Registration toolbox Manual

Léo Guignard

November 4, 2022

Contents

Chapter 1

Notes on this registration algorithm

1.1 Purpose

This repository has for purpose to wrap the set of tools provided by the `'blockmatching'` algorithm described in [Ourselin et al., 2000] and used in [McDole et al., 2018; Guignard et al., 2017, 2014].

Provided the `'blockmatching'` library installed (see here) this wrapper allows to easily apply it to register in time or in space 3D+t movies.

1.2 Notations

In this manual (as for the manual dedicated to the `'blockmatching'` algorithm and as often in the literature), we define an image as a function that maps nD coordinates (in our case 3D) onto a predefined subset (most likely finite) of $\mathbb{R}$ or $\mathbb{N}$, the dynamic range of the image.

Also, we refer to image transformations as functions that map 3D coordinates in a given space S_1 to 3D coordinates of a different space S_2. Such a transformation is named $T_{S_2 \leftarrow S_1}$. Then, $T_{S_1 \leftarrow S_2} = T^{-1}_{S_2 \leftarrow S_1}$ is the transformation that maps the coordinates of the space S_2 onto the space S_1 and allows to register an image I_1 in S_1 into the space S_2 of an image I_2, building a new, registered, image $I_{1\rightarrow 2}$:

$$\underbrace{I_{1\rightarrow 2}}_{I_1 \text{ in } I_2 \text{ space}} = \underbrace{I_1}_{I_1 \text{ in its original space}} \circ \underbrace{T_{S_1 \leftarrow S_2}}_{\text{trsf from } I_2 \text{ space to } I_1 \text{ space}}$$

1.3 Provided transformations

In these scripts, let assume one wants to register a floating image I_{flo} onto a reference image I_{ref} to then create the registered floating image into the reference space (the image $I_{flo\rightarrow ref}$). Then the first output transformation provided is $T_{flo\leftarrow ref}$, sometimes to help the communication with other softwares (such as BigDataViewer in Fiji for example), the inverse of the original transformation is provided. It is important to remember that the output transformation provided by these scripts is almost always (a few counter examples are specified later) in the units specified by the voxel size (and it is assumed that the unit provided is similar across all the images to register). This allows to simplify potential problems of difference of voxel size between the different images to co-register.

The fact that the transformations are provided in the unit specified by the user means that these scripts only compute the transformations from a given space to an other space. It does not take care of any potentially desired scaling. One can apply a scaling when specifying how to apply the transformations that have been computed (see later in this manual).

1.4 Talking to BigDataViewer et al. (obsolete, maybe in a future version)

The scripts provided here does not include any visualization means beside rewriting each images after transformation. While this can be useful to then build movies of maximum intensity projections or volumic rendering for example, it is sometimes better not to apply the transformations to the images. In that case, one can input

the calculated transformations to the Fiji plugin BigDataViewer which is a great way to have a look at 3D+t data. BigDataViewer is also an entry point to other Fiji plugin such as MaMuT which allows you to manually and semi-automatically detect and track cells. Having registered data in time, in that context can be very useful.

In order to ease the transition from the output of these scripts to the input of BigDataViewer the scripts provided here can output an xml file that should be readable by BigDataViewer.

This feature is somewhat experimental and I do **not** guaranty that it will always work as intended. That being said it is good to be aware of what BigDataViewer expect, and what can be done to meet these expectations:

- The transformations have to be from the floating frame to the reference frame (as opposed to the original output of this script)
- The transformations have to be in isotropic voxel unit (as opposed to physical unit like μm)
- The transformations have to go from the anisotropic voxel floating space to the isotropic reference space (meaning that it has to encompass the resampling)
- Since the transformations **have to** be in voxel unit (it is especially important when using plugins such as Multiview Reconstruction), the voxel physical size can be specified (though it will only be used to put in the meta data of the output image).

Chapter 2

Installation

2.1 Requirements and installation (tested on Linux)

We strongly advice to install this code in a separate environment using conda for example.

Once conda is installed, one can proceed the following way:

```
$ conda create -n registration python=3.10
$ conda activate registration
$ conda install vt -c morpheme
$ git clone https://github.com/guignardlab/registration-tools.git
$ cd registration-tools
$ pip install .
```

To test whether your installation was successful or not, you can run the following commands:

```
$ pip install '.[testing]'
$ pytest
```

You should maybe get warnings but no error.

