`blockmatching` and associated tools, a very short documentation

Grégoire Malandain

January 17, 2018

Contents

Part I

User documentation

Chapter 1

Command Line Interfaces

1.1 `blockmatching`

1.1.1 Basic use

`blockmatching` registers a floating image I_{flo} onto a reference image I_{ref}, and yields two results: the transformation from the reference image frame towards the floating image frame, $T_{res} = T_{I_{flo} \leftarrow I_{ref}}$, and the floating image resampled in the same frame than the reference image, $I_{res} = I_{flo} \circ T_{I_{flo} \leftarrow I_{ref}}$.

```
% blockmatching -flo Iflo -ref Iref -res Ires -res-trsf Tres ...
```

$T_{res} = T_{I_{flo} \leftarrow I_{ref}}$ allows to resample I_{flo}, or any image defined in the same frame than I_{flo}, into the same frame than I_{ref}, which can also be done afterwards with `applyTrsf` .

```
% applyTrsf Iflo Ires -trsf Tres -template Iref
```

For instance, this allows to *visualize* the deformation undergone by I_{flo} by applying the transformation to a grid image (see `createGrid`).

1.1.2 Principle

The principle of `blockmatching` is to pair blocks from the floating image I_{flo} with blocks of the reference image I_{ref}, i.e. for each blocks in the floating image I_{flo}, we are looking in a neighborhood of the reference image I_{ref} for the best similar block.

More precisely, `blockmatching` computes $T_{res} = T_{I_{flo} \leftarrow I_{ref}}$ that allows to resample I_{flo} in the frame of T_{ref}. At iteration i, pairings are built between $I_{flo} \circ T^{(i)}_{I_{flo} \leftarrow I_{ref}}$ (I_{flo} resampled by the estimation of $T_{I_{flo} \leftarrow I_{ref}}$ at iteration i)

and T_{ref}. With these pairings, an incremental transformation $\delta T^{(i)} = T_{I_{flo}^{(i)} \leftarrow I_{ref}}$ is computed and used to update the transformation:

$$T^{(i+1)}_{I_{flo} \leftarrow I_{ref}} = T^{(i)}_{I_{flo} \leftarrow I_{ref}} \circ \delta T^{(i)}$$

- Registration can be conducted within a hierarchical approach, by the means of image pyramids: see section 1.1.6.1.
- Pairings are built by associating similar blocks (or small images) between the resampled floating image $I_{flo} \circ T^{(i)}_{I_{flo} \leftarrow I_{ref}}$ and the reference image I_{ref}: section 1.1.6.2 introduces some option for the block definition while section 1.1.6.3 presents options for block pairings.
- Transformations are estimated by the means of (weighted) least (trimmed) squares: see section 1.1.6.4 for generic options for transformation estimation, and section 1.1.6.5 that presents dedicated options for vector field estimation.

1.1.3 Options and parameters (general informations)

Running the program without any options gives the minimal syntax:

```
% blockmatching
```

Running it with either `'-h'` or `'--h'` gives the option list:

```
% blockmatching -h
```

Running it with either `'-help'` or `'--help'` gives some details about the options:

```
% blockmatching -help
```

[Pay attention] Default parameters depend on the transformation type (linear or non-linear). The `'-print-parameters'` allows printing the values of the parameters, so running `blockmatching` with this option along with the chosen type of transformation (with `'-trsf-type'`) may be a good idea.

```
% blockmatching ...  -print-parameters
```

If a logfile name is given (with option `'-logfile'`), parameter values will be printed out in this file.

In addition, when changing parameters, comparing the parameter values with the `'-print-parameters'` before and after the parameter changes allows to check that the applied changes are the expected ones.

1.1.4 Initial transformations

Two transformations, T_{left} and T_{init}, can be passed as parameters. While T_{left} (left-handed transformation) will remain unchanged during the registration procedure, T_{init} is the initial value of the transformation to be computed (recall this is an iterative calculation). More precisely (see also section 5.6), let $T^{(0)}$ denote the initial value the transformation to be computed.

- if T_{init} is given, $T^{(0)} = T_{init}$
- else
  - if T_{left} is given, $T^{(0)} = \mathbf{Id}$
  - else (neither T_{init} nor T_{left} are given), $T^{(0)}$ is the default transformation (can be specified with the `'-default-transformation'` option). It is either the translation that superimposes the centers of the fields of view of the two images I_{ref} and I_{flo} (`'-default-transformation fovcenter'`), or the identity (`'-default-transformation identity'`). The default behavior is to superimpose the centers of field of view (see section 5.6).

The initial state of the registration procedure is the comparison of the reference image I_{ref} with the transformed floating image $I_{flo}\ (\circ T_{left}) \circ T^{(0)}$.

`'-[left|initial]-transformation'` allows to initialize T_{left}, while

`'-initial-result-transformation'` allows to initialize T_{init}

- the option `'-[initial|left]-[voxel-]transformation'` is used to specify a transformation T_{left} that is applied to the floating image I_{flo}. Therefore, it comes to register $I_{flo} \circ T_{left}$ with I_{ref}. Thus, the resulting transformation, T_{res}, allows to resample I_{flo} onto I_{ref} by calculating $I_{flo} \circ T_{left} \circ T_{res}$.

  In other words, the transformation T_{res} obtained with

  ```
  % blockmatching -flo I_flo -ref I_ref ...  -initial-transformation T_left
  -res-trsf T_res
  ```

  is comparable to the one, $T_{res,2}$ obtained with the following commands

  ```
  % applyTrsf I_flo I_flo,2 -trsf T_left
  % blockmatching -flo I_flo,2 -ref I_ref ...  -res-trsf T_res,2
  ```

  [Note:] When the `'-composition-with-[initial|left]'` is specified, the result transformation that is written is $T_{left} \circ T_{res}$.

  This may be useful in case of successive registrations, e.g. with different transformation types. Indeed, one may want to first register the two images I_{flo} and I_{ref} with a rigid transformation and then with an affine transformation.

  1. A first solution is to compute the rigid transformation

     ```
     % blockmatching -flo I_flo -ref I_ref ...  -res-trsf T_rig -trsf-type rigid
     -res I_flo,2
     ```

and then an affine transformation by comparing the previous result $I_{flo,2} = I_{flo} \circ T_{rig}$ with I_{ref}

% `blockmatching -flo` $I_{flo,2}$ `-ref` I_{ref} `... -res-trsf` T_{aff} `-trsf-type affine -res` $I_{flo,3}$

I_{flo} can be then be directly resampled onto I_{ref} with the composed transformation $T_{resampling} = T_{rig} \circ T_{aff}$

% `composeTrsf -res` $T_{resampling}$ `-trsfs` T_{rig} T_{aff}

2. A second solution consists in also first computing the rigid transformation

   % `blockmatching -flo` I_{flo} `-ref` I_{ref} `... -res-trsf` T_{rig} `-trsf-type rigid -res` $I_{flo,2}$

   but then T_{rig} is used as parameter to `'-left-transformation'` with I_{flo} as floating image

   % `blockmatching -flo` I_{flo} `-ref` I_{ref} `... -res-trsf` T_{aff} `-trsf-type affine -res` $I_{flo,4}$ `-left-transformation` T_{rig}

   This second solution is to be prefered. Indeed, the resampled version of I_{flo} by T_{rig} can miss some data (data of I_{flo} that are not in the field of view (FOV) of I_{ref} after resampling) and cropping effects may appear (parts of FOV of I_{ref} that do not have correspondant areas in I_{flo} by T_{rig}).

   Note that the command

   % `blockmatching -flo` I_{flo} `-ref` I_{ref} `... -res-trsf` $T_{resampling}$ `-trsf-type affine -res` $I_{flo,4}$ `-left-transformation` T_{rig} `-composition-with-left`

   has $T_{resampling}$ as output transformation.

- the option `'-initial-result-[voxel-]transformation'` is used to specify the initial value of the transformation to be computed. This can be used to continue a registration done at the higher scale.

  For instance, the result transformation T_{res} computed in one shot by

  % `blockmatching -flo` I_{flo} `-ref` I_{ref} `... -res-trsf` T_{res} `-py-ll 0 -py-hl 3 -flo-frac 0.75`

  is equal to the result transformation $T_{res,2}$ that is computed in two step (the first step uses the scales 3 and 2, while the second one uses the scales 1 and 0).

  % `blockmatching -flo` I_{flo} `-ref` I_{ref} `... -res-trsf` $T_{intermediary}$ `-py-ll 2 -py-hl 3 -flo-frac 0.75`
  % `blockmatching -flo` I_{flo} `-ref` I_{ref} `... -init-res-trsf` $T_{intermediary}$ `-res-trsf` $T_{res,2}$ `-py-ll 0 -py-hl 1 -flo-frac 0.75`

  This may allow to qualitatively evaluate the intermediary result before running the algorithm at the lower scales that are computationaly expensive.

[**Note:**] Please note the `'-flo-frac 0.75'` (see option `'-floating-selection-fraction'`, page 10) that sets the fraction of blocks of the floating image to be kept for the pairing search. Thanks to this option, it is constant through the pyramid levels, and for both computation schemes (one-step and two-steps).

Initial transformations can be obtained by pairing points with `pointmatching` (see section 1.12).

1.1.5 Result transformation

`'-result-[voxel-]transformation'` is used to specify the name of the output transformation.

When no left-handed transformation is passed, it is the transformation $T_{res} = T_{I_{flo} \leftarrow I_{ref}}$ that allows to resample I_{flo} onto I_{ref}.

When a left-handed transformation is passed with the option `'-left-[voxel-]transformation'`, the result transformation T_{res} is the one that allows to resample $I_{flo} \circ T_{left}$ onto I_{ref}, i.e. $T_{res} = T_{I_{flo} \circ T_{left} \leftarrow I_{ref}}$. This should be the default behavior that can be enforced with the option `'-no-composition-with-left'`. To resample directly I_{flo} onto I_{ref}, it is required to have $T_{left} \circ T_{res}$ at hand. While it is straightforward to compute it thanks to `composeTrsf` , it is also possible to get the composed transformation $T_{left} \circ T_{res}$ as the output transformation with the option `'-composition-with-left'`.

1.1.6 Some useful parameters

The use of some parameters is detailed here. For a complete list of parameters (and short description of them), the user is advised to use the `'-help'` option.

1.1.6.1 Hierarchical registration

To speed up the computation and to handle large deformations, the transformation is computed hierarchically. For each image I_{flo} and I_{ref}, pyramids of images are built, i.e. the pyramid $\{I^n_{flo}, \ldots, I^0_{flo}\}$ for the I_{flo} image, where $I^0_{flo} = I_{flo}$ and where the dimensions of I^{i+1}_{flo} are the ones of I^i_{flo} divided[1] by 2.

Thus, the registration is done with pyramids $\{I^h_{flo}, \ldots, I^l_{flo}\}$ and $\{I^h_{ref}, \ldots, I^l_{ref}\}$, with h and l respectively specified by the `'-pyramid-highest-level'` and `'-pyramid-lowest-level'` options. A first transformation is computed with $\{I^h_{flo}, I^h_{ref}\}$, that is used as initialization for the registration of $\{I^{h-1}_{flo}, I^{h-1}_{ref}\}$, and so on and so forth until $\{I^l_{flo}, I^l_{ref}\}$.

The lowest level of the pyramid (i.e. the image at its native sizes) is specified by 0, i.e. `'-pyramid-lowest-level 0'`.

[1]This is not strictly true, since the dimensions of I^1_{flo} will be chosen to be the 2^k values that are closest (and smaller) to I^0_{flo} dimensions when I^0_{flo} dimensions are not powers of 2.

- Specifying a high value of h allows to search for long-range pairings (since a small displacement at a high scale image corresponds to a large displacement at a lower scale), and to speed up the computation: for 3D images, the data are reduced with a factor 8 between two consecutive scales (and so is the number of blocks).
- On the other hand, when looking for small deformations, specifying a large h is definitively not adequate, except if the search neighborhood size is adapted to the scale.

The finest the resolution, the more heavy the computation. When it comes to tune parameters, it may be useful to only perform the registration with the highest levels of the pyramid (ie to specify a high value to the `'-pyramid-lowest-level'` option, and to visually check the results before launching the computation with the lowest levels of the pyramid.

Moreover, if the images to be registered are quite large, I/0 operations and pyramid computation can be quite costly. It may then be considered to compute subsampled images beforehand, either by choosing the subsampling parameters with `applyTrsf` (see section 1.3) or by picking one of the pyramid images built with `buildPyramidImage` (see section 1.5) that can be computed also beforehand, and then perform the registration on subsampled images. It is detailed in section 1.1.7.2.

1.1.6.2 Block definition/selection parameters

To build blocks, two parameters are important:

> `'-block-size'` allows to specify the size of the blocks. Small blocks require a small computational effort to be processed (since the similarity measure complexity depends on the number of points inside the block), but are more likely to be paired to the "wrong" block.
>
> On the contrary, since the information distribution is more complex in a large block, pairing large blocks is more likely to build the "right" pairings
>
> `'-block-spacing'` specify the spacing between consecutive blocks. Increasing the spacing allows to speed up the computation (by decreasing the number of blocks), at the price of sparser pairings.

Some of the built blocks can be discarded for the pairing search, either because they may not yield pertinent pairings, or because they are not carrying useful information.

A first selection of points or blocks can be done based on intensity values, which may be useful to prevent blocks to be build in low intensity areas as the background.

> `'-floating-low-threshold'` and `'-floating-high-threshold'` allow to specify two intensity thresholds for the blocks built from the floating image I_{flo}. Points with values lower or equal to the low threshold, or

higher or equal to the high threshold are discarded from the block for the similarity measure computation.

`'-floating-removed-fraction'` specifies the maximal fraction of points that can be removed from a block. If too many points are removed (because of the two thresholds), the block is discarded and not considered for further computation.

options `'-reference-low-threshold'`,`'-reference-high-threshold'`, and `'-reference-removed-fraction'` are similar options for the reference image I_{ref}.

Previous options help to control the building of individual blocks. To further control the total number of blocks (and hence decrease the computational effort), it is also possible to select the most informative blocks from the block list. According that the standard deviation is representative of the information (blocks of uniform values have a small standard deviation), the percentage of blocks of higher standard deviation can be specified, and only these blocks are considered for the transformation computation.

`'-floating-selection-fraction'` allows to specify the fraction of blocks of the floating image to be kept for the pairing search. It can be more finely tuned with the suffixes `'-ll'` et `'-lt'`.

`'-floating-selection-fraction-ht'` and `'-floating-selection-fraction-lt'` allow to specify this fraction for respectively the highest and the lowest level of the pyramid (see section 1.1.6.1) while the fraction value for intermediary levels is linearly interpolated from these two fractions.

While it may not be required (even not advised) to remove blocks of the floating image for the pairing search (and thus for the transformation computation) at highest levels of the pyramid (where there are few blocks), it may be convenient (even advised) to remove some of them ar the lowest levels (where areas of constant image intensity are likely to occur). Thus, default values for this fraction are set to 1.0 for the highest level and 0.5 for the lowest level, independently of the level values. This has to be kept in mind when registration is conducted in several steps through the pyramid levels (see option `'-initial-result-transformation'` page 7).

