Contents

Chapter 1

Installation

ASTEC can run only on Linux system and was tested on:

- Ubuntu 14.04 64 bits

1.1 Requirements

In order to be able to compile the different source code you need to install several packages, mostly from the terminal application.

1.1.1 to run the different codes:

- python 2.7 or higher
  Installation:
  - Linux: Should be installed, refer with command line
    `'python -V'`
    to get the version or visit `http://php.net`
- and a C compiler
  Installation
  - Linux: Should be installed, refer with command line
    `'dpkg --list | grep compiler'`
    to get list of available C compiler version or install `gcc` with
    `'sudo apt-get install gcc'`

1.1.2 the install libraries are necessary:

- `pip`, an installer for python (`https://pypi.org/project/pip/`). Installation:
  - Linux: run command line
    `'sudo apt-get install python-pip python-dev build-essential'`
  - Others: see *Installation* from `https://pypi.org/project/pip/`
- `cmake`, a software to build source code (`http://www.cmake.org`)
  - Linux: run command line
    `'sudo apt-get install cmake'`
  - Others: see *Download* from `https://cmake.org/`

1.1.3 the following python libraries are necessary:

- numpy, scipy, matplotlib: different scientific packages for python (`http://www.numpy.org`, `http://www.scipy.org`, `http://matplotlib.org`).

  Installation

  - Linux: run command line

    `'sudo apt-get install python-numpy python-scipy python-matplotlib ipython ipython-notebook python-pandas python-sympy python-nose'`

- jupyter notebook: a open-source framework to share scientific code (`http://jupyter.org`)

  Installation:

  - Linux: run command line

    `'sudo pip install jupyter'`

- libhdf5-dev,cython,h5py a library to read/write hdf5 images (`https://www.hdfgroup.org/HDF5/`)

  Installation:

  - Linux: run the command lines for a global installation

    `'sudo apt-get install libhdf5-dev'`

    `'sudo pip install cython'`

    `'sudo pip install h5py'`

    or

    `'pip install --user cython'`

    `'pip install --user h5py'`

- pylibtiff a library to read/write tiff/tif images (`https://pypi.python.org/pypi/libtiff/`).

  Installation:

  - You can install the pylibtiff distributed with ASTEC

    `'cd path/to/Package/ASTEC/CommunFunctions/libtiff-0.4.0/; sudo python setup.py install'`

  - Alternatively:

    `'sudo pip install libtiff'`

    or

    `'pip install --user libtiff'`

1.1.4 the following libraries are necessary:

- `zlib`, a compression library (`http://www.zlib.net`)

  Installation (if not already installed)

  - Linux: run command line

    `'sudo apt-get install zlib1g-dev'`

- `libtiff`, `http://libtiff.org`

1.2 Installation

You need to clone the morpheme-privat project in your computer: run

‘cd <morpheme-privat_parent_directory>‘

‘git clone git+ssh://<username>@scm.gforge.inria.fr/gitroot/morpheme-privat/morpheme-privat.git‘

Please note that for this step, you may need to ask for an account on the inria gforge and for the access to the morpheme-privat project (for further information, contact gregoire.malandain@inria.fr)

Then you need to compile the dependencies LEMON, NIFTICLIB, TIFF in <morpheme-privat_parent_directory>/e by creating in each subrepository a build folder, going in it, and run:

‘ccmake ..‘

then configure, generate and quit (for NIFTICLIB, please take note that you should set the CMAKE_C_FLAGS to value '-fPIC' in the advanced mode ; you need therefore to press 't' to toogle advanced mode)

run ‘make -j<nb proc>‘ where <nb proc> is the number of processor require to compile the code Then go to <morpheme-privat_parent_directory>/vt directory and follow these instructions:

‘mkdir build‘

‘cd build‘

‘ccmake ..‘

Press 'c' to configure. At this step, set the links to the build folders of NIFTI, LEMON and TIFF. Press 'c' to re-configure. Press 'g' to generate and 'q' to quit.

Run ‘make -j<nb proc>‘ where <nb proc> is the number of processor require to compile the code

1.2.1 Previously

Since the vt library is compiled, you need to create the symbolic link of the folder <morpheme-privat_parent_directory>/ to ASTEC/CommunFunctions/cpp. in a terminal:

‘cd path/to/Package/ASTEC/CommunFunctions‘

ln -s <complete_path_of_vt_folder> cpp

1.2.2 From version v2.0

Since the vt library is compiled, you need to create the symbolic link of the folder <morpheme-privat_parent_directory>/ to ASTEC/CommunFunctions/cpp. in a terminal:

‘cd path/to/Package/ASTEC/CommunFunctions‘

ln -s <complete_path_of_bin_folder> cpp

1.3 Troubleshooting

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/experiment/
├── 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.

```
    ├── 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` indicate 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

```
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.

```
path/to/sample/
├── FUSE/
├── 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-1.py
```

`1.5-intraregistration-tutorial-parameters-1.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` indicate 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/sample/
└── 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` 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).