2.2 OLD, KEPT for posterity ... Requirements and installation (tested on Linux)

To install `'blockmatching'`, it is necessary to have cmake, a c/c++ compiler and zlib installed:

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

Here are the steps to install `'blockmatching'`: First one have to install the external libraries. To do so, one can run the following commands in a terminal from the BlockMatching folder:

To install klb read/write tools:

```
$ cd external/KLBFILE/
$ mkdir build
$ cd build
$ cmake ../keller-lab-block-filetype
$ make -j<desired number of cores>
```

To install tiff read/write tools, from the folder BlockMatching:

```
$ cd external/TIFF
$ mkdir build
$ cd build
$ cmake ../tiff-4.0.6
$ make -j<desired number of cores>
```

Once these are installed one can run the following commands in a terminal from the BlockMatching folder:

```
$ mkdir build
$ cd build
$ ccmake ..
```

press c then e
Then enter the correct absolute paths for the tiff and klb builds (ie '/path/to/BlockMatching/external/TIFF/build and /path/to/BlockMatching/external/KLBFILE/build').
Then c then e then g

```
$ make -j<desired number of cores>
```

Then this newly built binaries (found in BlockMatching/build/bin) have to be accessible from the different scripts that will be ran. To do so one can add the following command to their /.bashrc (/.profile for mac users):

```
export PATH=$PATH:/path/to/BlockMatching/build/bin
```

One 'direct' way to do so is to run the following command:

```
echo 'export PATH=$PATH:/path/to/BlockMatching/build/bin' >>  /.bashrc
```

or add a line to the json configuration file to specify the path to the binaries (see below).

Then, for this wrapper to it work it is first necessary to have installed:

- Python 2.7.x (https://www.python.org/download/releases/2.7/)

Some Python libraries are also required:

- Scipy (https://www.scipy.org/)
- Numpy (https://numpy.org/)
- IO (https://github.com/leoguignard/IO)
  - h5py (https://pypi.python.org/pypi/h5py)
  - pylibtiff (https://github.com/pearu/pylibtiff)
- pyklb (https://github.com/bhoeckendorf/pyklb)
- statsmodels (https://www.statsmodels.org)

Once everything is installed the python scripts are ready to run.

Chapter 3

Time Registration

3.1 What time registration can do for you.

This wrapper allows to do an intra-registration of a time-series in order to stabilize it in time. Intra stabilization is important for time-series in order to ease the extraction of quantitative features. Such features could be cell detection and tracking for images of nuclei where the tracking is harder is the image has artificial movement or the extraction of $\Delta F/F$ for functional imaging where, for the analysis, it is often assumed that that each voxel contains the same part of the brain throughout the movie. But it is mostly important to have nicer movies for presentations!

3.1.1 Transformation types

For the computation of the transformations different degrees or freedom are available, namely 3D translation, rigid (rotation + translation), affine (rigid + scaling and shearing) and fully non linear. Of course each mode has its usage (see Tab. 3.1).

3.1.2 Alignment scheme

The transformations to stabilize a movie can be computed in multiple ways. The two that can be used with this wrapper are 1) by registering every single time point directly onto the reference time point and 2) by registering consecutive time points onto each others and build the composition of the transformation (see Tab. 3.2).

Moreover, when computing translations one can decide to only compute a subset of the time points and interpolate the translations in-between using a spline interpolation.

3.2 Running a time registration

Once installed, one can run a time registration either by running the script `'time_registration.py'` from a terminal or by using the `'TimeRegistration'` Python class.

3.2.1 Running from the terminal as a script

To run the script `'time_registration.py'` directly from the terminal, one can run the following command:

```
$ time_registration.py
```

The script then prompt the user to inform a path to the json configuration files or to a folder containing at least one json configuration file. Some examples are provided in the folder `'json-examples'`.
If the configuration file is correctly filled, the script will then compute the registrations and output the registered images (according to what was specified by the user in the json file).
In order to skip entering the path to the configuration file(s) one can also run the script as previously appending the name of the configuration file to the command:

```
$ time_registration.py /path/to/configuration/file.json
```

[1] specific to this wrapper
[2] for this wrapper, only if the transformation is a translation