1.1.6.3 Block search/pairing

To build pairings, each (selected) block of the floating image is compared to blocks of the reference image, and is paired with its best correspondent. The comparison is restricted to a neighbourhood.

`'-search-neighborhood-half-size'` allows to specify the size of the search neighbourhood (in the reference image for a given block from the floating image) to build pairings. The larger the value, the larger the searched displacement (and the larger the potential resulting deformation), and, of course, the larger the computational effort.

`'-search-neighborhood-step'` allows to specify the step (the increment) between blocks in the search neighbourhood. Typically, a step equal to 2 allows to decrease by a factor 4 in 2D (8 in 3D) the number of blocks to tested in the search neighbourhood. However, the resulting pairings will be less precise.

1.1.6.4 Transformation estimation

`'-transformation-type'` allows to specify the type of computed transformation.It is desirable to compute transformation in a hierarchical manner, from the ones with few degrees of freedom to the ones with more degrees of freedom: e.g. `'rigid'`, then `'affine'`, then `'vectorfield'`.

`'-estimator-type'` allows to specify the calculation method to estimate the incremental transformation δT from the pairings. Calculation is done by the mean of (weighted) least (trimmed) square. In case of weighted least (trimmed) squares, residuals to be minimized are weighted by the block similarity. Least trimmed squares methods allow to discard outliers from the transformation estimation.

Least trimmed squares estimation is an iterative method. At each iteration (except the first one where least squares estimation is used), outliers are discarded. Outliers are defined as the samples having the largest residuals after the previous estimation.

`'-lts-fraction'` defines the fraction of samples to be kept for estimation. Obviously, it should be larger than `'0.5'`.

`'-lts-deviation'` allows to define the outliers with respect to residual statistics (according they follows a normal law). Let $\hat{m}_r$ and $\hat{\sigma}_r$ be respectively the mean and the standard deviation of the residuals, passing the value c to `'-lts-deviation'` set the residual rejection threshold at $\hat{m}_r + c\hat{\sigma}_r$.

`'-lts-iterations'` set a maximum number of iterations for the least trimmed squares estimation.

1.1.6.5 Vector field transformations

Non-linear transformations are (up to today) encoded through vector fields. At each iteration i, an incremental transformation δT est computed from the pairings, and then composed with the current transformation T^i to yield the updated transformation T^{i+1}:

Building a vector field from pairing is done by interpolation with a Gaussian kernel whose standard deviation is specified by the `'-fluid-sigma'` option. Indeed, this interpolation also regularizes the pairings, and it can viewed as a fluid regularization since it is done on the incremental transformation. It can be more finely tuned with the suffixes `'-ll'` et `'-lt'`.

Note that the outliers detection (and removal) of trimmed estimations is done by comparing the pairings to the regularized δT.

`'-elastic-sigma'` is used afterwards (after composition with the incremental transformation) to regularize the global transformation T^{i+1}.

`'-vector-propagation-distance'` and `'-vector-fading-distance'` are presented in the section dedicated to `pointmatching` (section 1.12). They are more useful in case of (very) sparse pairings.

1.1.7 Hints

1.1.7.1 How to tune parameters

- A priori, pairings can be better built (and outliers can be better avoided) by specifying larger blocks with `'-block-size'`
- Non-informative (i.e. almost of uniform value) blocks should be avoided since they may yield poor pairings. Selection can be done either on point-wise criteria (e.g. intensity based selection with `'-floating-low-threshold'` and `'-floating-high-threshold'`) or block-wise criteria (e.g. `'-floating-selection-fraction'`).
- When looking for small displacements/deformations, long distance pairings may be discouraged by
  - only considering low levels of the pyramid (`'-pyramid-highest-level'`),
  - diminishing the size of the search neighbourhood (`'-search-neighborhood-half-size'`)
- The regularization of non-linear deformations can be tuned with both `'-fluid-sigma'` and `'-elastic-sigma'`. The larger the values, the less local the deformations.

1.1.7.2 Hand-made hierarchical registration

If the images to be registered are quite large, I/0 operations and pyramid computation can be quite costly. It may then be considered to compute subsampled images beforehand, either by choosing the subsampling parameters with `applyTrsf` (see section 1.3.2) or by picking one of the pyramid images (see section 1.5) that can be computed also beforehand, and then perform the registration on subsampled images. In both case, it is mandatory to use `'-res-trsf'` to get the subsampling transformation.

Let us consider the images I_{flo} and I_{ref} to be registered. I_{flo} can be subsampled into I^s_{flo} with:

`% applyTrsf` I_{flo} I^s_{flo} ... `-res-trsf` T^s_{flo} `-resize`

and we get $I^s_{flo} = I_{flo} \circ T^s_{flo}$. The same stands for I_{ref} and we get $I^s_{ref} = I_{ref} \circ T^s_{ref}$. Registration can be done on subsampled images with

% `blockmatching -flo` I^s_{flo} `-ref` I^s_{ref} `-res` I^s_{res} `-res-trsf` T^s_{res} ...

$I^s_{res} = I^s_{flo} \circ T_{res}$ is I^s_{flo} resampled in the geometry of I^s_{ref}. We have then

$$\begin{aligned} I^s_{res} = I^s_{flo} \circ T_{res} \quad & \Rightarrow \quad I_{res} \circ T^s_{ref} = I_{flo} \circ T^s_{flo} \circ T_{res} \\ & \Rightarrow \quad I_{res} = I_{flo} \circ T^s_{flo} \circ T^s_{res} \circ {T^s_{ref}}^{(-1)} \end{aligned}$$

A transformation $\tilde{T}_{res}$ that allows to transform I_{flo} in the frame of I_{ref} can be estimated by $\tilde{T}_{res} = T^s_{flo} \circ T^s_{res} \circ {T^s_{ref}}^{(-1)}$, then with the commands

% `invTrsf` T^s_{ref} ${T^s_{ref}}^{(-1)}$

% `composeTrsf -res` $\tilde{T}_{res}$ `-trsfs` T^s_{flo} T^s_{res} ${T^s_{ref}}^{(-1)}$ `-template` I_{ref}

$\tilde{T}_{res}$ can be calculated. Please notice the use of `'-template'` that ensures that $\tilde{T}_{res}$ is defined on the I_{ref} frame (mandatory for non-linear transformations). Last,

% `applyTrsf` I_{flo} $\tilde{I}_{res}$ `-trsf` $\tilde{T}_{res}$ `-template` I_{ref}

yields $\tilde{I}_{res}$, ie I_{flo} resampled in I_{ref} frame thanks to $\tilde{T}_{res}$.

This is exemplified in section 3.2.

1.2 `blockmatching` versus `baladin`

A typical call to `blockmatching` is

% `blockmatching -flo` I_{flo} `-ref` I_{ref} `-res` I_{block} `-res-trsf` T_{block} `-res-voxel-trsf` $\hat{T}_{block}$ ...

where I_{block}, T_{block}, and $\hat{T}_{block}$ denote respectively the result image, i.e. the floating image resampled in the frame of I_{ref}, the transformation result in *real* coordinates that allows to goes from I_{ref} frame towards I_{flo} frame (and then to resample I_{flo} in the frame of I_{ref}), and the transformation result in *voxel* coordinates.

A typical call to `baladin` is

% `baladin -flo` I_{flo} `-ref` I_{ref} `-res` I_{balad} `-result-matrix` $\hat{T}_{balad}$ `-result-real-matrix` T_{balad} ...

where I_{balad}, T_{balad}, and $\hat{T}_{balad}$ denote respectively the result image, i.e. the floating image resampled in the frame of I_{ref}, the transformation result in *real* coordinates that allows to goes from I_{flo} frame towards I_{ref} frame, and the transformation result in *voxel* coordinates.

[**Pay attention**] The result transformation of `baladin` is then the inverse of that of `blockmatching` . We have

$$T_{block} \sim T^{-1}_{balad} \quad \text{and} \quad \hat{T}_{block} \sim \hat{T}^{-1}_{balad}$$

Therefore $T_{block} \circ T_{balad}$ and $\hat{T}_{block} \circ \hat{T}_{balad}$ should be close to the identity, and this can be verified by computing $T_{test} = T_{block} \circ T_{balad}$ and $\hat{T}_{test} = \hat{T}_{block} \circ \hat{T}_{balad}$:

```
% composeTrsf -res $T_{test}$ -trsfs $T_{block}$ $T_{balad}$
% printTrsf $T_{test}$
% composeTrsf -res $\hat{T}_{test}$ -trsfs $\hat{T}_{block}$ $\hat{T}_{balad}$
% printTrsf $\hat{T}_{test}$
```

The transformations issued from `baladin` can be used to resample the floating image I_{flo} but require to be inverted beforehand. To compute the resampled floating image I_{balad} from the *real* transformation T_{balad}, the commands are:

```
% invTrsf $T_{balad}$ $T^{-1}_{balad}$
% applyTrsf $I_{flo}$ $I_{balad}$ -template $I_{ref}$ -trsf $T^{-1}_{balad}$
```

I_{balad} can also be computed from from the *voxel* transformation $\hat{T}_{balad}$ accordingly

```
% invTrsf $\hat{T}_{balad}$ $\hat{T}^{-1}_{balad}$
% applyTrsf $I_{flo}$ $I_{balad}$ -template $I_{ref}$ -voxel-trsf $\hat{T}^{-1}_{balad}$
```

1.3 applyTrsf

1.3.1 Basic use

`applyTrsf` allows to resample an image according to a transformation T. One has to recall that the transformation goes from the *destination/result image* towards the *image to be resampled.* This is counter-intuitive, but can easily be explained: to compute the value of the point M in the destination/result image, one has to know where this point comes from in the image to be resampled.

So, to resample the image I_{flo} in the same frame than the image I_{ref} with a transformation that T goes from I_{ref} towards I_{flo}, i.e.

$$\begin{aligned} T &= T_{I_{flo} \leftarrow I_{ref}} \\ I_{res} &= I_{flo} \circ T \end{aligned}$$

the command is

```
% applyTrsf $I_{flo}$ $I_{res}$ -trsf $T$ -template $I_{ref}$
```

The `'-trsf'` implies that the transformation is in *real frames* (i.e. the coordinates of the points are in real units, for instance millimeters). We recall that to a voxel point $M_{\mathbb{Z}} = (i, j, k)$ is associated a real point $M_{\mathbb{R}} = (x, y, z)$ through a conversion matrix $H_{I,\mathbb{R} \leftarrow \mathbb{Z}}$ (typically a diagonal matrix containing the voxel sizes along each direction).

The `'-voxel-trsf'` allows to specify the transformation in *voxel frames*. The transformation in the *voxel frames*, $T_{I_{flo} \leftarrow I_{ref}, \mathbb{Z}}$ is obtained from the transformation in the *real frames*, $T_{I_{flo} \leftarrow I_{ref}, \mathbb{R}}$, by multiplying it by the conversion matrices:

$$T_{I_{flo} \leftarrow I_{ref}, \mathbb{Z}} = H^{-1}_{I_{flo}, \mathbb{R} \leftarrow \mathbb{Z}} \circ T_{I_{flo} \leftarrow I_{ref}, \mathbb{R}} \circ H_{I_{ref}, \mathbb{R} \leftarrow \mathbb{Z}}$$

Conversely,

$$T_{I_{flo} \leftarrow I_{ref}, \mathbb{R}} = H_{I_{flo}, \mathbb{R} \leftarrow \mathbb{Z}} \circ T_{I_{flo} \leftarrow I_{ref}, \mathbb{Z}} \circ H^{-1}_{I_{ref}, \mathbb{R} \leftarrow \mathbb{Z}}$$

1.3.2 Changing image geometry

`applyTrsf` may be used to change the image geometry, i.e. either changing the number of pixel/voxel or the pixel/voxel size along a direction. Please note that the field of view is considered as unchanged, so that the number of pixel/voxel time the pixel/voxel size is a constant. This is done by the `'-resize'` option.

Let consider an image I of dimensions 1000×1000 with a pixel size of 1 along each direction. Then

```
% applyTrsf I J -dim 200 300 -res-trsf T -resize
```

creates an image J of dimensions 200×300: pixel/voxel sizes are calculated so as the image spans the same field of view and are respectively 5.00 and 3.33. The transformation T is the transformation used to resample I, we have then $J = I \circ T$. When expressed in the real frame, this transformation is a translation (see section 4.2) that superimposes the centers of the fields of view of the two images.

Image geometry can also be "imported" from an other image. Let us consider an image I and a template image R, thus

```
% applyTrsf -flo I -ref R -res J
```

will produce an image J with the same geometry than the image R. The voxel-to-voxel transformation that allows to resample I in R is here computed by

$$\mathbf{H}_{I, \mathbb{Z} \leftarrow \mathbb{R}} \circ \mathbf{Id} \circ \mathbf{H}_{R, \mathbb{R} \leftarrow \mathbb{Z}}$$

where **Id** stands for the identity. (please refer to section 4.2 for the definition of the conversion matrices **H**). Please note that here the centers of fields of view will not superimpose.

Last, the command

```
% applyTrsf -flo I -ref R -res J -initial-trsf fovcenter
```

will also produce an image J with the same geometry than the image R, but will also superimposes the centers of field of view. The voxel-to-voxel transformation that allows to resample I in R is here computed by

$$\mathbf{H}_{I, \mathbb{Z} \leftarrow \mathbb{R}} \circ \mathbf{T}_{FOV} \circ \mathbf{H}_{R, \mathbb{R} \leftarrow \mathbb{Z}}$$

where $\mathbf{T}_{FOV}$ is the translation (in real coordinates) that superimposes the centers of field of view (see section 5.6). Such an example (options are similar to `blockmatching` command line, where the default initial transformation superimposes the centers of field of view) emphasizes that `applyTrsf` can be used to resample the floating image I at its starting position when registering it with a reference image R.

1.4 applyTrsfToPoints

1.5 buildPyramidImage

1.6 composeTrsf

`composeTrsf` allows to compose a series of transformations. The transformations to be composed are introduced by the `'-trsfs'` option:

% `composeTrsf ... -res` T_{res} `-trsfs` T_1 T_2 `...` T_N

Transformations are composed in the order they are given. The line `'-trsfs` T_1 T_2 `...` T_N`'` assumes that the transformation T_i goes from image I_{i+1} to image I_i (then allows to resample I_i in the same frame than I_{i+1}), i.e.

$$T_i = T_{I_i \leftarrow I_{i+1}}$$

The resulting transformation will goes from I_{N+1} to I_1 (then allows to resample I_1 in the same frame than I_{N+1}). Thus

% `composeTrsf ... -res` T_{res} `-trsfs` T_1 T_2 `...` T_N

computes

$$\begin{aligned} T_{res} &= T_1 \circ T_2 \circ ... \circ T_N \\ &= T_{I_1 \leftarrow I_2} \circ T_{I_2 \leftarrow I_3} \circ ... \circ T_{I_N \leftarrow I_{N+1}} \\ &= T_{I_1 \leftarrow I_{N+1}} \end{aligned}$$

Example: the following series of resampling

% `applyTrsf` I_0 I_1 `-trsf` T_0 `...`
% `applyTrsf` I_1 I_2 `-trsf` T_1 `...`
% `applyTrsf` I_2 I_3 `-trsf` T_2 `...`

is equivalent to the transformation composition

% `composeTrsf -res` $T_{I_0 \leftarrow I_3}$ `-trsfs` T_0 T_1 T_2

that allows to get I_3 directly from I_0

% `applyTrsf` I_0 I_3 `-trsf` $T_{I_0 \leftarrow I_3}$ `...`

Example: when registering non-linearly two images, it is usual to perform several registration with an increasing complexity of the sought transformations:

- either rigid $\rightarrow$ affine $\rightarrow$ vectorfield,
- or affine $\rightarrow$ vectorfield.