2.4 Starting with a toy embryo

Say we have an embryo named `160708-Tutorial` with four time points (these are the four first time points of the embryo `160708-Aquila-St8`). Thus at the beginning, we only have the raw data in the `<EMBRYO>=160708-Tutorial` file tree structure.

```
<EMBRYO>=160708-Tutorial
└── RAWDATA
    └── LC
        ├── Stack0000
        │   ├── Cam_Left_00001.zip
        │   ├── Cam_Left_00002.zip
        │   ├── Cam_Left_00003.zip
        │   └── Cam_Left_00004.zip
        └── Stack0001
            └── Cam_Left_00001.zip
```

```
        ├── Cam_Left_00002.zip
        ├── Cam_Left_00003.zip
        └── Cam_Left_00004.zip
    └── RC
        ├── Stack0000
        │   ├── Cam_Right_00001.zip
        │   ├── Cam_Right_00002.zip
        │   ├── Cam_Right_00003.zip
        │   └── Cam_Right_00004.zip
        └── Stack0001
            ├── Cam_Right_00001.zip
            ├── Cam_Right_00002.zip
            ├── Cam_Right_00003.zip
            └── Cam_Right_00004.zip
```

2.5 Fusing data

Let start by fusing data. I advised to create a `PARAMETERS` repository under the root `<EMBRYO>=160708-Tutorial` to store all the parameters file we will use. Then we copy there the `parameters.py` file and rename it as `parameters-fuse1.py`. I added `-fuse` to easily recognize it as a parameter file for fusing, and also add the numbering `1` to handle the case of several parameters files for fusing (if fusion parameters have to be changed for a few time points).

```
$ cd /path/to/160708-Tutorial
$ mkdir PARAMETERS
$ cp /path/to/astec-package/parameters.py PARAMETERS/parameters-fuse1.py
```

Then, this file `parameters-fuse1.py` is edited to be adapted to our experiment at hand. Here, the modification are as follows. Note that we only have the following changes with respect to the template version of the parameter file (see section **??**).

```
...
5 PATH_EMBRYO='/Users/greg/COLLABORATIONS/DIGEM/TUTO-ASTEC/160708-Tutorial'
...
9 EN='160708-Tutorial'
...
16 EXP_FUSE='EXP1'
...
44 end=4
...
58 raw_ori = 'right'
...
60 raw_resolution = (.21, .21, 1.)
...
68 raw_mirrors = True
...
```

We have then the following file tree.

```
<EMBRYO>=160708-Tutorial
├── RAWDATA
│   └── ...
└── PARAMETERS
```

```
   └── parameters-fuse1.py
```

Now we can run the fusion

```
$ cd /path/to/160708-Tutorial
$ 1-fuse.py -p PARAMETERS/parameters-fuse1.py
```

After completion of the fusing stage, the tree structure is

```
<EMBRYO>=160708-Tutorial
├── RAWDATA
│   └── ...
├── PARAMETERS
│   └── ...
└── FUSE
    └── FUSE_EXP1
        ├── 1-fuse.log
        ├── 160708-Tutorial_fuse_t001.inr
        ├── 160708-Tutorial_fuse_t002.inr
        ├── 160708-Tutorial_fuse_t003.inr
        ├── 160708-Tutorial_fuse_t004.inr
        └── parameters-fuse1.py
```

The folder `FUSE_EXP1` not only contains the fused images, but also a copy of the parameter file (hence the importance that all parameter files have different names) as well as a `log` file (`1-fuse.log`) that keeps trace of all the Astec launched commands. However, it won't keep trace of the other commands (e.g. the user erases a file), thus it is still important to keep all this elsewhere.

2.6 Segmenting the first time point

The segmentation of the first time point consists in labeling each and every cell of the embryo. By convention, the background has the 1 label (0 is used when label images are resampled). Thus cell labels are always larger or equal than 2.

Although any method can be used to obtain this segmentation, the script `2-mars.py` provide a means to use the `MARS` method [1].