Trans-formation type	Conserve original geometry	Potential quality loss when anysotropic image	Slow/Fast	Degrees of freedom	Best suited for
Translation	Yes	No	Really Fast	3	Really small drift over long period of time (Functional imaging)
Rigid	Yes	Yes	Fast	6	Large movements over long period of development (Embryo development)
Affine	No	Yes	Slowish	9	???
Non-linear	No	Yes	Really slow	#voxels of the image	Correcting for punctual deformations (twitching embryo)

Table 3.1: List of transformation types with their properties

Registration scheme	Independent between time points	Handle large sample dis-placement	Propagate errors	Interpolation	Has to compute composi-tion	Best suited for
t_n onto $t_{n\pm1}$	✗	✓	✓	✗[1]	✓	Not fixed samples for extended period of time (Embryo development)
t_n onto t_{ref}	✓	✗	✗	✓[2]	✗	Fixed samples (Functional imaging)

Table 3.2: List of schemes with their pros and cons

3.2.2 Running from Python as a class

One can run a time registration directly from Python (in a notebook for example). It can be done the following way:

```
from registrationtools import TimeRegistration
tr = TimeRegistration('path/to/param.json')
tr.run_trsf()
```

or

```
from registrationtools import TimeRegistration
tr = TimeRegistration('path/to/json/folder')
tr.run_trsf()
```

or

```
from registrationtools import TimeRegistration
tr = TimeRegistration()
tr.run_trsf()
```

In the last case, a path will be asked to the user.

3.3 Description of the configuration file

Everything that the script will do is determined by the configuration file. The script being somewhat flexible, the configuration file has a significant number of possible parameters. Some of them are required, others have default values and some are optional depending on the chosen options. The parameter files are written in json (`https://www.json.org/`), please follow the standard format so your configuration files can be read correctly.

3.3.1 List of all parameters

Here is an exhaustive list of all the possible parameters:
File path format:

- `path_to_data` `str`, mandatory, common path to the image data
- `file_name`, `str`): mandatory, image name pattern
- `trsf_folder`, `str`, mandatory, output path for the transformations
- `output_format`, `str`, mandatory, how to write the ouput image. Can be a folder, in that case the original image name, `'file_name'` is kept. Can be a file name, in that case it will be written in the original folder `'path_to_data'`. Can be a folder plus name pattern. Can be a short string (suffix), in that case the string will be appended to the original name, before the file extension.
- `projection_path`, `str`, default value: `None`, path to the output folder for the maximum intensity projection images. If not specified, the projection will not be performed.
- `check_TP`, `[0 | 1]`, default value: `0`, if `1` check if all the required images exist.
- `path_to_bin`, `str`, default value: `""`, path to the blockmatching binaries, not necessary if that path is specified in your $PATH

Image information:

- `voxel_size`, [`float`, `float`, `float`], mandatory, voxel size of the image (can simply be the aspect ratio but real size should be considered)
- `first`, `int`, mandatory, first time point of the sequence to register
- `last`, `int`, mandatory, last time point of the sequence to register
- `not_to_do`, [`int`, ...], default value: `[]`, list of time points to not process (they will be totally ignored from the process)

Registration parameters:

- `compute_trsf`, `[0 | 1]`, default value: `1`, whether or not computing the transformations (one might not want to do it if the transformations have already been computed for a different channel)

- `sequential`, `[0 | 1]`, default value `1`, whether or not performing the spatial registration sequentially (t_n onto $t_{n\pm1}$)
- `ref_TP`, `int`, mandatory, time point of the reference image onto which the registration will be made.
- `ref_path`, `str`, default value: `None`, path to the reference image, if specified, every image will be directly registered to that particular image
- `trsf_type`, `str`, default value: `"rigid"`, choose between `"translation"`, `"rigid"`, `"affine"`, `"vectorfield"`
- `registration_depth`, `int`, default value: `3`, Maximum size of the blocks for the registration algorithm (min `0`, max `5`), the lower the value is the more the registration will take into account local details but also the more it will take time to process
- `padding`, `[0 | 1]`, default value: `1`, put to `1` if you want the resulting image bounding boxes to be padded to the union of the transformed bounding boxes. Otherwise (value at `0`) all the images will be written in the reference image bounding box
- `recompute`, `[0 | 1]`, default value: `1`, if `1`, it forces to recompute the transformations even though they already exist
- `lowess`, `[0 | 1]`, default value: `0`, if `1`, smoothes the transformations, so far only works with `"translations"`
- `window_size`, `int`, default value: `5`, size of the window for the lowess interpolation. To be used `lowess` has to be set to `1`
- `trsf_interpolation`, `[0 | 1]`, default value: `0`, if `1`, interpolate between transformations using univariate spline interpolation, so far only works with `"translations"`
- `step_size`, `int`, default value: `100`, if it is not necessary to compute all the time points, this is the step size between the computed times. To be used `trsf_interpolation` has to be `1`
- `spline`, $1 \geq$ `int` ≥ 5, default value: `1`, degree of the smoothing spline, 1 is piecewise linear interpolation.
- `pre_2D`, `[0 | 1]`, default value `0`, whether or not (if `1` it will do it) to do a pre alignment in 2D. For some more complex datasets it can help.
- `low_th`, `float`, default value `0`, an intensity threshold bellow which voxels will not be considered. If left at `0`, no threshold is applied.
- `sigma`, `float`, default value: `2.0`, smoothing parameter for the non-linear registration
- `keep_vectorfield`, `[0 | 1]`, default value: `0`, if `1`, will write the transformation vector field (it will be large!)