In the latter case, it comes to do

```
% blockmatching -flo $I_{flo}$ -ref $I_{ref}$ -res $I'_{flo}$ -res-trsf $T_0$
    -trsf-type affine ...
% blockmatching -flo $I'_{flo}$ -ref $I_{ref}$ -res $I''_{flo}$ -res-trsf $T_1$
    -trsf-type vectorfield ...
```

Thus, to directly resample I_{flo} into the geometry of I_{ref}, the commands are

```
% composeTrsf -res $T_{I_{flo} \leftarrow I_{ref}}$ -trsfs $T_0$ $T_1$
% applyTrsf $I_{flo}$ $I_{res}$ -trsf $T_{I_{flo} \leftarrow I_{ref}}$ ...
```

I_{res} being comparable to I''_{flo}.

[**Pay attention**] Transformations are assumed to be in real units.

1.7 copyTrsf

`copyTrsf` allows to copy a transformation from one type to an other or/and to convert it from *real units* to *voxel units* or conversely.

The command

```
% blockmatching -flo $I_{flo}$ -ref $I_{ref}$ -res $I_{res}$ -res-trsf $T_{res,\mathbb{R}}$
-res-voxel-trsf $T_{res,\mathbb{Z}}$ ...
```

allows to register the image I_{flo} onto I_{ref} and computes the transformation $T_{res,\mathbb{R}}$ (in real units) that allows to resample I_{flo} in the same frame tham I_{ref}, the transformation $T_{res,\mathbb{Z}}$ being $T_{res,\mathbb{R}}$ expressed in voxel units).

The conversion from *real* to *voxel* units can also be achieved by

```
% copyTrsf $T_{res,\mathbb{R}}$ $T_{res,\mathbb{Z}}$ -floating $I_{flo}$ -template $I_{ref}$ -input-unit real
-output-unit voxel
```

while the conversion from *voxel* to *real* units can also be achieved by

```
% copyTrsf $T_{res,\mathbb{Z}}$ $T_{res,\mathbb{R}}$ -floating $I_{flo}$ -template $I_{ref}$ -input-unit voxel
-output-unit real
```

`copyTrsf` can also be used to copy a linear transformation T_{linear}, expressed as a matrice, in a vector field, T_{vector}. It is mandatory to provide a template image that defines the geometry of the vector field (which is nothing but a vectorial image).

```
% copyTrsf $T_{linear}$ $T_{vector}$ -template $I_{ref}$ -trsf-type vectorfield[2D,3D]
```

1.8 createGrid

`createGrid` creates an image containing a grid, which can be useful to "visualize" transformations or deformations.

Example: the following registration has been ran

```
% blockmatching -flo $I_{flo}$ -ref $I_{ref}$ ...  -res-trsf $T_{res}$ -res $I_{res}$
```

One can create a grid image having the same geometry than I_{flo} (thanks to '`-template` I_{flo}')

```
% createGrid $I_{grid}$ -template $I_{flo}$
```

and use T_{res} to resample this grid image into the geometry of I_{ref} (thanks to '`-template` I_{ref}')

```
% applyTrsf $I_{grid}$ $I_{resampled\_grid}$ -trsf $T_{res}$ -template $I_{ref}$
```

$I_{resampled\_grid}$ exhibits the same transformation/deformation with respect to I_{grid} than I_{res} with respect to I_{flo}.

1.9 createTrsf

1.10 cropImage

`cropImage` allows to crop an image. As a side result, it can also write the transformation "summarizing" the crop.
Example: the following command crops, from I_{ref}, a subvolume J_{ref} of dimensions $[100, 90, 80]$ from the point $(25, 35, 45)$ [by convention, the default origin is (0,0,0)].

```
% cropImage $I_{ref}$ $J_{ref}$ -origin 25 35 45 -dim 100 90 80 -res-trsf $C_{ref}$
```

The transformation C_{ref} defines the "crop" operation as a transformation, i.e. $J_{ref} = I_{ref} \circ C_{ref}$, this the same "crop" can also be done by

```
% applyTrsf $I_{ref}$ $J_{ref}$ -trsf $C_{ref}$ -dim 100 90 80 -voxel ...
```

according one specifies the correct voxel sizes. In other words, we have

$$J_{ref} = I_{ref} \circ C_{ref}$$

This allows to compute a registration transformation from subvolumes, and then to estimate the transformation for the whole volumes.

1. The commands
   ```
   % cropImage $I_{ref}$ $J_{ref}$ ...  -res-trsf $C_{ref}$
   % cropImage $I_{flo}$ $J_{flo}$ ...  -res-trsf $C_{flo}$
   ```
   generates the subvolumes $J_{ref} = I_{ref} \circ C_{ref}$ and $J_{flo} = I_{flo} \circ C_{flo}$ together with the crop transformations C_{ref} and C_{flo}.

2. The cropped images are co-registered, i.e. J_{flo} can be registered onto J_{ref} with
   ```
   % blockmatching -flo $J_{flo}$ -ref $J_{ref}$ -res $J_{res}$ -res-trsf $T'_{res}$ ...
   ```

3. The resampling of the cropped image J_{flo} into J_{res} with T'_{res} can cause some zeroed areas appearing at the J_{res} image border. Since we have

$$J_{res} = J_{flo} \circ T'_{res}$$

$$= \quad I_{flo} \circ C_{flo} \circ T'_{res}$$

this effect can be reduced by resampling I_{flo} into J_{res} with the transformation $C_{flo} \circ T'_{res}$

```
% composeTrsf -res $T''_{res}$ -trsfs $C_{flo}$ $T'_{res}$
% applyTrsf $I_{flo}$ $J_{res}$ -trsf $T''_{res}$ -template $J_{ref}$
```

4. Last, we also have

$$\begin{aligned} J_{res} = J_{flo} \circ T'_{res} &\sim J_{ref} \\ I_{flo} \circ C_{flo} \circ T'_{res} &\sim I_{ref} \circ C_{ref} \\ I_{flo} \circ C_{flo} \circ T'_{res} \circ C_{ref}^{-1} &\sim I_{ref} \end{aligned}$$

thus $T_{res} = C_{flo} \circ T'_{res} \circ C_{ref}^{-1}$ allows to resample I_{flo} onto I_{ref}, with a transformation T'_{res} computed by the co-registration of the cropped images J_{flo} and J_{ref}.

```
% invTrsf $C_{ref}$ $C_{ref}^{-1}$
% composeTrsf -res $T_{res}$ -trsfs $C_{flo}$ $T'_{res}$ $C_{ref}^{-1}$
% applyTrsf $I_{flo}$ $I_{res}$ -trsf $T_{res}$ -template $I_{ref}$
```

1.11 invTrsf

`invTrsf` allows to invert transformations. Attention should be paid for vector field transformations, since they are defined as vectorial images. When registering images with `blockmatching` ,

```
% blockmatching -flo $I_{flo}$ -ref $I_{ref}$ -res $I_{res}$ -res-trsf $T_{res}$ ...
```

the geometry (image dimensions and voxel sizes) of the vector field is that of the reference image I_{ref} (recall that T_{res} allows to resample I_{flo} into the geometry of I_{ref} and can be denoted by $T_{I_{flo} \leftarrow I_{ref}}$, see section 1.3).

Inverting T_{res} into T_{res}^{-1} will allows to resample I_{ref} into the geometry of I_{flo} and has then to be defined with I_{flo} geometry with `'-template-image'`

```
% invTrsf $T_{res}$ $T_{res}^{-1}$ -template-image $I_{flo}$ ...
```

Alternatively, the vector field geometry can be given with both `'-template-dimension'` and `'-template-voxel'`.

Inverting the vector field transformations is done with an iterative procedure. Some options allows to tune related parameters.

1.12 pointmatching

`pointmatching` allows to compute a transformation from a list of paired points. It used the same computation methods, ie (weighted) least (trimmed) squares, and the same transformation classes than `blockmatching` . Basically, it uses the same routine than `blockmatching` (see section 1.1) to compute the incremental transformation δT from the block pairings. It may be useful to compute an initial transformation when the two images to be registered are too far apart.

Assume that the files P_{flo} and P_{ref} contain respectively the coordinates of points (i.e. each line is of the form '`x y z`', with one point per line, points being in real units) of respectively the floating and the reference images, I_{flo} and I_{ref}, the i^{th} point (i.e. corresponding to the i^{th} line of the file) of P_{flo} being paired to the i^{th} point of P_{ref}. The command

% `pointmatching -flo` P_{flo} `-ref` P_{ref}`-res-trsf` T_{init}

allows to compute the transformation T_{init} that can be either used to resample I_{flo} onto I_{ref} with

% `applyTrsf` I_{flo} $I_{res'}$ `-trsf` T_{init} `-template` I_{ref}

or that can be served as initialization for a subsequent registration

% `blockmatching -flo` I_{flo} `-ref` I_{ref} `-init-trsf` T_{init} `-res` I_{res} `-res-trsf` T_{res}

Note that the naming conventions (*floating* and *reference*) are coherent with those of `blockmatching` so that the obtained transformation can be used directly to resample the floating image (there is no need to compose the result transformation T_{res} with the initial one T_{int}), i.e. by the command

% `applyTrsf` I_{flo} I_{res} `-trsf` T `-template` I_{ref}

[**Pay attention**] When looking for transformations involving rigid transformations (i.e. rigid transformations or similitudes), it is mandatory to give point coordinates in real units (unless the voxel is isotropic). If points are known in voxel/pixel units, the voxel/pixel sizes may be specified either with the ad-hoc option or with a template image. It is assumed that the voxel/pixel sizes of both the reference I_{ref} and the floating image I_{flo} are the same.

Transformation estimation options are the same than the ones of `blockmatching` (refer to section 1.1.6.4 for generic options for transformation estimation, and section 1.1.6.5 for dedicated options for vector field estimation).

1.12.1 Vector field estimation

Vector field estimation within the context of `blockmatching` is done with a dense field of pairings, while only a sparse field of pairings can be passed (through the the files P_{flo} and P_{ref}) to `pointmatching` . Basically, the (dense) vector field estimation is done by interpolating the (sparse) pairings with a gaussian kernel whose standard deviation id given by the `'-fluid-sigma'` option.

When pairings are far apart (with respect to the standard deviation value), it (theoretically) comes to propagate the pairings except in medial areas where pairings will be interpolated. There are two possible drawbacks.

1. Small standard deviation values comes to deal with very small weight values (far away from the pairings), which may cause numerical instabilities.
2. Pairings are also propagated to the image borders, which may be an undesirable side effect.

To address these drawback, the `'-vector-propagation-distance'` and `'-vector-fading-distance'` options may help to more finely build the desired vector field, which is computed as follows:

1. `'-vector-propagation-distance %d'` propagates pairings at the distance given by the option (if not null), if possible (up to the extension of the Voronoï diagram of the pairings).
2. `'-vector-fading-distance %d'` also propagates the pairings at the distance given by the option, but with a fading effect, ensuring than identity (null vector pairing) are built far away (i.e. the sum of the two above distances) from the pairings.
3. The pairing field if then regularized by gaussian filtering (standard deviation given by the `'-fluid-sigma'` option).

Note that fading propagation and regularization may be required to get a regular dense vector field from sparse pairings. Tuning these parameters depends on both the sparsity of the pairings and the lengths of the displacement. It is then advised to conduct several experiments, using images created by `createGrid` for a convenient visualization of the produced vector fields (see section 3.1).

1.13 `printImage`

1.14 `printTrsf`

Chapter 2

Application Programming Interfaces

Some Application Programming Interfaces (APIs) have been written that mimics more or less the inline commands behavior: instead of specifying a number of control parameters (that may change because of code evolution), the user only have to give a pointer to a string (`char *`) containing the control parameters that he/she would have given in a inline command.

API procedure are named after the inline command, i.e. for the inline command `blockmatching` , the API procedure is named `API_blockmatching`.

[**Pay attention**] Typically, the inline command has only to deal with the I/O (reading the input structures, e.g. images or transformation, and writing the result structures after processing), while the processing takes place in the API procedure.

However, because of already existing interfaces, and to keep a backward compatibility, a more complex scheme may have to be built, see e.g. the `blockmatching` API in section 2.1.

2.1 `blockmatching` API

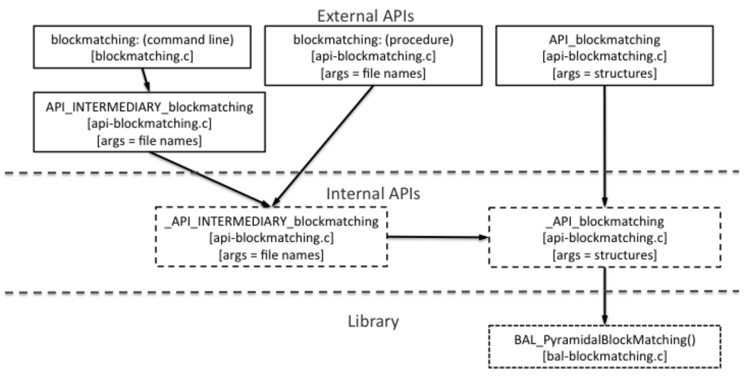


Figure 2.1: Organization of the blockmatching related APIs.

Part II

Examples of use

Chapter 3

Command Line Interfaces

3.1 `pointmatching`

We exemplify here the use of `pointmatching` to generate a vector field with two pairings $\{(35,35,0) \to (60,40,0), (65,60,0) \to (40,65,0)\}$, the first points being in the floating image while the second ones are in the reference image (see figure 3.1).

