix

3.3 2-mars.py

The name mars comes from [FDM+10] where MARS is the acronym of *multiangle image acquisition, 3D reconstruction and cell segmentation.*

This method aims at producing a segmentation of a membrane cell image (e.g. a fused image) into a segmention image. This segmentation image is a integer-valued image where each integer labeled an unique cell in the image. By convention, '1' is the background label, while cells have labels greater than 2. It is is made of the following steps:

1. Optionally, a transformation of the input image.
2. A seeded watershed.

3.3.1 2-mars.py options

The following options are available:

-h prints a help message
-p file set the parameter file to be parsed
-e path set the path to the directory where the RAWDATA/ directory is located
-k allows to keep the temporary files
-f forces execution, even if (temporary) result files are already existing
-v increases verboseness (both at console and in the log file)
-nv no verboseness
-d increases debug information (in the log file)
-nd no debug information

3.3.2 Output data

The results are stored in sub-directories `SEG/SEG_<EXP_SEG>` under the `/path/to/experiment/` directory where where `<EXP_SEG>` is the value of the variable `EXP_SEG` (its default value is '`RELEASE`').

```
/path/to/experiment/
├── ...
├── SEG/
│   └── SEG_<EXP_SEG>/
│       ├── <EN>_mars_t<begin>.inr
│       ├── LOGS/
│       └── RECONSTRUCTION/
└── ...
```

3.3.3 Segmentation parameters

3.3.3.1 Input image for watershed computation

Before the watershed segmentation, the input image may be pre-processed. This pre-processing is controlled by the two variables.

- `mars_intensity_transformation` whose values are to be chosen in `None`, `'Identity'`, or `'Normalization_to_u8'`. Default is `'Identity'`.
- `mars_intensity_enhancement` whose values are to be chosen in `None` or `GACE`. Default is `None`.

Each of these variables, if not `None`, induce a transformation of the input (i.e. fused) image. If both values are not known, the input image for the watershed is the result of the maximum operator over the two images.

- `mars_intensity_transformation = 'Identity'`: the input image is not transformed.
- `mars_intensity_transformation = 'Normalization_to_u8'`: input images are usually encoded on 2 bytes. The choice transformed the input image in an 1-byte image by linearly mapping the input image values from $[I_{min}, I_{max}]$ to $[0, 255]$. I_{min} and I_{max} correspond respectively to the 1% and to the 99% percentiles of the input image cumulative histogram. Values below I_{min} are set to 0 while values above I_{max} are set to 255.
- `mars_intensity_enhancement = 'GACE'`: `GACE` stands for *Global Automated Cell Extractor*. This is the method described in [MGFM14, Mic16]. It consists in

  1. extracting a centerplane image of the membranes,
  2. thresholding this centerplane image, and
  3. reconstruct the membranes through a tensor voting method.

If the input image is transformed before segmented, the transformed image is named `<EN>_fuse_t<begin>_membrane.inr` and stored in the directory `SEG/SEG_<EXP_SEG>/RECONSTRUCTION/` if the value of the variable `mars_keep_reconstruction` is set to `True`.

3.3.3.2 Seeded watershed

The seed extraction is made of the following steps:

1. Gaussian smoothing of the input image, the gaussian standard deviation being given by the variable `watershed_seed_sigma`.
2. Extraction of the h-minima of the previous image, h being given by the variable `watershed_seed_hmin`.
3. Hysteresis thresholding (and labeling) of the h-minima image, with a high threshold equal to h and and a low threshold equal to 1. It then only selects the h-minima that have an actual depth of h.

Given the seeds, the watershed is performed on the smoothed input image (gaussian standard deviation being given by the variable `watershed_membrane_sigma`).

3.3.4 Parameter list

Please also refer to the file `parameter-file-examples/2-mars-parameters.py`

- `EN`
- `EXP_FUSE`
- `EXP_SEG`
- `PATH_EMBRYO`
- `begin`
- `default_image_suffix`
- `delta`
- `mars_begin`
- `mars_end`
- `mars_hard_threshold`
- `mars_intensity_enhancement`
- `mars_intensity_transformation`
- `mars_keep_reconstruction`
- `mars_manual`
- `mars_manual_sigma`
- `mars_sample`
- `mars_sensitivity`
- `mars_sigma_TV`
- `mars_sigma_membrane`
- `result_image_suffix`
- `watershed_membrane_sigma`
- `watershed_seed_hmin`
- `watershed_seed_sigma`

3.4 `3-manualcorrection.py`

The seeded watershed is likely to produce segmentation errors, even with a careful choice of parameters. It is advised to set the parameters to favour over-segmentations insted of under-segmentations since the former are much more easier to correct, which is the purpose of `3-manualcorrection.py`.