Apply registration parameter:

- `apply_trsf`, `[0 | 1]`, default value: `1`, if `1`, will apply the computed transformations to the images.
- `image_interpolation`, `str`, default value: `"linear"`, choose between `"nearest"` (for label images), `"linear"` and `"cspline"` (the `"cspline"` interpolation has undesirable border effects when the signal is too low)

BigDataViewer Output:

- `do_bdv`, `[1 | 0]`, default `0`, whether or not to output the BigDataViewer xml file
- `out_bdv`, `string`, default `trsf_paths` [0] + `bdv.xml`, path and name of the output BigDataViewer xml
- `bdv_voxel_size`, [`float`, ...], default `ref_voxel`, physical voxel size to specify to the BigDataViewer xml, Warning: it will **not** be use for any of the computation by this script, only be written in the BigDataViewer file
- `bdv_unit`, `string`, default `microns`, physical unit of `bdv_voxel_size`
- `time_tag`, `string`, default `TM`, string pattern that is before the information of time in the image data
- `ref_im_size`, [`float`, ...], default `flo_im_sizes`[0], size of the image if different from the floating images (only useful in the case for creating the BigDataViewer output).

- `bdv_im`, [`string`, ...], default [`path_to_data + ref_im`] + (`path_to_data + flo_im`), absolute path to the images, useful when running this script on a different file system than the one that will be used to run BigDataViewer plugin (ie running the script on Linux and running BigDataViewer on Windows).

As a general point, the format for specifying time is in the "Pythonic" way of the function `str.format()` with `t` being the time argument. For example, if you expect your time being on 6 digits with 0s to fill the corresponding format is {`t:06d`}, `t` is the name of the field, `0` is the filling value, `6` is the number of digits and `d` stands for digit (therefore integer). If the time is present multiple times in the image name and or path to the image, the pattern can be entered multiple times. For example, if your path to an image is similar to: `'/path/to/image/T000493/Im_C0_t000493.tif'` then the correct values for `'path_to_data'` is `'/path/to/image/T{t:06d}/'` and for `'file_name'` is `'Im_C0_t{t:06d}.tif'`.

3.4 Concrete examples

This section exhibits concrete examples of problems and how to build a parameter file to solve it.

3.4.1 Long movies where the sample is not fixed

A typical example is movies where the sample is somewhat free to move in front of the camera. It can happen when the part of the sample that is held in the mounting system is not rigid to the part of the sample to image (for example the cone of the mouse embryo after implantation, see [McDole et al., 2018]). It can also be the case when there is no possibility to hold the sample, which is then dropped and free to move (see [Guignard et al., 2017]). Then the desired main parameters are the following:

- Transformation type: Rigid (keeping in mind that resolution loss might happen)
  `"trsf_type":"rigid"`

- Computation Scheme: t_n onto $t_{n\pm1}$
  `"sequential":1`

- Padding: yes
  `"padding":1`

- Undersampling time: no
  `"trsf_interpolation":0`

- Lowess interpolation: no
  `"lowess":0`

3.4.2 Long movies where the sample is fixed

When the sample can be held better, then a rigid transformation might be to many degrees of freedom. More over, when the images have an anysotropic voxel size then the rotation part of the rigid transformation might orient the voxel in a way that result in the lost of some of its information. It might then be better to only allow for a translation. A typical example for such setup would be the imaging if the *Drosophila* embryo. In that case the main parameters are the following:

- Transformation type: Translation
  `"trsf_type":"translation"`

- Computation Scheme: t_n onto $t_{n\pm1}$ or t_n onto t_{ref}
  `"sequential":1` or
  `"sequential":0`

- Padding: yes
  `"padding":1`

- Undersampling time: no
  `"trsf_interpolation":0`

- Lowess interpolation: no
  `"lowess":0`

3.4.3 Extremely large movie where the sample slowly drifts over time

If the sample is imaged often enough compared to its drift from the camera, then the difference between consecutive time points might not be large enough to be captured by our registration algorithm, it is then necessary to register every time point directly to the common reference frame. More over, if these movies have an extremely large number of time points and that the displacement is a smooth, slow drift, it might be useful to only compute the transformations for a subset of the time points and then to a Lowess interpolation between these time points. A typical example for such a case would be high frequency functional imaging of the nervous system explant of a *Drosophila* larva [Chen's paper]. The desired main parameters are then:

- Transformation type: Translation
  `"trsf_type":"translation"`
- Computation Scheme: t_n onto t_{ref}
  `"sequential":0`
- Padding: no
  `"padding":0`
- Undersampling time: yes
  `"trsf_interpolation":1`
- Lowess interpolation: depends
  `"lowess":0` or `"lowess":1`

3.4.4 Correcting for non linear biological movements

In some extreme cases, the sample is not immobile in front of the camera creating non-linear distortions of the sample that can prevent the identification and tracking of objects in this very sample. An example for such a case is the imaging of the spinal chord of a non paralyzed Zebrafish, for these movies, the muscles start to have some activity that then deform in a non-linear manner the spinal chord. In order to look at calcium activity in cells, regardless of these muscle contractions, it is necessary to deform each time points non-linearly so they match a chosen reference point [Yinan's paper]. In that case the main parameters are the following:

- Transformation type: Non linear
  `"trsf_type":"vectorfield"`
- Computation Scheme: t_n onto t_{ref}
  `"sequential":0`
- Padding: no
  `"padding":0`
- Undersampling time: no
  `"trsf_interpolation":0`
- Lowess interpolation: no
  `"lowess":0`

3.4.5 Some special scenarios that could happen

Computing the transformations on one channel and applying them onto a second one.

Sometimes, the focus within the sample is towards a specific subset of this sample. If this subset has independent movement from the embryo itself, it can perturb the registration protocol to the point of giving erroneous results. Such examples could be imaging gene expression dynamic patterning or imaging blinking labeling strategies. To circumvent this issue we advise, when possible, on using a second channel where more consistent labeling methods (to the global embryo) is used. An obvious example of such a labeling method would be ubiquitous labeling of the nuclei. If it is possible, then the dataset consists in two or more channels. Let $C1$ be the ubiquitous channel and $C2$ the relevant channel for this particular experiment. To do so it is necessary to sequentially run the algorithm two times, the first time to compute the transformation and the second time to apply the transformations.
Computing the transformations:

- Path to the image: path to channel $C1$
  `"path_to_data":"/path/to/channel1/folder/"`
  `"file_name":"channel-`$C1$`-image-pattern-T{t:03d}.tif"`

- : Compute the transformation: yes
  `"compute_trsf":1`
- Apply the transformation: no
  `"apply_trsf":0`

Applying the transformations:

- Path to the image: path to channel $C2$
  `"path_to_data":"/path/to/channel2/folder/"`
  `"file_name":"channel-`$C2$`-image-pattern-T{t:03d}.tif"`
- Compute the transformation: no
  `"compute_trsf":0`
- Apply the transformation: yes
  `"apply_trsf":1`

[...]

Chapter 4

Spatial registration

4.1 What spatial registration can do for you

Given n images, provided a reference image (I_0), this wrapper allows you to compute the set of transformations that register the $n-1$ images onto the reference image I_0. This is mainly useful when multiple angles of the same sample have been acquired that then need to be co-registered together to ultimately be potentially deconvolved and fused. Warning: This wrapper does not allow you to fuse the different angles together!!

4.1.1 Transformation types

As for the time registration you can choose to allow increasing degrees of freedom starting with translation to fully non linear. In the case of the spatial registration translation and rigid transformations are usually not enough due to potential small alignment of the optical system creating aberrations that require higher number of degrees of freedom to correct.

4.1.2 Alignment scheme

While multiple alignment can be developed and built, this wrapper only allows you to use one scheme, the simplest one: every image is registered onto a provided reference image using potential initial transformations.