Test images can be generated with the following commands.

```
% createGrid -dim 100 100 mosaic.mha -type mosaic -spacing 10 10
% createGrid -dim 100 100 grid.mha -spacing 10 10 copy -norma
mosaic.mha mosaic.mha
```

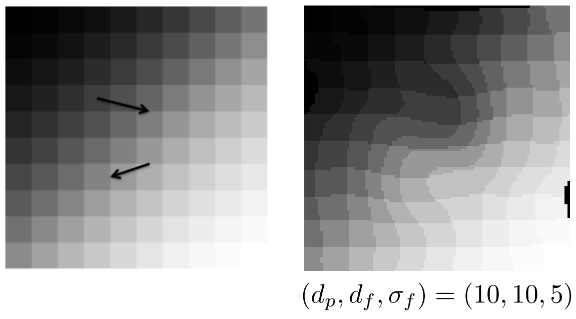


Figure 3.1: The two pairings superimposed on a mosaic test image, and the resampled image with the deformation computed with $(d_p, d_f, \sigma_f) = (10, 10, 5)$.

The options passed to `pointmatching` will be
`-vector-propagation-distance` d_p `-vector-fading-distance` d_f `-fluid-sigma` σ_f
where d_p, d_f, and σ_f denotes respectively the propagation distance (of the

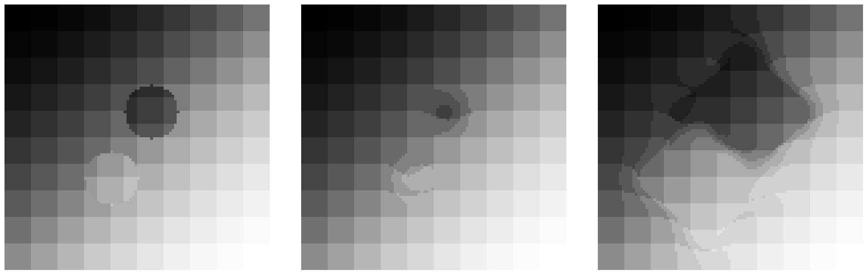
pairings), the fading distance (of the pairings, after the propagation), and the standard deviation for the regularization (with gaussian interpolation).

Thus, non-linear transformations are computed with different settings for (d_p, d_f, σ_f)

```
% pointmatching -flo floating.pts -ref reference.pts -trsf-type
vectorfield -template mosaic.mha -vector-propagation-distance $d_p$
-vector-fading-distance $d_f$ -fluid-sigma $\sigma_f$ -res-trsf vectorfield.trsf
```

and the floating image is resampled thanks to

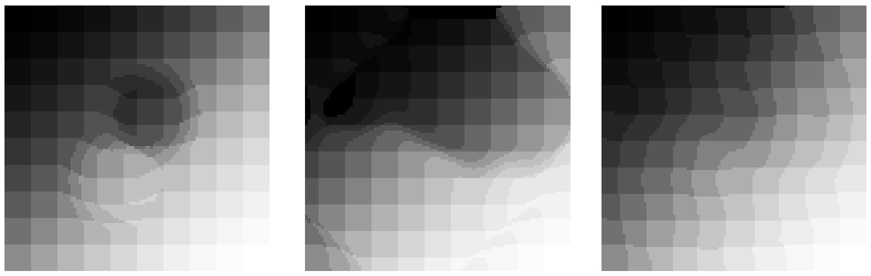
```
% applyTrsf mosaic.mha mosaic-result.mha -trsf vectorfield.trsf
-interpolation nearest
```



$(d_p, d_f, \sigma_f) = (10, 0, 0)$ $(d_p, d_f, \sigma_f) = (0, 10, 0)$ $(d_p, d_f, \sigma_f) = (0, 0, 5)$

Figure 3.2: Vector field deformation computation with only one non-null parameter.

On figure 3.2, it can be seen that deformations calculated with only one non-null parameter among (d_p, d_f, σ_f) lead to non-homotopic deformations. Note that using only σ_f should act as pure propagation (except at Voronoï diagram borders), but due to numerical reasons, the deformation becomes null when gaussian weights are too small.



$(d_p, d_f, \sigma_f) = (10, 10, 0)$ $(d_p, d_f, \sigma_f) = (10, 0, 5)$ $(d_p, d_f, \sigma_f) = (0, 10, 5)$

Figure 3.3: Vector field deformation computation with only one null parameter.

Using two non-null parameters (see figure 3.3) allows to get continuous deformations: e.g., $(d_p, d_f, \sigma_f) = (10, 0, 5)$ and $(d_p, d_f, \sigma_f) = (0, 10, 5)$. However, deformations may be large at the Voronoï diagram borders ($(d_p, d_f, \sigma_f) = (10, 0, 5)$) or resulting deformations may be quite far away the desired ones ($(d_p, d_f, \sigma_f) = (10, 0, 5)$).

Using all the three parameters may yield a deformation close to the expected one (see figure 3.1, left).

3.2 `applyTrsf` and `blockmatching`

We exemplified here how to compute a transformation at a lower resolution (also described in section 1.1.7.2) while still getting a deformed floating image at full resolution. Since the registration computation is done at a lower resolution, it is obviously faster (as it will be by tuning the `'-pyramid-lowest-level'`), but it also requires a little less computer memory. Such an approach can be used to tune parameters for instance.

Input images are of dimensions 481×481 with a pixel size of 0.5 image (see figure 3.4). They can be non-linearly registered through (registration result can be seen in figure 3.4, middle)

```
% blockmatching -ref ref.mha -flo flo.mha -res res.mha -trsf-type
vectorfield -res-trsf vector.trsf -elastic-sigma 2.5 -fluid-sigma 1.0
```

Options `'-elastic-sigma 2.5 -fluid-sigma 1.0'` have been chosen to enhance deformations.

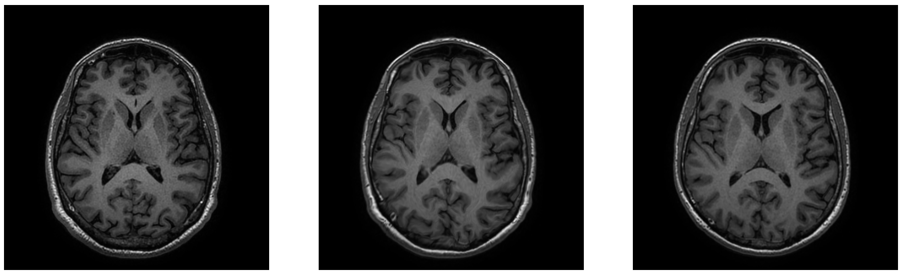


Figure 3.4: Input images for registration. Please notice the smaller ventricles of the floating image. Images dimensions are 481×481.

Input images are downsampled with the following commands (see figure 3.5). Note that, even here the commands are similar, they could have been resampled at different resolutions.

```
% applyTrsf flo.mha lower-flo.mha -iso 2.0 -resize -res-trsf
lower-flo.trsf
% applyTrsf ref.mha lower-ref.mha -iso 2.0 -resize -res-trsf
lower-ref.trsf
```

Here images are downsampled to a pixel size of 2.0, yielding images of dimensions 120×120. These images can be non-linearly registered through (registration result can be seen in figure 3.5, middle)

```
% blockmatching -ref lower-ref.mha -flo lower-flo.mha -res
lower-res.mha -trsf-type vectorfield -res-trsf lower-vector.trsf
-elastic-sigma 2.5 -fluid-sigma 1.0
```

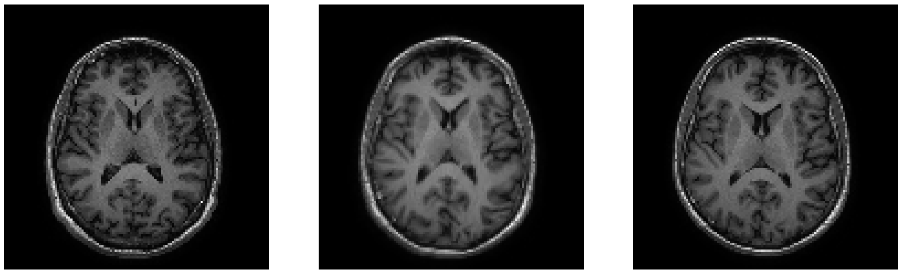


Figure 3.5: Downsampled images and registration result. Images dimensions are 160×160.

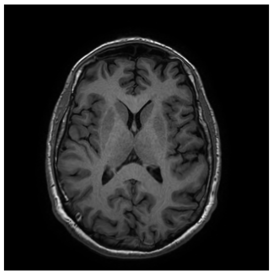
To get the registration result at the original resolution, there are different possibilities.

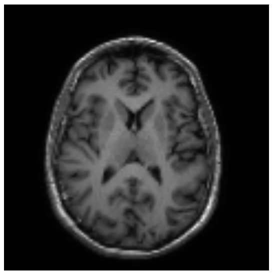
1. The low resolution registration result can be upsampled in the same geometry than the reference image. To that end, the downsampling matrix (of the reference image) has first to be inverted. Then, the low resolution registration result is resampled with this inverted transformation. It can be achieved with

   ```
   % invTrsf lower-ref.trsf inv-lower-ref.trsf
   % applyTrsf lower-res.mha resampled-lower-res.mha -trsf
   inv-lower-ref.trsf -template ref.mha
   ```

   However, even though the upsampled low resolution registration result is of dimensions 481×481 with a pixel size of 0.5, the downsampling is visually present (see figure 3.6).

2. A deformation at the high resolution can be computed from the deformation at the low resolution. It comes to compose transformations (see section 1.1.7.2). The floating image at the high resolution can be then resampled thanks to this transformation. It can be done thanks to


High resolution result
481 × 481


Upsampled low resolution result
481 × 481

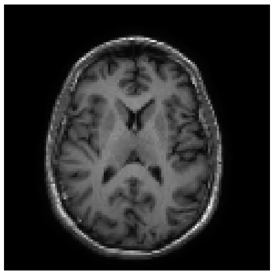
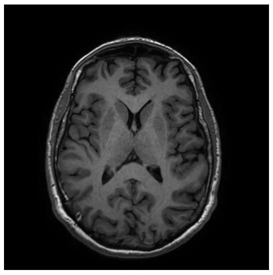

Low resolution result
160 × 160

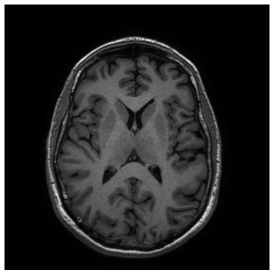
Figure 3.6: Upsampling the low resolution registration result.

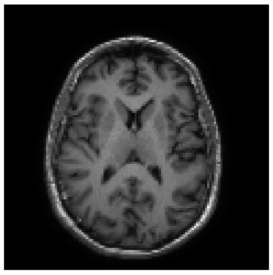
```
% invTrsf lower-ref.trsf inv-lower-ref.trsf
% composeTrsf resampled-lower-vector.trsf -trsfs lower-flo.trsf
lower-vector.trsf inv-lower-ref.trsf -template ref.mha
% applyTrsf flo.mha lower-trsf-res.mha -trsf resampled-lower-vector.trsf
-template ref.mha
```

The result can be see in figure 3.7.


High resolution result

481 × 481


High resolution result
from low resolution transformation
481 × 481


Low resolution result

160 × 160

Figure 3.7: Upsampling the low resolution registration result.

Part III

Methodological notes

Chapter 4

Frames

4.1 The digital world ($\mathbb{Z}$)

Pixels or voxels are defined over $\mathbb{Z}^2$ or $\mathbb{Z}^3$ and have integer coordinates.

The voxel is a small rectangular cuboid that has a spatial extent. Our conventions are that the voxel coordinates design the center of the voxel. Let (v_x, v_y, v_z) be the voxel size of image I, the spatial area of a voxel $M_{\mathbb{R}} = (x, y, z)$ in the real world $\mathbb{R}$ is the cuboid $[x - v_x/2, x + v_x/2] \times [y - v_y/2, y + v_y/2] \times [z - v_z/2, z + v_z/2]$ (see figure 4.1).

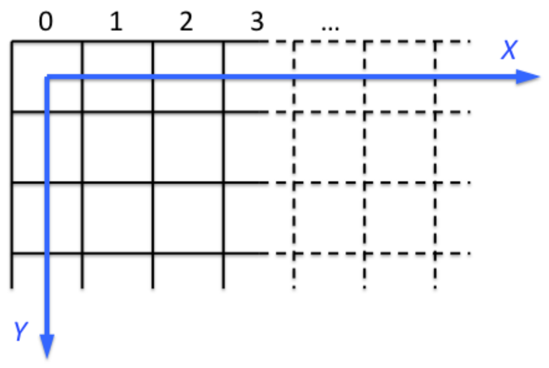


Figure 4.1: Definition of the field of view with respect to the *voxel* one. The origin of the *real* frame is at the center of the upper left pixel/voxel, here the one of coordinate $(0, 0, \ldots)$ (C/C++ conventions).

Let us consider an image that has a voxel size v_x and a number of voxels of d_x along the X direction. From the C/C++ conventions, the d_x voxels have coordinates in $\mathbb{Z}$ from 0 to $d_x - 1$. The length of the field of view (FOV) is obviously $v_x d_x$ and spans the interval $[v_x * 0 - v_x/2, v_x * (d_x - 1) + v_x/2] =$

$[-v_x/2, v_x d_x - v_x/2]$ in $\mathbb{R}$.

As a consequence, the center of the field of view in $\mathbb{Z}$ is (here in 3D)

$$C_{\mathbb{Z}} = \begin{pmatrix} \frac{(d_x-1)}{2} \\ \frac{(d_y-1)}{2} \\ \frac{(d_z-1)}{2} \end{pmatrix}$$

Converting voxel coordinates into real ones can trivially be done by multiplying by the voxel sizes (see section 4.2). The voxel with zero's coordinates defines then the origin of the *real* frame.

4.2 Image: $\mathbb{R} \leftrightarrow \mathbb{Z}$

An image I is defined over the discrete frame $\mathbb{Z}$ and a point M may be defined either by its 'voxel' coordinates (i, j, k), or by 'real' coordinates (x, y, z) that can be deduced from (i, j, k) thanks to the imaging acquisition information.

Conversion from the voxel frame, denoted $\mathbb{Z}$, to the real one, denoted $\mathbb{R}$, is achieved through conversion matrices that are associated with every image I.

Without any other specification, there are diagonal matrices with the voxel sizes (or their inverses) along the diagonal. Let (v_x, v_y, v_z) be the voxel size of image I, thus a point $M_{\mathbb{R}} = (x, y, z)$ in the real frame $\mathbb{R}$ correspond to the voxel point $M_{\mathbb{Z}} = (i, j, k)$ in the voxel frame $\mathbb{Z}$ with

$$M_{\mathbb{R}} = \mathbf{H}_{I,\mathbb{R}\leftarrow\mathbb{Z}} M_{\mathbb{Z}} \quad \text{with} \quad \mathbf{H}_{I,\mathbb{R}\leftarrow\mathbb{Z}} = \begin{pmatrix} v_x & . & . & . \\ . & v_y & . & . \\ . & . & v_z & . \\ . & . & . & 1 \end{pmatrix}$$

Accordingly,

$$M_{\mathbb{Z}} = \mathbf{H}_{I,\mathbb{Z}\leftarrow\mathbb{R}} M_{\mathbb{R}} \quad \text{with} \quad \mathbf{H}_{I,\mathbb{Z}\leftarrow\mathbb{R}} = \begin{pmatrix} 1/v_x & . & . & . \\ . & 1/v_y & . & . \\ . & . & 1/v_z & . \\ . & . & . & 1 \end{pmatrix}$$

Obviously, we have $\mathbf{H}_{I,\mathbb{Z}\leftarrow\mathbb{R}} = \mathbf{H}^{-1}_{I,\mathbb{R}\leftarrow\mathbb{Z}}$. Note that we use homogeneous coordinates (see section 5.2) to define these matrices.