3.4.1 `3-manualcorrection.py` options

The following options are available:

`-h` prints a help message
`-p file` indicates the parameter file to be parsed
`-e path` indicates the `path` to the directory where the `RAWDATA/` directory is located
`-k` allows to keep the temporary files
`-f` forces execution, even if (temporary) result files are already existing
`-v` increases verboseness (both at console and in the log file)
`-nv` no verboseness
`-d` increases debug information (in the log file)
`-nd` no debug information
`-i input_image` set the `input_image` file to be corrected. Allows to skip the automated naming of files.
`-o output_image` set the resulting `ouput_image` file to be saved. Allows to skip the automated naming of files.
`-m mapping_file` set the `mapping_file` to be used for the correction.
`-nsc smallest_cells` set the number of the smallest cells to be displayed after correction. The smallest cells are the most likely to be issued from an over-segmentation.
`-nlc largest_cells` set the number of the largest cells to be displayed after correction. The largest cells are the most likely to be issued from an under-segmentation.

3.4.2 Output data

The results are stored in sub-directories SEG/SEG_<EXP_SEG> under the /path/to/experiment/ directory where <EXP_SEG> is the value of the variable EXP_SEG (its default value is 'RELEASE'). <EN>_seg_t<begin>.inr is the correction of the segmentation image <EN>_mars_t<begin>.inr.

```
/path/to/experiment/
├── ...
├── SEG/
│   └── SEG_<EXP_SEG>/
│       ├── <EN>_mars_t<begin>.inr
│       ├── <EN>_seg_t<begin>.inr
│       ├── LOGS/
│       └── RECONSTRUCTION/
└── ...
```

3.4.3 Segmentation correction parameters

3-manualcorrection.py parses a correction file whose name is given by the variable mancor_mapping_file. The syntax of this file is very simple. Lines beginning with # are ignored (and can be used to insert comments in the files). Non-empty lines should contain two numbers separated by a space, and 3-manualcorrection.py will replace the first number by the second in the segmentation file.

E.g. a cell c is recognized to be over-segmented, and then is represented by two labels, says 9 and 10. Thus the line

```
10 9
```

will replace all 10's by 9's in the segmentation image, thus c will only be represented by 9's after correction. See also the tutorial section 2.5 for an other example.

3.4.4 Parameter list

Please also refer to the file parameter-file-examples/3-manualcorrection-parameters.py

- EN
- EXP_SEG
- PATH_EMBRYO
- begin
- default_image_suffix
- delta
- mancor_input_seg_file
- mancor_mapping_file
- mancor_output_seg_file
- mars_begin
- mars_end
- result_image_suffix

3.5 4-astec.py

The name astec comes from the Phd work of L. Guignard [Gui15] where ASTEC is the acronym of *adaptive segmentation and tracking of embryonic cells*.

3.6 5-postcorrection.py

3.7 X-embryoproperties.py

X-embryoproperties.py can be used either to extract cell properties as well as cell lineage from a co-registered image sequence or to handle a property file (pkl or xml).

3.7.1 X-embryoproperties.py options

The following options are available:

-h prints a help message
-p file set the parameter file to be parsed
-e path set the path to the directory where the RAWDATA/ directory is located
-i files ... input files (pkl or xml) to be read
-o files ... output files (pkl or xml) to be read
-c files ... files (pkl or xml) to be compared to those given by -i
-feature features ... features to be extracted from the input files, that are to be written in the output files. Features have to be chosen in 'lineage', 'h_min', 'volume', 'surface', 'sigma', 'label_in_time', 'barycenter', 'fate', 'fate2', 'fate3', 'fate4', 'all-cells', 'principal-value', 'name', 'contact', 'history', 'principal-vector', 'name-score', 'cell-compactness'
-property features ... same as -feature
--diagnosis performs some test on the read properties
--diagnosis-minimal-volume DIAGNOSIS_MINIMAL_VOLUME displays all cells with volume smaller than DIAGNOSIS_MINIM
--diagnosis-items DIAGNOSIS_ITEMS minimal number of items to be displayed
--print-content print the keys of the input file(s) (read as python dictionary)
--print-keys same as --print-content
--print-types print types of read features (for debug purpose)
-k allows to keep the temporary files
-f forces execution, even if (temporary) result files are already existing
-v increases verboseness (both at console and in the log file)
-nv no verboseness
-d increases debug information (in the log file)
-nd no debug information

3.7.2 Extracting properties from a co-registered image sequence

When a parameter file is passed after the -p option, X-embryoproperties.py will compute image sequence properties. Computing cell related informations as well as the lineage tree requires that the (post-corrected) segmentation images have already been co-registered (with 1.5-intraregistration.py see section 3.2). X-embryoproperties.py will parse the INTRAREG/INTRAREG_<EXP_INTRAREG>/ directory, and will compute the properties from the images in the POST/ sub-directory, if existing, else of from the SEG/ sub-directory.

3.7.2.1 Output data

The results are stored in the POST/ or SEG/ sub-directory under the INTRAREG/INTRAREG_<EXP_INTRAREG> under the directory where <EXP_INTRAREG> is the value of the variable EXP_INTRAREG (its default value is 'RELEASE'). The resulting properties will be stored in the same directory than the images they are issued. It will be stored as a pickle python file, and also as a XML file. Both files contain exactly the same information.