4.2 Running a spatial registration

Once installed, one can run a time registration either by running the script `'spatial_registration.py'` from a terminal or by using the `'SpatialRegistration'` Python class.

4.2.1 Running from the terminal as a script

To run the script `'spatial_registration.py'` directly from the terminal, one can run the following command:

```
$ spatial_registration.py
```

The script then prompt the user to inform a path to the json configuration files or to a folder containing at least one json configuration file. Some examples are provided in the folder `'json-examples'`.
If the configuration file is correctly filled, the script will then compute the registrations and output the registered images (according to what was specified by the user in the json file).
In order to skip entering the path to the configuration file(s) one can also run the script as previously appending the name of the configuration file to the command:

```
$ spatial_registration.py /path/to/configuration/file.json
```

4.2.2 Running from Python as a class

One can run a time registration directly from Python (in a notebook for example). It can be done the following way:

```
from registrationtools import SpatialRegistration
tr = SpatialRegistration('path/to/param.json')
tr.run_trsf()
```

or
```
from registrationtools import SpatialRegistration
tr = SpatialRegistration('path/to/json/folder')
tr.run_trsf()
```
or
```
from registrationtools import SpatialRegistration
tr = SpatialRegistration()
tr.run_trsf()
```
In the last case, a path will be asked to the user.

4.3 Description of the configuration file

Everything that the script will do is determined by the configuration file. The script being somewhat flexible, the configuration file has a significant number of possible parameters. Some of them are required, others have default values and some are optional depending on the chosen options. The parameter files are written in json (`https://www.json.org/`), please follow the standard format so your configuration files can be read correctly.

4.3.1 List of all parameters

Here is an exhaustive list of all the possible parameters:
File path format:

- `path_to_data`, `str`, mandatory, common path to the image data
- `ref_im`, `str`, mandatory, name of the reference image (`path_to_data` + `ref_im` should be valid)
- `flo_ims`, [`str`, ...], mandatory, list of names to the floating images, the order given in that list has to be conserved throughout the configuration file.

Transformation computation:

- `compute_trsf`, `[1 | 0]`, default `1`, whether or not to compute the transformations, 1 yes.
- `trsf_paths`, `str`, mandatory, paths to the output transformation folders (for each of the angles to register). If only a folder is provided the default transformation names will be `'Aa:s-trsf:s.trsf'`, the `a` parameter is for the angle number (0 is the reference, 1 the first angle and so on and so forth). The `trsf` parameter is for the transformation type, if not provided only the last transformation will be kept in the case of multiple transformations are asked.
- `init_trsf`, [`str`, ...] or [`trsf description`, ...], when `str` it is the path to 4x4 matrices to use as initial transformations, last line should be $[0, 0, 0, 1]$, last column $[T_x, T_y, T_z, 1]^T$. It is possible to specify the initial transformations manually, `trsf description` is a list, with 2 or 3 elements, the first element is the type of transformation it can be `flip` for a flip of the image or `rot` for a rotation of the image. If `flip` is the first element then only a second element is expect and it is the flip axis, it can be `X`, `Y` or `Z`. If `rot` if the first element then two other elements are expected the rotation axis (`X, Y, Z`) and then the rotation angle in degrees. If multiple transformations are necessary (flip plus rotation) these transformations can be chained. Warning: it is recommended to use that way to specify the initial transformation to avoid mismatch between what is expected by the script and what is given in your text file.
- `test_init`, `[1 | 0]`, whether or not to test the initial transformations. When on, it is the only thing the script does. It is highly recommended to check if the initial transformation is correct before trying to compute any other transformation.
- `trsf_type`, [`str`, ...], mandatory if `compute_trsf` is at 1, list of transformations to compute (to choose from `translation, rigid, affine, vectorfield`. The first transformation will be computed first and so on and so forth. The order is therefore important, while there is no actual constraint on it, it does not serve any purpose to compute higher degrees of freedom first.
- `ref_voxel`, [`float`, `float`, `float`], mandatory, x, y, z voxel size of the reference image
- `flo_voxels`, [[`float`, `float`, `float`], ...], mandatory, x, y, z voxel sizes of the floating images
- `flo_im_sizes`, [[`int`, `int`, `int`], ...], mandatory if describing the initial transformations (as opposed to providing text files), x, y, z number of voxels of the floating images.