[Pay attention] With such a default convention, the frame origins in both the real and the voxel frames superimpose. However, the frame origin in the real frame is not the upper left corner of the field of view (see figure 4.1).

Note that the matrix $\mathbf{H}$ can be any linear matrix. E.g. they can be re-orientation matrices that allows to map the voxel array in standard radiology conventions [1].

[1]Nifti and Inrimage image formats can embed such rigid transformations (the so-called qform matrices in Nifti format). However, such information can be lost when converting to other formats.

Chapter 5

Transformations

5.1 Definitions

5.1.1 Linear transformations

Classically, linear transformations (i.e. translations, rigid transformations, affine transformations, etc.) are expressed as combinations of a 3×3 matrix (representing the vectorial part of the transformation) $\mathbf{A}$ and a 3D vector $\mathbf{t}$ (representing the translation). Thus, the transformed point M' of M by the transformation $T = (\mathbf{A}, \mathbf{t})$ is expressed by

$$M' = T(M) = \mathbf{A}M + \mathbf{t}$$

Note that linear transformations are mainly expressed in homogeneous coordinates all along this document (see section 5.2.1), thus we will use the somewhat abusive notation

$$M' = \mathbf{T}M$$

where $\mathbf{T}$ is a 4×4 matrix.

5.1.2 Vector fields

Vector fields are used to encode non-linear transformations. For a transformation T, the vector $\mathbf{v}(M)$ at point M is the displacement of the point M, meaning that

$$M' = T(M) = M + \mathbf{v}(M)$$

5.2 Homogeneous coordinates

4D homogeneous coordinates are implicitly used. However, there are some with transformations expressed as vector fields.

5.2.1 Linear transformations

It is more convenient to express the linear transformations as 4×4 matrices embedding the translation, so combinations of linear transformations can be expressed by multiplications of matrices. Such a 4×4 matrix $\mathbf{T}$ is designed by

$$\mathbf{T} = \left(\begin{array}{ccc|c} & \mathbf{A} & & \mathbf{t} \\ \hline 0 & 0 & 0 & 1 \end{array} \right)$$

A point M has then implicitly 4 coordinates, the first three ones being the spatial coordinates (x, y, z) and the last one being 1. We have thus

$$M' = T(M) = \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \mathbf{T}M = \left(\begin{array}{ccc|c} & \mathbf{A} & & \mathbf{t} \\ \hline 0 & 0 & 0 & 1 \end{array} \right) \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

5.2.2 Vector fields

The transformation with a vector field is encoded by

$$M' = M + \mathbf{v}(M)$$

When composing (at left) with a linear transformation, we have

$$\begin{aligned} M'' &= \mathbf{A}M' + \mathbf{t} \\ &= \mathbf{A}M + \mathbf{A}\mathbf{v}(M) + \mathbf{t} \end{aligned} \tag{5.1}$$

If homogeneous coordinates are used, it comes

$$\begin{aligned} M'' &= \mathbf{T}M' \, (= \mathbf{A}M' + \mathbf{t}) \\ &= \mathbf{T}M + \mathbf{T}\mathbf{v}(M) \end{aligned}$$

From equation 5.1, it comes that

$$\mathbf{T}\mathbf{v}(M) = \mathbf{A}\mathbf{v}(M)$$

This is compatible with homogeneous coordinates if the fourth coordinate of $\mathbf{v}$ is 0, indeed

$$\mathbf{T}\mathbf{v} = \left(\begin{array}{ccc|c} & \mathbf{A} & & \mathbf{t} \\ \hline 0 & 0 & 0 & 1 \end{array} \right) \begin{pmatrix} \mathbf{v}_x \\ \mathbf{v}_y \\ \mathbf{v}_z \\ 0 \end{pmatrix} = \left(\begin{array}{c} \mathbf{A}\mathbf{v} \\ \hline 0 \end{array} \right)$$

5.3 'voxel' versus 'real' definitions

Recall that an image I is defined over the discrete frame $\mathbb{Z}$. Thus a point M of I can be expressed either in the discrete frame in voxel coordinates (and will be denoted $M_{\mathbb{Z}}$) or in real coordinates (and will be denoted $M_{\mathbb{R}}$). Conversion matrices (see section 4.2) allow to go from the voxel frame to the real one, and conversely.

A transformation $T_{flo \leftarrow ref}$ from an image I_{ref} towards an image I_{flo} can be then either defined in the voxel frame or in the real one. These transformations are denoted respectively $T_{flo \leftarrow ref, \mathbb{Z}}$ and $T_{flo \leftarrow ref, \mathbb{R}}$:

$$\begin{array}{rl} T_{flo \leftarrow ref, \mathbb{Z}} : & I_{ref} \rightarrow I_{flo} \\ & M_{ref, \mathbb{Z}} \mapsto M_{flo, \mathbb{Z}} \end{array}$$

and

$$\begin{array}{rl} T_{flo \leftarrow ref, \mathbb{R}} : & I_{ref} \rightarrow I_{flo} \\ & M_{ref, \mathbb{R}} \mapsto M_{flo, \mathbb{R}} \end{array}$$

We have then

$$\begin{array}{rcl} M_{flo, \mathbb{Z}} & = & T_{flo \leftarrow ref, \mathbb{Z}}(M_{ref, \mathbb{Z}}) \\ M_{flo, \mathbb{R}} & = & T_{flo \leftarrow ref, \mathbb{R}}(M_{ref, \mathbb{R}}) \end{array}$$

Transformations in 'voxel' coordinates are required for image resampling, while transformations in 'real' coordinates may be necessary when dealing when some transformation classes (e.g. rigid) or to compute some measurements (e.g. mean displacements) in world units. Thus, conversion of these transformations from the voxel world to the real one, using the image conversion matrices (see section 4.2), has to be made explicit.

5.3.1 Linear transformations

Linear transformations can be expressed by $4x4$ matrices in homogeneous coordinates (see section 5.2.1), thus we have

$$\begin{array}{rcl} M_{flo, \mathbb{Z}} & = & T_{flo \leftarrow ref, \mathbb{Z}} M_{ref, \mathbb{Z}} \\ M_{flo, \mathbb{R}} & = & T_{flo \leftarrow ref, \mathbb{R}} M_{ref, \mathbb{R}} \end{array}$$

To convert the transformations from the voxel world to the real one (and conversely), we apply the composition rules that are here simply matrices multiplication:

$$\mathbf{T}_{flo \leftarrow ref, \mathbb{Z}} = \mathbf{H}_{flo, \mathbb{Z} \leftarrow \mathbb{R}} \circ \mathbf{T}_{flo \leftarrow ref, \mathbb{R}} \circ \mathbf{H}_{ref, \mathbb{R} \leftarrow \mathbb{Z}}$$

and

$$\mathbf{T}_{flo \leftarrow ref, \mathbb{R}} = \mathbf{H}_{flo, \mathbb{R} \leftarrow \mathbb{Z}} \circ \mathbf{T}_{flo \leftarrow ref, \mathbb{Z}} \circ \mathbf{H}_{ref, \mathbb{Z} \leftarrow \mathbb{R}}$$

5.3.2 Vector fields

The difficulty comes from the fact that the non-linear transformation is encoded by a displacement/vector field that is defined over a discrete lattice (i.e. an image). We have then

$$\begin{aligned}
M_{flo,\mathbb{Z}} &= T_{flo\leftarrow ref,\mathbb{Z}}(M_{ref,\mathbb{Z}}) \\
&= M_{ref,\mathbb{Z}} + \mathbf{v}_{flo\leftarrow ref,\mathbb{Z}}(M_{ref,\mathbb{Z}}) \\
M_{flo,\mathbb{R}} &= T_{flo\leftarrow ref,\mathbb{R}}(M_{ref,\mathbb{R}}) \\
&= M_{ref,\mathbb{R}} + \mathbf{v}_{flo\leftarrow ref,\mathbb{R}}(M_{ref,\mathbb{Z}}) \\
&= M_{ref,\mathbb{R}} + \mathbf{v}_{flo\leftarrow ref,\mathbb{R}}(\mathbf{H}_{ref,\mathbb{Z}\leftarrow\mathbb{R}} M_{ref,\mathbb{R}})
\end{aligned}$$

where $\mathbf{H}_{ref,\mathbb{Z}\leftarrow\mathbb{R}}$ denotes the conversion matrix from the real world $\mathbb{R}$ to the discrete world $\mathbb{Z}$ for image I_{ref}.

The vector field $\mathbf{v}$ indicates the displacement at every point. For a transformation from image I_{ref} to image I_{flo}, this vector is defined on the same frame than I_{ref}. Thus the vector in *real* coordinates $\mathbf{v}_{flo\leftarrow ref,\mathbb{R}}$ at $M_\mathbb{R}$ gives the displacement of the point $M_\mathbb{R}$.

However, since $\mathbf{v}_{flo\leftarrow ref}$ is defined over a discrete lattice, the vector image stores the vectors $\mathbf{v}_\mathbb{R}(M_\mathbb{Z})$, thus $\mathbf{v}_\mathbb{R}$ at $M_\mathbb{R}$ is $\mathbf{v}_\mathbb{R}(\mathbf{H}_{\mathbb{Z}\leftarrow\mathbb{R}} M_\mathbb{R})$.

5.3.2.1 Vector fields: $\mathbb{R} \rightarrow \mathbb{Z}$

The transformation $\mathbf{T}_{flo\leftarrow ref,\mathbb{R}}$ from image I_{ref} to image I_{flo} is defined by the vector field $\mathbf{v}_{flo\leftarrow ref,\mathbb{R}}$

$$\begin{aligned}
M_{flo,\mathbb{R}} &= \mathbf{T}(M_{ref,\mathbb{R}}) = M_{ref,\mathbb{R}} + \mathbf{v}_{flo\leftarrow ref,\mathbb{R}}(\mathbf{H}_{ref,\mathbb{Z}\leftarrow\mathbb{R}} M_{ref,\mathbb{R}}) \\
&= \mathbf{T}(M_{ref,\mathbb{R}}) = M_{ref,\mathbb{R}} + \mathbf{v}_{flo\leftarrow ref,\mathbb{R}}(M_{ref,\mathbb{Z}})
\end{aligned}$$

When expressing this formula in the discrete lattice, it comes

$$\mathbf{H}_{flo,\mathbb{R}\leftarrow\mathbb{Z}} M_{flo,\mathbb{Z}} = \mathbf{H}_{ref,\mathbb{R}\leftarrow\mathbb{Z}} M_{ref,\mathbb{Z}} + \mathbf{v}_{flo\leftarrow ref,\mathbb{R}}(M_{ref,\mathbb{Z}})$$

$$M_{flo,\mathbb{Z}} = \mathbf{H}^{-1}_{flo,\mathbb{R}\leftarrow\mathbb{Z}} \circ \mathbf{H}_{ref,\mathbb{R}\leftarrow\mathbb{Z}} M_{ref,\mathbb{Z}} + \mathbf{H}^{-1}_{flo,\mathbb{R}\leftarrow\mathbb{Z}} \circ \mathbf{v}_{flo\leftarrow ref,\mathbb{R}}(M_{ref,\mathbb{Z}})$$

The displacement in *voxel* coordinates $\mathbf{v}_{flo\leftarrow ref,\mathbb{Z}}$ (associated to $\mathbf{T}_{flo\leftarrow ref,\mathbb{Z}}$) is then defined by

$$\begin{aligned}
&\mathbf{v}_{flo\leftarrow ref,\mathbb{Z}}(M_{ref,\mathbb{Z}}) = M_{flo,\mathbb{Z}} - M_{ref,\mathbb{Z}} \\
&= \left(\mathbf{H}^{-1}_{flo,\mathbb{R}\leftarrow\mathbb{Z}} \circ \mathbf{H}_{ref,\mathbb{R}\leftarrow\mathbb{Z}} - \mathbf{Id}\right) M_{ref,\mathbb{Z}} + \mathbf{H}^{-1}_{flo,\mathbb{R}\leftarrow\mathbb{Z}} \circ \mathbf{v}_{flo\leftarrow ref,\mathbb{R}}(M_{ref,\mathbb{Z}}) \\
&= \left(\mathbf{H}_{flo,\mathbb{Z}\leftarrow\mathbb{R}} \circ \mathbf{H}_{ref,\mathbb{R}\leftarrow\mathbb{Z}} - \mathbf{Id}\right) M_{ref,\mathbb{Z}} + \mathbf{H}_{flo,\mathbb{Z}\leftarrow\mathbb{R}} \circ \mathbf{v}_{flo\leftarrow ref,\mathbb{R}}(M_{ref,\mathbb{Z}})
\end{aligned}$$

(where $\mathbf{Id}$ denotes the identity matrix) and may be different from a simple scaling of the vector field defined in *real* coordinates.

