Time registration Manual

Léo Guignard

September 17, 2019

Contents

Transformation 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 1: List of transformation types with their properties

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 3D movies.

2 What this wrapper 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!

2.1 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. 1).

2.2 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. 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 lowess interpolation [Cleveland and Loader, 1996].

Registration scheme	Independent between time points	Handle large sample displacement	Propagate errors	Interpolation	Has to compute composition	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 2: List of schemes with their pros and cons

3 Requirements and installation, (tested on Linux, Ubuntu 18.04)

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):

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

```
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 `'time-registration.py'` is ready to run.

4 Running `'time-registration.py'`

To run the script `'time-registration.py'` one can run the following command from the folder containing it:

```
$ python 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 specified 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:

```
$ python time-registration.py /path/to/configuration/file.json
```

5 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.

5.1 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` default value: `None`, path to 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.
- `suffix`, `str`, default value: `None`, suffix to add to the output image if output_format is not specified

- `projection_path`, `str`, default value: `None`, path to the output folder for the maximum intensity projection images. If not specified it will be written like the output format with `-1` for the time.
- `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`, wheter or not computing the transformations (one might not want to do it if the transformations have already been computed for a different channel)
- `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
- `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)
- `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_interpolation`, `[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_interpolation` has to be `1`
- `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 `lowess_interpolation` has to be `1`
- `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.

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_CO_t000493.tif'` then the correct values for `'path_to_data'` is `'/path/to/image/T{t:06d}/'` and for `'file_name'` is `'Im_CO_t{t:06d}.tif'`.

6 Concrete examples

This section exhibits concrete examples of problems and how to build a parameter file to solve it (see the summary Tab).

6.1 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)
- Computation Scheme: t_n onto $t_{n\pm1}$
- Padding: yes
- Undersampling time with Lowess interpolation: no

6.2 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
- Computation Scheme: t_n onto $t_{n\pm1}$ or t_n onto t_{ref}
- Padding: yes
- Undersampling time with Lowess interpolation: no

6.3 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
- Computation Scheme: t_n onto t_{ref}
- Padding: no
- Undersampling time with Lowess interpolation: yes

6.4 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
- Computation Scheme: t_n onto t_{ref}

- Padding: no
- Undersampling time with Lowess interpolation: no

Then it is possible to use already produced transformations onto a different channel. This can be important when comparing two channels from the same movie.

6.5 Stabilizing a movie with high displacements over time

When imaging for a long period of time, the imaged sample can move compared to the camera reference. It is even more pronounce if the sample cannot be attached during the imaging period. This movements of the sample in front of the camera over time can make the movie harder to interpret when looking at it but mostly it increases the difficulty for detection and tracking algorithms. Registering the whole movie onto a single time point would mostly get rid of these artificial movements and help the algorithms to extract quantitative information.

Let `'/path/to/time/series/T000/CH00-T0000.tif'` be the pattern for the images of our movie. Our movie goes from time point 0 to time point 200 but during the imaging period the time points 23 and 145 were wrongly imaged. This is then an example of a json configuration file:

```
{
   "path_to_data":"/path/to/time/series/T{t:03d}/",
   "file_name":"CH00-T{t:04d}.tif",
   "trsf_folder":"/path/to/output/trsf/",
   "output_format":"CH00-T{t:04d}-rigid.klb",
   "projection_path":"/path/to/output/projection/",
   "check_TP":1,
   "voxel_size":[
      0.593,
      0.593,
      5
   ],
   "first":0,
   "last":200,
   "compute_trsf":1,
   "ref_TP":100,
   "trsf_type":"rigid",
   "not_to_do":[
      23,
      145
   ],
   "padding":1,
   "apply_trsf":1
}
```

From this configuration file

References

Cleveland, W. S. and Loader, C. (1996). Smoothing by local regression: Principles and methods. In Härdle, W. and Schimek, M. G., editors, *Statistical Theory and Computational Aspects of Smoothing*, pages 10–49, Heidelberg. Physica-Verlag HD.

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.