- `registration_depth`, `int`, default value: `3`, Maximum size of the blocks for the registration algorithm (min `0`, max `5`), the lower the value is the more the registration will take into account local details but also the more it will take time to process

Applying the transformations:

- `apply_trsf`, `[1 | 0]`, default `1`, whether or not to apply the transformations, 1 yes.
- `out_voxel`, [`float`, `float`, `float`], voxel size of the registered images. If not specified `ref_voxel` is taken.
- `out_pattern`, `str`, mandatory, how to write the ouput images. Can be a folder, in that case the original image name, `'file_name'` is kept. Can be a short string (no `"/"` in it) (suffix), in that case the string will be appended to the original name, before the file extension.
- `image_interpolation`, `str`, default value: `"linear"`, choose between `"nearest"` (for label images), `"linear"` and `"cspline"` (the `"cspline"` interpolation has undesirable border effects when the signal is too low)
- `begin`, `int`, if the transformations should be applied to multiple time points one can specify the position of time information in `ref_im` and `flo_im` using the `.format` syntax (for example {`t:06d`}. Then this parameter `begin` will be the first time point to be processed
- `end`, `int`, see `begin`, this is the last time point to be processed.
- `copy_ref`, `[ 1 | 0 ]`, default `0`, only useful when the output image has the same voxel size as the input image. In that case, it is possible to decide whether or not to copy the reference image. This can be useful to keep the same file name pattern across all the angles.

BigDataViewer Output:

- `do_bdv`, `[1 | 0]`, default `0`, whether or not to output the BigDataViewer xml file
- `out_bdv`, `string`, default `trsf_paths`[0] + `bdv.xml`, path and name of the output BigDataViewer xml
- `bdv_voxel_size`, [`float`, ...], default `ref_voxel`, physical voxel size to specify to the BigDataViewer xml, Warning: it will **not** be use for any of the computation by this script, only be written in the BigDataViewer file
- `bdv_unit`, `string`, default `microns`, physical unit of `bdv_voxel_size`
- `time_tag`, `string`, default `TM`, string pattern that is before the information of time in the image data
- `ref_im_size`, [`float`, ...], default `flo_im_sizes`[0], size of the image if different from the floating images (only useful in the case for creating the BigDataViewer output).
- `bdv_im`, [`string`, ...], default [`path_to_data + ref_im`] + (`path_to_data + flo_im`), absolute path to the images, useful when running this script on a different file system than the one that will be used to run BigDataViewer plugin (ie running the script on Linux and running BigDataViewer on Windows).

4.4 Notes on the produced outputs

It is important to remember that the produced outputs (especially the transformations) are written in the context of that script, meaning that if one want to use the transformations in an external context some other information are necessary.

First of all, the provided transformations are the composition of the initial transformations and all the intermediate transformations, meaning that it allows to register I_{flo} onto I_{ref}. The transformations unit is the unit min(`ref_voxel`) and do not include any re-sampling information (ie it always will be isometric units).

Second, the provided transformations go from the reference space into the floating space, allowing to rewrite the floating image into the reference frame. In some applications the inverse transformation is expected and therefore this script output the inverse transformations.

An example, if you would like to use these transformations in BigDataViewer you need to first provide the voxel ratio for the voxel size (and not the actual μm value), to use the inverse transformations and to apply the scaling matrix.

Bibliography

Guignard, L., Fiuza, U.-M., Leggio, B., Faure, E., Laussu, J., Hufnagel, L., Malandain, G., Godin, C., and Lemaire, P. (2017). Contact-dependent cell communications drive morphological invariance during ascidian embryogenesis.

Guignard, L., Godin, C., Fiuza, U.-M., Hufnagel, L., Lemaire, P., and Malandain, G. (2014). Spatio-temporal registration of embryo images. In *ISBI - International Symposium on Biomedical Imaging*, Pekin, Chine. IEEE.

McDole, K., Guignard, L., Amat, F., Berger, A., Malandain, G., Royer, L. A., Turaga, S. C., Branson, K., and Keller, P. J. (2018). In toto imaging and reconstruction of post-implantation mouse development at the single-cell level. *Cell*, 175(3):859–876.

Ourselin, S., Roche, A., Prima, S., and Ayache, N. (2000). Block matching: A general framework to improve robustness of rigid registration of medical images. In *MICCAI'00*, volume 1935 of *LNCS*, pages 557–566. Springer.