5.3.2.2 Vector fields: $\mathbb{Z} \rightarrow \mathbb{R}$

We have

$$M_{flo,\mathbb{Z}} = M_{ref,\mathbb{Z}} + \mathbf{v}_{flo\leftarrow ref,\mathbb{Z}}(M_{ref,\mathbb{Z}})$$

When expressing this formula in the real world, it comes

$$\mathbf{H}_{flo,\mathbb{Z}\leftarrow\mathbb{R}} M_{flo,\mathbb{R}} = \mathbf{H}_{ref,\mathbb{Z}\leftarrow\mathbb{R}} M_{ref,\mathbb{R}} + \mathbf{v}_{flo\leftarrow ref,\mathbb{Z}}(M_{ref,\mathbb{Z}})$$

$$M_{flo,\mathbb{R}} = \mathbf{H}^{-1}_{flo,\mathbb{Z}\leftarrow\mathbb{R}} \circ \mathbf{H}_{ref,\mathbb{Z}\leftarrow\mathbb{R}} M_{ref,\mathbb{R}} + \mathbf{H}^{-1}_{flo,\mathbb{Z}\leftarrow\mathbb{R}} \circ \mathbf{v}_{flo\leftarrow ref,\mathbb{Z}}(M_{ref,\mathbb{Z}})$$

The displacement in *real* coordinates $\mathbf{v}_{flo\leftarrow ref,\mathbb{R}}$ is then defined by

$$\begin{aligned}
&\mathbf{v}_{flo\leftarrow ref,\mathbb{R}}(M_{ref,\mathbb{Z}}) = M_{flo,\mathbb{R}} - M_{ref,\mathbb{R}} \\
&= \left(\mathbf{H}^{-1}_{flo,\mathbb{Z}\leftarrow\mathbb{R}} \circ \mathbf{H}_{ref,\mathbb{Z}\leftarrow\mathbb{R}} - \mathbf{Id}\right) M_{ref,\mathbb{R}} + \mathbf{H}^{-1}_{flo,\mathbb{Z}\leftarrow\mathbb{R}} \circ \mathbf{v}_{flo\leftarrow ref,\mathbb{Z}}(M_{ref,\mathbb{Z}}) \\
&= \left(\mathbf{H}^{-1}_{flo,\mathbb{Z}\leftarrow\mathbb{R}} \circ \mathbf{H}_{ref,\mathbb{Z}\leftarrow\mathbb{R}} - \mathbf{Id}\right) \mathbf{H}_{ref,\mathbb{R}\leftarrow\mathbb{Z}} M_{ref,\mathbb{Z}} + \mathbf{H}_{flo,\mathbb{R}\leftarrow\mathbb{Z}} \circ \mathbf{v}_{flo\leftarrow ref,\mathbb{Z}}(M_{ref,\mathbb{Z}}) \\
&= \left(\mathbf{H}_{flo,\mathbb{R}\leftarrow\mathbb{Z}} - \mathbf{H}_{ref,\mathbb{R}\leftarrow\mathbb{Z}}\right) M_{ref,\mathbb{Z}} + \mathbf{H}_{flo,\mathbb{R}\leftarrow\mathbb{Z}} \circ \mathbf{v}_{flo\leftarrow ref,\mathbb{Z}}(M_{ref,\mathbb{Z}})
\end{aligned}$$

5.4 Transformations: linear $\rightarrow$ vector field

$\mathbf{T}_{flo\leftarrow ref,\mathbb{R}}$ is a linear transformation in real space. We have then

$$M_{flo,\mathbb{R}} = \mathbf{T}_{flo\leftarrow ref,\mathbb{R}} M_{ref,\mathbb{R}}$$

For vector fields, the transformation is expressed by

$$M_{flo,\mathbb{R}} = M_{ref,\mathbb{R}} + \mathbf{v}_{flo\leftarrow ref,\mathbb{R}}(M_{ref,\mathbb{Z}})$$

We have then

$$\begin{aligned}
\mathbf{v}_{flo\leftarrow ref,\mathbb{R}}(M_{ref,\mathbb{Z}}) &= M_{flo,\mathbb{R}} - M_{ref,\mathbb{R}} \\
&= \mathbf{T}_{flo\leftarrow ref,\mathbb{R}} M_{ref,\mathbb{R}} - M_{ref,\mathbb{R}} \\
&= (\mathbf{T}_{flo\leftarrow ref,\mathbb{R}} - \mathbf{Id}) \circ \mathbf{H}_{ref,\mathbb{R}\leftarrow\mathbb{Z}} M_{ref,\mathbb{Z}} \\
&= (\mathbf{H}_{flo,\mathbb{R}\leftarrow\mathbb{Z}} \circ \mathbf{T}_{flo\leftarrow ref,\mathbb{Z}} \circ \mathbf{H}_{ref,\mathbb{Z}\leftarrow\mathbb{R}} - \mathbf{Id}) \circ \mathbf{H}_{ref,\mathbb{R}\leftarrow\mathbb{Z}} M_{ref,\mathbb{Z}}
\end{aligned}$$

Remarks:

- Estimating the vector field in real/world units $\mathbf{v}_{flo\leftarrow ref,\mathbb{R}}$ from a linear transformation in real/world units $\mathbf{T}_{flo\leftarrow ref,\mathbb{Z}}$ is done with

$$(\mathbf{T}_{flo\leftarrow ref,\mathbb{R}} - \mathbf{Id}) \circ \mathbf{H}_{ref,\mathbb{R}\leftarrow\mathbb{Z}} M_{ref,\mathbb{Z}}$$

  Note that the voxel-to-world transformation for the reference image/frame $\mathbf{H}_{ref,\mathbb{R}\leftarrow\mathbb{Z}}$ is also embedded in the non-linear transformation $\mathbf{v}_{flo\leftarrow ref,\mathbb{R}}$.

- Estimating the vector field in real/world units $\mathbf{v}_{flo \leftarrow ref,\mathbb{R}}$ from a linear transformation in voxel units $\mathbf{T}_{flo \leftarrow ref,\mathbb{Z}}$

  $$\begin{aligned}&\mathbf{v}_{flo \leftarrow ref,\mathbb{R}}(M_{ref,\mathbb{Z}}) = \\ &\qquad (\mathbf{H}_{flo,\mathbb{R} \leftarrow \mathbb{Z}} \circ \mathbf{T}_{flo \leftarrow ref,\mathbb{Z}} \circ \mathbf{H}_{ref,\mathbb{Z} \leftarrow \mathbb{R}} - \mathbf{Id}) \circ \mathbf{H}_{ref,\mathbb{R} \leftarrow \mathbb{Z}} M_{ref,\mathbb{Z}}\end{aligned}$$

  requires to have the voxel-to-world transformation for the floating image/frame $\mathbf{H}_{flo,\mathbb{R} \leftarrow \mathbb{Z}}$ at hand.

- When $\mathbf{T}_{flo \leftarrow ref,\mathbb{R}}$ is a pure translation $\mathbf{t}_{\mathbb{R}}$, meaning that $M_{flo,\mathbb{R}} = \mathbf{Id} M_{ref,\mathbb{R}} + \mathbf{t}_{\mathbb{R}}$, we have then

  $$\mathbf{v}_{flo \leftarrow ref,\mathbb{R}}(M_{ref,\mathbb{Z}}) = \mathbf{t}_{\mathbb{R}}$$

5.5 Transformation composition

We simply use the composition rules to transformation composition.

$$\mathbf{T}_{K \leftarrow I,\mathbb{R}} = \mathbf{T}_{K \leftarrow J,\mathbb{R}} \circ \mathbf{T}_{J \leftarrow I,\mathbb{R}}$$

and

$$\mathbf{T}_{K \leftarrow I,\mathbb{Z}} = \mathbf{T}_{K \leftarrow J,\mathbb{Z}} \circ \mathbf{T}_{J \leftarrow I,\mathbb{Z}}$$

5.5.1 Linear $\circ$ linear

The composition is quite trivial, since it only uses matrices multiplication.

5.5.2 Linear $\circ$ vector field in $\mathbb{R}$

$$\begin{aligned}M_{J,\mathbb{R}} &= M_{I,\mathbb{R}} + \mathbf{v}_{J \leftarrow I,\mathbb{R}}(\mathbf{H}_{I,\mathbb{Z} \leftarrow \mathbb{R}} M_{I,\mathbb{R}}) \\ &= \mathbf{H}_{I,\mathbb{R} \leftarrow \mathbb{Z}} M_{I,\mathbb{Z}} + \mathbf{v}_{J \leftarrow I,\mathbb{R}}(M_{I,\mathbb{Z}}) \\ M_{K,\mathbb{R}} &= \mathbf{T}_{K \leftarrow J,\mathbb{R}} M_{J,\mathbb{R}} \\ &= \mathbf{T}_{K \leftarrow J,\mathbb{R}} \mathbf{H}_{I,\mathbb{R} \leftarrow \mathbb{Z}} M_{I,\mathbb{Z}} + \mathbf{T}_{K \leftarrow J,\mathbb{R}} \mathbf{v}_{J \leftarrow I,\mathbb{R}}(M_{I,\mathbb{Z}})\end{aligned}$$

$$\begin{aligned}\mathbf{v}_{K \leftarrow I,\mathbb{R}}(M_{I,\mathbb{Z}}) &= M_{K,\mathbb{R}} - M_{I,\mathbb{R}} \\ &= (\mathbf{T}_{K \leftarrow J,\mathbb{R}} - \mathbf{Id}) \circ \mathbf{H}_{I,\mathbb{R} \leftarrow \mathbb{Z}} M_{I,\mathbb{Z}} + \mathbf{T}_{K \leftarrow J,\mathbb{R}} \mathbf{v}_{J \leftarrow I,\mathbb{R}}(M_{I,\mathbb{Z}})\end{aligned}$$

5.5.3 Linear $\circ$ vector field in $\mathbb{Z}$

$$\begin{aligned}M_{J,\mathbb{Z}} &= M_{I,\mathbb{Z}} + \mathbf{v}_{J \leftarrow I,\mathbb{Z}}(M_{I,\mathbb{Z}}) \\ M_{K,\mathbb{Z}} &= \mathbf{T}_{K \leftarrow J,\mathbb{Z}} M_{J,\mathbb{Z}} \\ &= \mathbf{T}_{K \leftarrow J,\mathbb{Z}} M_{I,\mathbb{Z}} + \mathbf{T}_{K \leftarrow J,\mathbb{Z}} \mathbf{v}_{J \leftarrow I,\mathbb{Z}}(M_{I,\mathbb{Z}})\end{aligned}$$

$$\begin{aligned}\mathbf{v}_{K \leftarrow I,\mathbb{Z}}(M_{I,\mathbb{Z}}) &= M_{K,\mathbb{Z}} - M_{I,\mathbb{Z}} \\ &= (\mathbf{T}_{K \leftarrow J,\mathbb{Z}} - \mathbf{Id})\, M_{I,\mathbb{Z}} + \mathbf{T}_{K \leftarrow J,\mathbb{Z}} \mathbf{v}_{J \leftarrow I,\mathbb{Z}}(M_{I,\mathbb{Z}})\end{aligned}$$

5.5.4 Vector field $\circ$ linear in $\mathbb{R}$

$$\begin{aligned}
M_{J,\mathbb{R}} &= \mathbf{T}_{J\leftarrow I,\mathbb{R}} M_{I,\mathbb{R}} \\
M_{K,\mathbb{R}} &= M_{J,\mathbb{R}} + \mathbf{v}_{K\leftarrow J,\mathbb{R}}(\mathbf{H}_{J,\mathbb{Z}\leftarrow\mathbb{R}} M_{J,\mathbb{R}}) \\
&= \mathbf{T}_{J\leftarrow I,\mathbb{R}} M_{I,\mathbb{R}} + \mathbf{v}_{K\leftarrow J,\mathbb{R}}(\mathbf{H}_{J,\mathbb{Z}\leftarrow\mathbb{R}} \mathbf{T}_{J\leftarrow I,\mathbb{R}} M_{I,\mathbb{R}})
\end{aligned}$$

$$\begin{aligned}
\mathbf{v}_{K\leftarrow I,\mathbb{R}}(M_{I,\mathbb{Z}}) &= M_{K,\mathbb{R}} - M_{I,\mathbb{R}} \\
&= (\mathbf{T}_{J\leftarrow I,\mathbb{R}} - \mathbf{Id}) \circ \mathbf{H}_{I,\mathbb{R}\leftarrow\mathbb{Z}} M_{I,\mathbb{Z}} \\
&\quad + \mathbf{v}_{K\leftarrow J,\mathbb{R}}(\mathbf{H}_{J,\mathbb{Z}\leftarrow\mathbb{R}} \mathbf{T}_{J\leftarrow I,\mathbb{R}} \mathbf{H}_{I,\mathbb{R}\leftarrow\mathbb{Z}} M_{I,\mathbb{Z}})
\end{aligned}$$

5.5.5 Vector field $\circ$ linear in $\mathbb{Z}$

$$\begin{aligned}
M_{J,\mathbb{Z}} &= \mathbf{T}_{J\leftarrow I,\mathbb{Z}} M_{I,\mathbb{Z}} \\
M_{K,\mathbb{Z}} &= M_{J,\mathbb{Z}} + \mathbf{v}_{K\leftarrow J,\mathbb{Z}}(M_{J,\mathbb{Z}}) \\
&= \mathbf{T}_{J\leftarrow I,\mathbb{Z}} M_{I,\mathbb{Z}} + \mathbf{v}_{K\leftarrow J,\mathbb{Z}}(\mathbf{T}_{J\leftarrow I,\mathbb{Z}} M_{I,\mathbb{Z}})
\end{aligned}$$

$$\begin{aligned}
\mathbf{v}_{K\leftarrow I,\mathbb{Z}}(M_{I,\mathbb{Z}}) &= M_{K,\mathbb{Z}} - M_{I,\mathbb{Z}} \\
&= (\mathbf{T}_{J\leftarrow I,\mathbb{Z}} - \mathbf{Id})\, M_{I,\mathbb{Z}} + \mathbf{v}_{K\leftarrow J,\mathbb{Z}}(\mathbf{T}_{J\leftarrow I,\mathbb{Z}} M_{I,\mathbb{Z}})
\end{aligned}$$

5.5.6 Vector field $\circ$ vector field in $\mathbb{R}$

$$\begin{aligned}
M_{J,\mathbb{R}} &= M_{I,\mathbb{R}} + \mathbf{v}_{J\leftarrow I,\mathbb{R}}(\mathbf{H}_{I,\mathbb{Z}\leftarrow\mathbb{R}} M_{I,\mathbb{R}}) \\
&= \mathbf{H}_{I,\mathbb{R}\leftarrow\mathbb{Z}} M_{I,\mathbb{Z}} + \mathbf{v}_{J\leftarrow I,\mathbb{R}}(M_{I,\mathbb{Z}}) \\
M_{K,\mathbb{R}} &= M_{J,\mathbb{R}} + \mathbf{v}_{K\leftarrow J,\mathbb{R}}(\mathbf{H}_{J,\mathbb{Z}\leftarrow\mathbb{R}} M_{J,\mathbb{R}}) \\
&= M_{I,\mathbb{R}} + \mathbf{v}_{J\leftarrow I,\mathbb{R}}(M_{I,\mathbb{Z}}) \\
&\quad + \mathbf{v}_{K\leftarrow J,\mathbb{R}}\left(\mathbf{H}_{J,\mathbb{Z}\leftarrow\mathbb{R}}\left(\mathbf{H}_{I,\mathbb{R}\leftarrow\mathbb{Z}} M_{I,\mathbb{Z}} + \mathbf{v}_{J\leftarrow I,\mathbb{R}}(M_{I,\mathbb{Z}})\right)\right)
\end{aligned}$$

$$\begin{aligned}
\mathbf{v}_{K\leftarrow I,\mathbb{R}}(M_{I,\mathbb{Z}}) &= M_{K,\mathbb{R}} - M_{I,\mathbb{R}} \\
&= \mathbf{v}_{J\leftarrow I,\mathbb{R}}(M_{I,\mathbb{Z}}) \\
&\quad + \mathbf{v}_{K\leftarrow J,\mathbb{R}}\left(\mathbf{H}_{J,\mathbb{Z}\leftarrow\mathbb{R}}\left(\mathbf{H}_{I,\mathbb{R}\leftarrow\mathbb{Z}} M_{I,\mathbb{Z}} + \mathbf{v}_{J\leftarrow I,\mathbb{R}}(M_{I,\mathbb{Z}})\right)\right) \\
&= \mathbf{v}_{J\leftarrow I,\mathbb{R}}(M_{I,\mathbb{Z}}) \\
&\quad + \mathbf{v}_{K\leftarrow J,\mathbb{R}}\left(\mathbf{H}_{J,\mathbb{Z}\leftarrow\mathbb{R}} \circ \mathbf{H}_{I,\mathbb{R}\leftarrow\mathbb{Z}} M_{I,\mathbb{Z}} + \mathbf{H}_{J,\mathbb{Z}\leftarrow\mathbb{R}} \circ \mathbf{v}_{J\leftarrow I,\mathbb{R}}(M_{I,\mathbb{Z}})\right)
\end{aligned}$$

5.5.7 Vector field ∘ vector field in $\mathbb{Z}$

$$\begin{aligned} M_{J,\mathbb{Z}} &= M_{I,\mathbb{Z}} + \mathbf{v}_{J\leftarrow I,\mathbb{Z}}(M_{I,\mathbb{Z}}) \\ M_{K,\mathbb{Z}} &= M_{J,\mathbb{Z}} + \mathbf{v}_{K\leftarrow J,\mathbb{Z}}(M_{J,\mathbb{Z}}) \\ &= M_{I,\mathbb{Z}} + \mathbf{v}_{J\leftarrow I,\mathbb{Z}}(M_{I,\mathbb{Z}}) + \mathbf{v}_{K\leftarrow J,\mathbb{Z}}\left(M_{I,\mathbb{Z}} + \mathbf{v}_{J\leftarrow I,\mathbb{Z}}(M_{I,\mathbb{Z}})\right) \end{aligned}$$

$$\begin{aligned} \mathbf{v}_{K\leftarrow I,\mathbb{Z}}(M_{I,\mathbb{Z}}) &= M_{K,\mathbb{Z}} - M_{I,\mathbb{Z}} \\ &= \mathbf{v}_{J\leftarrow I,\mathbb{Z}}(M_{I,\mathbb{Z}}) + \mathbf{v}_{K\leftarrow J,\mathbb{Z}}\left(M_{I,\mathbb{Z}} + \mathbf{v}_{J\leftarrow I,\mathbb{Z}}(M_{I,\mathbb{Z}})\right) \end{aligned}$$

5.6 Field of view center alignment

For computational reasons, it may be useful to change the resolution of an image, typically dividing the dimensions by a factor 2 (and then multiplying the voxel sizes by 2). It allows to deal with smaller images while letting the FOV sizes unchanged.