Again the parameter file `parameters.py` is copied and renamed to eventually edit it.

```
$ cd /path/to/160708-Tutorial
$ cp /path/to/astec-package/parameters.py PARAMETERS/parameters-seg-first1.py
```

Then, this file `parameters-seg-first1.py` is edited. The changes with respect to the template version of the parameter file (see section **??**) are

```
...
 5 PATH_EMBRYO='/Users/greg/COLLABORATIONS/DIGEM/TUTO-ASTEC/160708-Tutorial'
...
 9 EN='160708-Tutorial'
...
16 EXP_FUSE='EXP1'
...
26 EXP_MARS='EXPMARS1'
...
```

Note that the line #16 indicates where to find the fused images. The segmentation is done through

```
$ cd /path/to/160708-Tutorial
$ 2-mars.py -p PARAMETERS/parameters-seg-first1.py
```

After completion of the segmentation stage, the tree structure is

```
<EMBRYO>=160708-Tutorial
├── RAWDATA
│   └── ...
├── PARAMETERS
│   └── ...
├── FUSE
│   └── FUSE_EXP1
│       └── ...
└── SEG
    └── SEG_EXPMARS1
        ├── 160708-Tutorial_mars_t001.inr
        ├── 2-mars.log
        └── parameters-seg-first1.py
```

Note the name of the result image with the `_mars` suffix. Such a naming convention is mandatory and has to be followed in case of segmentation by other means.

2.7 Correcting the first time point segmentation

As the sequence segmentation proceeds by propagation, the first segmentation must be cured. Under-segmentations may be corrected either by changing segmentation parameters and to relaunch the `2-mars.py` script or by manual edition (then be careful not to use an already used label). Over-segmentations can be automatically corrected (after identification). They have to be listed into a file whom lines contains two numbers, the labels to be merged (more precisely, only the second label will be kept) . E.g.

```
43 54
38 20
```

means that cells of labels 43 and 54 will be merged together (forming a cell of label 54), as well as cells of labels 38 and 20 (forming a cell of label 20). The file `/path/to/astec-package/mapping.txt` is a template for this merging.

Again, copy the parameter template file, as well as the merging template file (cells to be merged have to identified beforehand).

```
$ cd /path/to/160708-Tutorial
$ cp /path/to/astec-package/parameters.py PARAMETERS/parameters-seg-correction1.py
$ cp /path/to/astec-package/mapping.txt PARAMETERS/mancor_mapping_file.txt
```

and edit the two files. The second file contains only the two lines indicated above. The changes with respect to the template version of the parameter file are

```
...
 5 PATH_EMBRYO='/Users/greg/COLLABORATIONS/DIGEM/TUTO-ASTEC/160708-Tutorial'
...
 9 EN='160708-Tutorial'
...
16 EXP_FUSE='EXP1'
...
26 EXP_MARS='EXPMARS1'
...
28 EXP_SEG='EXPSEG1'
```

```
  ...
186 mancor_mapping_file='PARAMETERS/mancor_mapping_file.txt'
  ...
```

Note that the line #186 gives the (here relative) name of the file containing the cells to be merged. The `MARS` result is supposed to be found in the `SEG/EXPMARS1` folder (given by the variable `EXP_MARS`) while the correction result will be placed into the `SEG/EXPSEG1` folder (given by the variable `EXP_SEG`)

```
$ cd /path/to/160708-Tutorial
$ 3-manualcorrection.py -p PARAMETERS/parameters-seg-correction1.py
```

After completion of this stage, we have:

```
<EMBRYO>=160708-Tutorial
├── RAWDATA
│   └── ...
├── PARAMETERS
│   └── ...
├── FUSE
│   ├── FUSE_EXP1
│   │   └── ...
└── SEG
    ├── SEG_EXPMARS1
    │   └── ...
    └── SEG_EXPSEG1
        ├── 160708-Tutorial_seg_t001.inr
        ├── 3-manualcorrection.log
        ├── mancor_mapping_file.txt
        └── parameters-seg-correction1.py
```

Note the name of the result image with the `_seg` suffix.

2.8 Segmentation propagation

```
$ cd /path/to/160708-Tutorial
$ cp /path/to/astec-package/parameters.py PARAMETERS/parameters-astec.py
```

```
  ...
  5 PATH_EMBRYO='/Users/greg/COLLABORATIONS/DIGEM/TUTO-ASTEC/160708-Tutorial'
  ...
  9 EN='160708-Tutorial'
  ...
 16 EXP_FUSE='EXP1'
  ...
 28 EXP_SEG='EXPSEG1'
  ...
 44 end=4
  ...
```

```
$ cd /path/to/160708-Tutorial
$ 4-astec.py -p PARAMETERS/parameters-astec.py
```

Chapter 3

User guide: command line interfaces

The Astec distribution contains 4 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 of the 'right' 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.
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` 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

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°).
- `raw_mirrors` is a parameter indicating whether the right camera images have to be mirrored or not. Its value is either `False` or `True`.
- `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.

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_STACK` `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
    │   └── ...
    └── stack_1_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-directories `FUSE/FUSE_XXX` under the `FUSE/` directory

```
/path/to/experiment/
└── RAWDATA/
```

```
   ├── ...
└── FUSE/
    ├── FUSE_RELEASE/
    │   └── ...
    └── FUSE_XXX/
        └── ...
```

'XXX' is given by 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_txxx.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 Tuning 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 [2] 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 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 sequence intra-registration is made of the following steps.