According that the POST/ sub-directory exists (that post-corrected segmentation images have been co-registered), 3 files will be created, named after <EN>

```
/path/to/experiment/
└── ...
```

```
├── INTRAREG/
│   └── INTRAREG_<EXP_INTRAREG>/
│       ├── ...
│       ├── POST/
│       │   ├── <EN>_intrareg_post_lineage.pkl
│       │   ├── <EN>_intrareg_post_lineage.txt
│       │   ├── <EN>_intrareg_post_lineage.xml
│       │   ├── ...
│       ├── ...
├── ...
```

The computed information are

`all_cells` All the cell identifiers. Each cell (in a segmentation image) has a given label (ranging from 2 and above, 1 being used for the background) in each image. To uniquely identify a cell in the sequence, it has been given an unique identifier computed by $i * 1000 + c$, i and c denoting respectively the image index (ranging in [`<begin>`, `<end>`]) and the cell label.

`cell_barycenter` Cell center of mass (in voxel coordinates)

`cell_contact_surface` For each cell, give for each neighboring cell the contact surface. The sum of these contact surfaces is the cell surface.

`cell_principal_vectors` The cell principal vectors are issued from the diagonalization of the cell covariance matrix (in voxel unit).

`cell_principal_values` The cell principal value are issued from the diagonalization of the cell covariance matrix (in voxel unit).

`cell_volume` Cell volume (in voxel unit)

`cell_compactness` The cell compactness is defined by $\mathcal{C} = \frac{\sqrt[3]{\mathcal{V}}}{\sqrt[2]{\mathcal{S}}}$ where $\mathcal{V}$ is the volume of the cell and $\mathcal{S}$ is its surface.

`cell_surface` Cell surface (in pixel unit). For this computation, is mandatory that the co-registered images are isotropic (the same voxel size along the 3 dimensions X, Y, and Z).

`cell_lineage`

The text file `<EN>_intrareg_post_lineage.txt` contains diagnosis information about the sequence. It lists

- the cell with the smallest sizes as well as the ones with the largest sizes
- the cell with a weird lineage: cells without a mother cell, or cells without daughter cells or having more than 2 daughter cells
- cells having a small intersection with its mother cell with respect to either the mother cell volume or the cell volume.

3.7.2.2 Parameter list

Please also refer to the file `parameter-file-examples/X-embryoproperties-parameters.py`

- `EN`
- `EXP_INTRAREG`
- `PATH_EMBRYO`
- `begin`
- `end`
- `properties_nb_proc` the property computation supports parallelism. However, it appears that the opening of several files at the same time may cause the computation to fail. Thus, the default behavior is a sequential processing. To enable parallelism, this parameter can be set to either any negative value (causing a default parallel behavior) or to a positive value indicating the number of threads to be created.

3.7.3 Handling properties files

`X-embryoproperties.py` can also help managing property files.

- Converting from `xml` to `pkl` and the other way around.

  ```
  $ X-embryoproperties.py -i file.pkl -o file.xml
  ```

  convert the pickle file `file.pkl` into the xml file `file.xml`
- Merging files.

  ```
  $ X-embryoproperties.py -i file1.pkl file2.xml ...filen.pkl -o
  merge.xml merge.pkl
  ```

  will merge the files `file1.pkl`, `file2.xml` , ..., `filen.pkl` (note that they can be either xml or pkl) and write the result both in `xml` and `pkl` formats.
- Extracting properties.

  ```
  $ X-embryoproperties.py -i file.pkl -feature volume surface -o file.xml
  ```

  will extract the cell volume and surface information from the pickle file `file.pkl` and write them into the xml file `file.xml`

Bibliography

[FDM+10] R. Fernandez, Pradeep Das, V. Mirabet, E. Moscardi, Jan Traas, J.-L. Verdeil, Grégoire Malandain, and Christophe Godin. Imaging plant growth in 4D: robust tissue reconstruction and lineaging at cell resolution. *Nature Methods*, 7:547–553, 2010.

[Gui15] Léo Guignard. *Quantitative analysis of animal morphogenesis: from high-throughput laser imaging to 4D virtual embryo in ascidians.* Theses, Université Montpellier, December 2015.

[MGFM14] Gaël Michelin, Léo Guignard, Ulla-Maj Fiuza, and Grégoire Malandain. Embryo Cell Membranes Reconstruction by Tensor Voting. In *ISBI - International Symposium on Biomedical Imaging*, Beijing, China, April 2014. IEEE.

[Mic16] Gal Michelin. *Image analysis tools and inter-individual registration for the study of animal and plant morphogenesis.* Theses, Université Côte d'Azur, October 2016.

[Ots79] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE Trans. Sys., Man., Cyber.*, 9(1):62–66, 1979.