However, because of our conventions where the frame origin is not at the upper left corner of the FOV but at the center of the upper left voxel, the transformation that goes from one frame to the other is a translation (the translation between the two origins).

Let us consider the general case where we want to build the translation that superimposes the FOV centers of two images I_{ref} and I_{flo}. Without loss of generality, we only consider the X direction. I_{ref} and I_{flo} have respectively a number of voxels of $d_{ref,x}$ and $d_{flo,x}$. The FOV intervals (in voxels) are respectively $[-1/2, d_{ref,x} - 1/2]$ and $[-1/2, d_{flo,x} - 1/2]$ in $\mathbb{Z}$. The FOV centers along the X direction are then respectively $\frac{(d_{ref,x}-1)}{2}$ and $\frac{(d_{fflo,x}-1)}{2}$.

In 3D, the FOV centers of I_{ref} and I_{flo} are respectively

$$C_{ref,\mathbb{Z}} = \begin{pmatrix} \frac{(d_{r,x}-1)}{2} \\ \frac{(d_{r,y}-1)}{2} \\ \frac{(d_{r,z}-1)}{2} \end{pmatrix} \quad \text{and} \quad C_{flo,\mathbb{Z}} = \begin{pmatrix} \frac{(d_{f,x}-1)}{2} \\ \frac{(d_{f,y}-1)}{2} \\ \frac{(d_{f,z}-1)}{2} \end{pmatrix}$$

The translation of $\mathbf{T}_{flo\leftarrow ref}$ that aligns the FOV centers in the real space is then

$$C_{flo,\mathbb{R}} - C_{ref,\mathbb{R}} = \mathbf{H}_{flo,\mathbb{R}\leftarrow\mathbb{Z}} C_{flo,\mathbb{Z}} - \mathbf{H}_{ref,\mathbb{R}\leftarrow\mathbb{Z}} C_{ref,\mathbb{Z}}$$

Moreover, such an alignment of the centers of field of view is also the initial transformation (when no initial transformation is given) for the registration through `blockmatching` .

5.7 Transformation inversion

Knowing the transformation $\mathbf{T}_{flo\leftarrow ref}$, we aim at estimating $\mathbf{T}_{ref\leftarrow flo}$,

5.7.1 Linear transformations

Since linear transformations are represented by 4×4 matrices, inverting them is trivial and comes to invert the matrices. We have

$$\mathbf{T}_{ref \leftarrow flo,\mathbb{Z}} = \mathbf{T}^{-1}_{flo \leftarrow ref,\mathbb{Z}} \quad \text{and} \quad \mathbf{T}_{ref \leftarrow flo,\mathbb{R}} = \mathbf{T}^{-1}_{flo \leftarrow ref,\mathbb{R}}$$

5.7.2 Vector field in $\mathbb{Z}$

5.7.2.1 Principle

From the direct transformation, we have

$$M_{flo,\mathbb{Z}} = M_{ref,\mathbb{Z}} + \mathbf{v}_{flo \leftarrow ref,\mathbb{Z}}(M_{ref,\mathbb{Z}})$$

and the inverse transformation allows to express $M_{ref,\mathbb{Z}}$ from $M_{flo,\mathbb{Z}}$

$$\begin{aligned} M_{ref,\mathbb{Z}} &= M_{flo,\mathbb{Z}} + \mathbf{v}_{ref \leftarrow flo,\mathbb{Z}}(M_{flo,\mathbb{Z}}) \\ &= M_{ref,\mathbb{Z}} + \mathbf{v}_{flo \leftarrow ref,\mathbb{Z}}(M_{ref,\mathbb{Z}}) + \mathbf{v}_{ref \leftarrow flo,\mathbb{Z}}(M_{flo,\mathbb{Z}}) \end{aligned}$$

Hence we have

$$\begin{aligned} 0 &= \mathbf{v}_{ref \leftarrow flo,\mathbb{Z}}(M_{flo,\mathbb{Z}}) + \mathbf{v}_{flo \leftarrow ref,\mathbb{Z}}(M_{ref,\mathbb{Z}}) \\ &= \mathbf{v}_{ref \leftarrow flo,\mathbb{Z}}(M_{flo,\mathbb{Z}}) + \mathbf{v}_{flo \leftarrow ref,\mathbb{Z}}(M_{flo,\mathbb{Z}} + \mathbf{v}_{ref \leftarrow flo,\mathbb{Z}}(M_{flo,\mathbb{Z}})) \end{aligned}$$

For the sake of simplicity, let us denote

$$\begin{aligned} M &= M_{flo,\mathbb{Z}} \\ \mathbf{v} &= \mathbf{v}_{flo \leftarrow ref,\mathbb{Z}} \\ \mathbf{v}^{-1} &= \mathbf{v}_{ref \leftarrow flo,\mathbb{Z}} \\ M' &= M_{ref,\mathbb{Z}} = M + \mathbf{v}^{-1}(M) = M_{flo,\mathbb{Z}} + \mathbf{v}_{ref \leftarrow flo,\mathbb{Z}}(M_{flo,\mathbb{Z}}) \end{aligned}$$

Computing the inverse transformation, i.e. the vector field $\mathbf{v}_{ref \leftarrow flo,\mathbb{Z}} = \mathbf{v}^{-1}$, aims at minimizing the above expression. Computing the inverse vector field can be achieved by minimizing

$$\mathbf{v}^{-1}(M) + \mathbf{v}(M') = \mathbf{v}^{-1}(M) + \mathbf{v}(M + \mathbf{v}^{-1}(M)) \tag{5.2}$$

The Newton method aims at estimating iteratively a small variation δ of $\mathbf{v}^{-1}(M)$ that make null equation 5.2.

$$\begin{aligned} E &= \mathbf{v}^{-1}(M) + \delta + \mathbf{v}(M' + \delta) \\ &\quad \text{recalling that } \mathbf{v}(M' + \delta) \approx \mathbf{v}(M') + (\mathbf{v}.\nabla^t)(M')\delta \\ &= \mathbf{v}^{-1}(M) + \delta + \mathbf{v}(M') + (\mathbf{v}.\nabla^t)(M')\delta \end{aligned}$$

Please note that the derivative of $\mathbf{v} = \mathbf{v}_{flo \leftarrow ref,\mathbb{Z}}$ are computed with respect to $M = M_{ref,\mathbb{Z}}$, i.e. in the voxel frame.

$$\begin{aligned} & E = 0 \\ \Leftrightarrow \quad & \left(\mathbf{Id} + (\mathbf{v}.\nabla^t)(M')\right)\delta = -\left(\mathbf{v}^{-1}(M) + \mathbf{v}(M')\right) \\ \Leftrightarrow \quad & \delta = -\left(\mathbf{Id} + (\mathbf{v}.\nabla^t)(M')\right)^{-1}\left(\mathbf{v}^{-1}(M) + \mathbf{v}(M')\right) \end{aligned}$$

5.7.2.2 Implementation

$\left(\mathbf{Id} + (\mathbf{v}.\nabla^t)\right)^{-1}$ are precomputed (I_{ref} frame).

For every point $M = M_{flo,\mathbb{Z}}$, we iterate:

1. computation of $M' = M_{ref,\mathbb{Z}} = M + \mathbf{v}^{-1}(M)$
2. computation of $\left(\mathbf{Id} + (\mathbf{v}.\nabla^t)\right)^{-1}(M')$ and $\mathbf{v}(M')$
3. test on $\|\mathbf{v}^{-1}(M) + \mathbf{v}(M')\|$ (ending condition)
4. computation of $\delta = \left(\mathbf{Id} + (\mathbf{v}.\nabla^t)\right)^{-1}(M')\left(\mathbf{v}^{-1}(M) + \mathbf{v}(M')\right)$
5. update of $\mathbf{v}^{-1}(M)$: $\mathbf{v}^{-1}(M) \leftarrow \mathbf{v}^{-1}(M) - \alpha\delta$

5.7.2.3 Initialization

From $M' = M_{ref,\mathbb{Z}}$, we compute

$$M_{flo,\mathbb{Z}} = M_{ref,\mathbb{Z}} + \mathbf{v}_{flo\leftarrow ref,\mathbb{Z}}(M_{ref,\mathbb{Z}})$$

Please note that $M_{flo,\mathbb{Z}}$ has real coordinates, though it is a point defined in the discrete frame. $-\mathbf{v}(M')$ is then distributed to the discrete points around $M = M_{flo,\mathbb{Z}}$.

5.7.3 Vector field in $\mathbb{R}$

5.7.3.1 Principle

We have

$$M_{flo,\mathbb{R}} = M_{ref,\mathbb{R}} + \mathbf{v}_{flo\leftarrow ref,\mathbb{R}}(M_{ref,\mathbb{Z}})$$

and

$$\begin{aligned} M_{ref,\mathbb{R}} &= M_{flo,\mathbb{R}} + \mathbf{v}_{ref\leftarrow flo,\mathbb{R}}(M_{flo,\mathbb{Z}}) \\ &= M_{ref,\mathbb{R}} + \mathbf{v}_{flo\leftarrow ref,\mathbb{R}}(M_{ref,\mathbb{Z}}) + \mathbf{v}_{ref\leftarrow flo,\mathbb{R}}(M_{flo,\mathbb{Z}}) \end{aligned}$$

Hence we have

$$\begin{aligned} 0 &= \mathbf{v}_{ref\leftarrow flo,\mathbb{R}}(M_{flo,\mathbb{Z}}) + \mathbf{v}_{flo\leftarrow ref,\mathbb{R}}(M_{ref,\mathbb{Z}}) \\ &= \mathbf{v}_{ref\leftarrow flo,\mathbb{R}}(M_{flo,\mathbb{Z}}) + \mathbf{v}_{flo\leftarrow ref,\mathbb{R}}(\mathbf{H}_{ref,\mathbb{Z}\leftarrow\mathbb{R}} M_{ref,\mathbb{R}}) \\ &= \mathbf{v}_{ref\leftarrow flo,\mathbb{R}}(M_{flo,\mathbb{Z}}) \\ &\quad + \mathbf{v}_{flo\leftarrow ref,\mathbb{R}}(\mathbf{H}_{ref,\mathbb{Z}\leftarrow\mathbb{R}}(M_{flo,\mathbb{R}} + \mathbf{v}_{ref\leftarrow flo,\mathbb{R}}(M_{flo,\mathbb{Z}}))) \\ &= \mathbf{v}_{ref\leftarrow flo,\mathbb{R}}(M_{flo,\mathbb{Z}}) \\ &\quad + \mathbf{v}_{flo\leftarrow ref,\mathbb{R}}(\mathbf{H}_{ref,\mathbb{Z}\leftarrow\mathbb{R}}\mathbf{H}_{flo,\mathbb{R}\leftarrow\mathbb{Z}} M_{flo,\mathbb{Z}} + \mathbf{H}_{ref,\mathbb{Z}\leftarrow\mathbb{R}}\mathbf{v}_{ref\leftarrow flo,\mathbb{R}}(M_{flo,\mathbb{Z}})) \end{aligned}$$

For the sake of simplicity, let us denote

$$\begin{aligned} M &= M_{flo,\mathbb{Z}} \\ \mathbf{v} &= \mathbf{v}_{flo\leftarrow ref,\mathbb{R}} \\ \mathbf{v}^{-1} &= \mathbf{v}_{ref\leftarrow flo,\mathbb{R}} \end{aligned}$$

$$\begin{aligned}
\mathbf{H}_r &= \mathbf{H}_{ref,\mathbb{Z}\leftarrow\mathbb{R}} \\
\mathbf{H}_f &= \mathbf{H}_{flo,\mathbb{R}\leftarrow\mathbb{Z}} \\
M' &= M_{ref,\mathbb{Z}} = \mathbf{H}_r\mathbf{H}_f M + \mathbf{H}_r\mathbf{v}^{-1}(M) \\
&= \mathbf{H}_{ref,\mathbb{Z}\leftarrow\mathbb{R}}\mathbf{H}_{flo,\mathbb{R}\leftarrow\mathbb{Z}}M_{flo,\mathbb{Z}} + \mathbf{H}_{ref,\mathbb{Z}\leftarrow\mathbb{R}}\mathbf{v}_{ref\leftarrow flo,\mathbb{R}}(M_{flo,\mathbb{Z}})
\end{aligned}$$

Computing the inverse transformation, i.e. the vector field $\mathbf{v}_{ref\leftarrow flo,\mathbb{R}} = \mathbf{v}^{-1}$, aims at minimizing the above expression. Computing the inverse vector field can be achieved by minimizing

$$\mathbf{v}^{-1}(M) + \mathbf{v}(M') = \mathbf{v}^{-1}(M) + \mathbf{v}(\mathbf{H}_r\mathbf{H}_f M + \mathbf{H}_r\mathbf{v}^{-1}(M)) \tag{5.3}$$

$$\begin{aligned}
E &= \mathbf{v}^{-1}(M) + \delta + \mathbf{v}(\mathbf{H}_r\mathbf{H}_f M + \mathbf{H}_r(\mathbf{v}^{-1}(M) + \delta)) \\
&= \mathbf{v}^{-1}(M) + \delta + \mathbf{v}(M' + \mathbf{H}_r\delta) \\
& \text{ recalling that } \mathbf{v}(M'+\delta) \approx \mathbf{v}(M') + (\mathbf{v}.\nabla^t)(M')\delta \\
&= \mathbf{v}^{-1}(M) + \delta + \mathbf{v}(M') + (\mathbf{v}.\nabla^t)(M')\mathbf{H}_r\delta
\end{aligned}$$

Please note that the derivative of $\mathbf{v} = \mathbf{v}_{flo\leftarrow ref,\mathbb{R}}$ are computed with respect to $M = M_{ref,\mathbb{Z}}$, i.e. in the voxel frame.

$$\begin{aligned}
&E = 0 \\
&\quad\Leftrightarrow\quad \delta = -\left(\mathbf{Id} + (\mathbf{v}.\nabla^t)(M')\mathbf{H}_r\right)^{-1}\left(\mathbf{v}^{-1}(M) + \mathbf{v}(M')\right)
\end{aligned}$$

5.7.3.2 Implementation

$(\mathbf{Id} + (\mathbf{v}.\nabla^t)\mathbf{H}_r)^{-1}$ are precomputed (I_{ref} frame).