1. Co-registration of every couple of successive fused images.
2. Composition of the co-registration transformations with respect to a reference fused image.
3. Building a template image that will

Its results will be stored in the INTRAREG/INTRAREG_<EXP_INTRAREG> subdirectory

where <EXP> is set by EXP_INTRAREG from the parameter file. POST and SEG directories are created only if respectively segmentation and post-segmentation sequences are required to be resampled/reconstructed in the common frame.

<EMBRYO>
└— ...

```
├── INTRAREG
│   └── INTRAREG_<EXP>
│       ├── CO-TRSFS
│       ├── FUSE
│       ├── LOGS
│       ├── MOVIES
│       ├── POST
│       ├── SEG
│       └── TRSFS_t<F>-<L>
└── ...
```

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

1. Co-registration of pairs of successive fused images (section 3.2.0.1). 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 3.2.0.2). 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 3.2.0.3). 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 3.2.0.4). 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.

3.2.0.1 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
```

3.2.0.2 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 `intra_registration_resolution` allows to decrease the resampled images volume (the default resolution i.e. voxel size, given by `target_resolution`, for the fusion step (see section **??**), is set to 0.3, while it is set to 0.6 here).

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).

3.2.0.3 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
```

3.2.0.4 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_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` 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.

From version 2.0, the data architecture should be as follows

```
<EMBRYO>
├── RAWDATA
│   ├── LC
│   │   ├── Stack0000
│   │   └── Stack0001
│   └── RC
│       ├── Stack0000
│       └── Stack0001
├── FUSE
│   ├── FUSE_<EXP_1>
│   ├── FUSE_<EXP_2>
│   ├── ...
│   └── FUSE_RELEASE
├── SEG
│   ├── SEG_<EXP>
│   ├── ...
│   └── SEG_RELEASE
└── POST
    ├── POST_<EXP>
    ├── ...
    └── POST_RELEASE
```

`<SOMETHING>_RELEASE` sub-directories should content the "last version" validated by an expert. `<SOMETHING>_<EXP_x>` are folders containing the different experiments conducted at one given step (with different parameters/methods). After inspection of the results, one of them is supposed to be renamed as `<SOMETHING>_RELEASE` and the others can be deleted.

Particular files:

`nomenclature.py`: file fixing the set of naming rules in working directories

- this file should not be modified.

`parameters.py`: file defining the set of parameters useful for all the process steps (parameters are all prefixed with respect to the step they are used in).

- it is a "template" file which should be duplicated and whose copy can be modified by the user to its convenience (only the parameter values should be modified, not their name...).

The scripts of steps `1-fuse.py`, `2-mars.py`, `3-manualcorrection.py`, `4-astec.py`, `5-postcorrection.py` are executables, so that each astec step calling can be made as described here: For example, in order to launch the fusion step on an embryo called `"171107-Karine-St8"` one should:

1. Duplicate the file `<astec-package>/parameters.py` and rename it as a new file `<arbitrary-path>/parameters_ka`
2. Edit this new file `<arbitrary-path>/parameters_karine.py`to specify the desired value of each parameter related to the fusion step
3. In a terminal,

   ```
   $ cd <astec-package> # in order to be in the astec directory
   (/media/DATA/Codes/astec-package on Hermione)
   $./1-fuse.py --parameters <arbitrary-path>/parameters_karine.py
   --embryo-rep /media/DATA/171107-Karine-St8/
   ```

   (or equivalently,a shorter format)

   ```
   $./1-fuse.py -p <arbitrary-path>/parameters_karine.py -e
   /media/DATA/171107-Karine-St8/
   ```

At each Astec step execution, a copy of the parameters file as well as a log file are automatically generated in the target working directory.

For each Astec step, it is possible to display the help relative to the corresponding script by launching the script with the option `'--help'`. For example, for the fusion step:

In a terminal, launch the command line:

```
$ <astec-package>/1-fuse.py --help
```

The terminal displays the following message:

```
Usage:  1-fuse.py [options]
Options:
-h, --help show this help message and exit
-p FILE, --parameters=FILE
python file containing parameters definition
-e PATH, --embryo-rep=PATH
path to the embryo data
-q, --quiet don't print status messages to stdout
```

3.3 List of command line interfaces

3.3.1 1.5-intraregistration-GC.py

3.3.2 2-mars.py

3.3.3 3-manualcorrection.py

3.3.4 4-astec.py

3.3.5 5-postcorrection.py

3.3.6 6-named.py

3.3.7 7-virtualembryo.py

Bibliography

[1] 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.

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