For every point $M = M_{flo,\mathbb{Z}}$, we iterate:

1. computation of $M' = M_{ref,\mathbb{Z}} = \mathbf{H}_r\mathbf{H}_f M + \mathbf{H}_r\mathbf{v}^{-1}(M)$
2. computation of $(\mathbf{Id} + (\mathbf{v}.\nabla^t)\mathbf{H}_r)^{-1}(M')$ and $\mathbf{v}(M')$
3. test on $\|\mathbf{v}^{-1}(M) + \mathbf{v}(M')\|$ (ending condition)
4. computation of $\delta = (\mathbf{Id} + (\mathbf{v}.\nabla^t)\mathbf{H}_r)^{-1}(M')\left(\mathbf{v}^{-1}(M) + \mathbf{v}(M')\right)$
5. update of $\mathbf{v}^{-1}(M)$: $\mathbf{v}^{-1}(M) \leftarrow \mathbf{v}^{-1}(M) - \alpha\delta$

5.7.3.3 Initialization

From $M' = M_{ref,\mathbb{Z}}$, we compute

$$\begin{aligned}
M_{flo,\mathbb{Z}} &= \mathbf{H}_{flo,\mathbb{Z}\leftarrow\mathbb{R}}M_{flo,\mathbb{R}} \\
&= \mathbf{H}_{flo,\mathbb{Z}\leftarrow\mathbb{R}}\left(M_{ref,\mathbb{R}} + \mathbf{v}_{flo\leftarrow ref,\mathbb{R}}(M_{ref,\mathbb{Z}})\right) \\
&= \mathbf{H}_{flo,\mathbb{Z}\leftarrow\mathbb{R}}\left(\mathbf{H}_{ref,\mathbb{R}\leftarrow\mathbb{Z}}M_{ref,\mathbb{Z}} + \mathbf{v}_{flo\leftarrow ref,\mathbb{R}}(M_{ref,\mathbb{Z}})\right)
\end{aligned}$$

Please note that $M_{flo,\mathbb{Z}}$ has real coordinates, though it is a point defined in the discrete frame. $-\mathbf{v}(M')$ is then distributed to the discrete points around $M = M_{flo,\mathbb{Z}}$.

Chapter 6

Image interpolation

6.1 Linear transformations

Let us consider two images I_0 and I_1, and the transformation $A_{0 \leftarrow 1}$ that allows to resample I_0 into I_1 frame, ie to compute $I_0 \circ A_{0 \leftarrow 1}$.

To interpolate an image I_t at an intermediary position $t \in [0 \ldots 1]$ from both I_0 and I_1, we have to resample both I_0 and I_1 in I_t frame and then combine them.

We have to estimate both $A_{0 \leftarrow t}$ and $A_{1 \leftarrow t}$, ie the transformations from I_t towards I_0 and I_1, that enable to resample I_0 and I_1 in the frame of I_t by $I_0 \circ A_{0 \leftarrow t}$ and $I_1 \circ A_{1 \leftarrow t}$.

Indeed, let us consider I_0. To resample this image in I_t frame, we pick every point (voxel) M_t of I_t frame, transform it into I_0 and compute (interpolate) its value there: it then requires the transformation from I_t towards I_0 that is denoted by $A_{0 \leftarrow t}$.

If we assume the "linearity" of the motion from M_0 to M_1, an intermediary point M_t can be defined

1. either from M_0 and M_1 with

$$\begin{aligned} M_t &= (1-t)M_0 + tM_1 = (1-t)A_{0 \leftarrow 1}M_1 + tM_1 \\ &= \left(t\mathbf{Id} + (1-t)A_{0 \leftarrow 1}\right) M_1 \end{aligned}$$

2. or from M_1 with

$$\begin{aligned} M_t &= M_1 + (1-t)\overrightarrow{M_1M_0} = M_1 + (1-t)(M_0 - M_1) \\ &= M_1 + (1-t)(A_{0 \leftarrow 1}M_1 - M_1) = \left(t\mathbf{Id} + (1-t)A_{0 \leftarrow 1}\right) M_1 \end{aligned}$$

thus

$$A_{t \leftarrow 1} = t\mathbf{Id} + (1-t)A_{0 \leftarrow 1}$$

and $A_{1 \leftarrow t}$ is computed by inverting the previous matrix

$$A_{1 \leftarrow t} = A_{t \leftarrow 1}^{-1}$$

The computation of $A_{0\leftarrow t}$ can be done

1. either with a similar calculation to the one of $A_{1\leftarrow t}$, i.e. by first computing $A_{t\leftarrow 0}$ with
$$A_{t\leftarrow 0} = (1-t)\mathbf{Id} + tA_{1\leftarrow 0}$$
with $A_{1\leftarrow 0} = A^{-1}_{0\leftarrow 1}$ and second inverting it, ie $A_{0\leftarrow t} = A^{-1}_{t\leftarrow 0}$,

2. or by observing that M_t is partway on the "line" joining M_0 and M_1 and at a "distance" t from M_1 and $(1-t)$ from M_0. We have then
$$\begin{aligned} M_t = (1-t)M_0 + tM_1 \quad &\Leftrightarrow \quad M_0 = \frac{1}{1-t}M_t - \frac{t}{1-t}M_1 \\ &\Leftrightarrow \quad M_0 = \frac{1}{1-t}M_t - \frac{t}{1-t}A_{1\leftarrow t}M_t \\ &\Leftrightarrow \quad M_0 = \left(\frac{1}{1-t}\mathbf{Id} - \frac{t}{1-t}A_{1\leftarrow t}\right)M_t \end{aligned}$$
Thus
$$A_{0\leftarrow t} = \frac{1}{1-t}\mathbf{Id} - \frac{t}{1-t}A_{1\leftarrow t}$$

Finally, the image I_t can be interpolated from the resampled I_0 and I_1 into I_t frame, ie $I_0 \circ A_{0\leftarrow t}$ and $I_1 \circ A_{1\leftarrow t}$ with
$$I_t = (1-t) \times I_0 \circ A_{0\leftarrow t} + t \times I_1 \circ A_{1\leftarrow t}$$

[**Pay attention**] When dealing with rigid transformations, the above interpolation scheme does not result in a rigidly displaced object/image, since a linear combination of matrices representing rigid displacements does not represent a rigid displacement (in the general case). If one wants to get an interpolation of the rigid displacement, one has to compute intermediary displacements along the geodesic line (in the rigid transformation manifold) from the identity to the final transformation $R_{0\leftarrow 1}$. Typically, if one rotates a sphere, it may be wanted that each point of the sphere follows a circle arc instead of a straight line.

6.2 Non-linear transformations defined by vector fields

Let us consider two images I_0 and I_1, and the transformation $T_{0\leftarrow 1}$ that allows to resample I_0 into I_1 frame, ie to compute $I_0 \circ T_{0\leftarrow 1}$. Please note that $T_{0\leftarrow 1}$ is a function defined from I_1 towards I_0. When dealing with non-linear transformations, $T_{0\leftarrow 1}$ may be represented by a vector field $\mathbf{u}_{0\leftarrow 1}$.

$$\left\{ \begin{array}{lcl} T_{0\leftarrow 1} & = & Id + \mathbf{u}_{0\leftarrow 1} \\ T_{0\leftarrow 1}(M) & = & M + \mathbf{u}_{0\leftarrow 1}(M) \quad \Longleftrightarrow \quad M_0 = M_1 + \mathbf{u}_{0\leftarrow 1}(M_1) \end{array} \right.$$

To interpolate an image I_t at an intermediary position $t \in [0 \dots 1]$ from both I_0 and I_1, we have to estimate both $T_{0 \leftarrow t}$ and $T_{1 \leftarrow t}$, ie the transformations from I_t towards I_0 and I_1, that enable to resample I_0 and I_1 in the frame of I_t by $I_0 \circ T_{0 \leftarrow t}$ and $I_1 \circ T_{1 \leftarrow t}$.

If we assume the "linearity" of the deformation from M_1 to M_0, an intermediary point M_t can be defined

1. either from M_0 and M_1 with

$$\begin{aligned} M_t &= (1-t)M_0 + tM_1 = (1-t)\left(M_1 + \mathbf{u}_{0 \leftarrow 1}(M_1)\right) + tM_1 \\ &= M_1 + (1-t)\mathbf{u}_{0 \leftarrow 1}(M_1) \end{aligned}$$

2. or from M_1 with

$$\begin{aligned} M_t &= M_1 + (1-t)\overrightarrow{M_1 M_0} = M_1 + (1-t)(M_0 - M_1) \\ &= M_1 + (1-t)\mathbf{u}_{0 \leftarrow 1}(M_1) \end{aligned}$$

thus $T_{t \leftarrow 1}$ is defined by the vector field $\mathbf{u}_{t \leftarrow 1} = (1-t)\mathbf{u}_{0 \leftarrow 1}$, and $T_{1 \leftarrow t}$ is then simply obtained by inverting $T_{t \leftarrow 1}$

$$T_{1 \leftarrow t} = T_{t \leftarrow 1}^{-1}$$

Let $\mathbf{u}_{1 \leftarrow t}$ be the vector field representing $T_{1 \leftarrow t}$, and we have

$$M_1 = M_t + \mathbf{u}_{1 \leftarrow t}(M_t)$$

[Pay attention] $\mathbf{u}_{1 \leftarrow t}$ *is not* $-\mathbf{u}_{t \leftarrow 1} = -(1-t)\mathbf{u}_{0 \leftarrow 1}$. The transformation $T_{t \leftarrow 1}$ allows to compute the point $M_t \in I_t$ that is the transformed point $M_1 \in I_1$ (M_t and M_1 have different coordinates as soon as $T_{t \leftarrow 1}$ is different from the identity transformation) by

$$M_t = M_1 + \mathbf{u}_{t \leftarrow 1}(M_1) \tag{6.1}$$

Using the opposite of $\mathbf{u}_{t \leftarrow 1}$ to define the inverse of $T_{t \leftarrow 1}$ comes to state that

$$M_1 = M_t - \mathbf{u}_{t \leftarrow 1}(M_t)$$

However, from Eq. (6.1) we have $M_1 = M_t - \mathbf{u}_{t \leftarrow 1}(M_1)$. Thus, using the opposite vector field to define the inverse transformation is valid iff

$$\mathbf{u}_{t \leftarrow 1}(M_t) = \mathbf{u}_{t \leftarrow 1}(M_1) \Leftrightarrow \mathbf{u}_{0 \leftarrow 1}(M_t) = \mathbf{u}_{0 \leftarrow 1}(M_1)$$

This may be a good estimate in case of small (hence M_t will be close to M_1) and slowly varying deformations, but is obviously false in the general case (see Fig. 6.1).

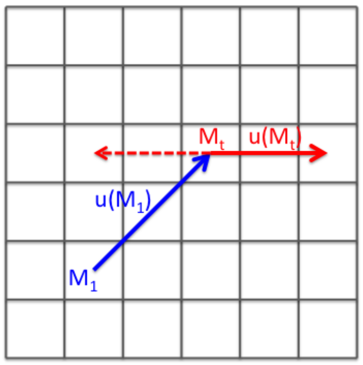


Figure 6.1: M_t is M_1 transformed by $T_{t\leftarrow 1}$, i.e. $M_t = M_1 + \mathbf{u}_{t\leftarrow 1}(M_1)$ where $\mathbf{u}_{t\leftarrow 1}(M_1)$ is the blue vector. $T_{t\leftarrow 1}^{-1}$ at M_t should then by defined by the opposite of this blue vector (so that M_t can project on M_1). By building the inverse of $T_{t\leftarrow 1}$ with the opposite of the $\mathbf{u}_{t\leftarrow 1}$ vector field, $T_{t\leftarrow 1}^{-1}$ at M_t would be defined by $-\mathbf{u}_{t\leftarrow 1}(M_t)$, i.e. the dashed red vector (and thus not equal to $-\mathbf{u}_{t\leftarrow 1}(M_1)$, the opposite of the blue vector). This can be an acceptable approximation iff $\mathbf{u}_{t\leftarrow 1}(M_t)$ (the red vector) is sufficiently similar to $\mathbf{u}_{t\leftarrow 1}(M_1)$ (the blue vector).

M_t is partway on the "line" joining M_0 and M_1 and at a "distance" t from M_1 and $(1-t)$ from M_0. We have then

$$\begin{aligned}
M_t = (1-t)M_0 + tM_1 \quad &\Leftrightarrow \quad M_0 = \frac{1}{1-t}M_t - \frac{t}{1-t}M_1 \\
&\Leftrightarrow \quad M_0 = \frac{1}{1-t}M_t - \frac{t}{1-t}T_{1\leftarrow t}M_t \\
&\Leftrightarrow \quad M_0 = \frac{1}{1-t}M_t - \frac{t}{1-t}\left(M_t + \mathbf{u}_{1\leftarrow t}(M_t)\right) \\
&\Leftrightarrow \quad M_0 = M_t - \frac{t}{1-t}\mathbf{u}_{1\leftarrow t}(M_t)
\end{aligned}$$

The transformation $T_{0\leftarrow t}$ is then represented by the vector field $\left(-\frac{t}{1-t}\mathbf{u}_{1\leftarrow t}\right)$.

Finally, the image I_t can be interpolated from the resampled I_0 and I_1 into I_t frame, ie $I_0 \circ T_{0\leftarrow t}$ and $I_1 \circ T_{1\leftarrow t}$ with

$$I_t = (1-t) \times I_0 \circ T_{0\leftarrow t} + t \times I_1 \circ T_{1\leftarrow t}$$

Index