Contents

Chapter 1

User documentation

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

$T_{res} = T_{I_{flo} \leftarrow I_{ref}}$ 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 $I_{flo}$ $I_{res}$ -trsf $T_{res}$ -template $I_{ref}$
```

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 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.

1.1.3 Transformation estimation

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.

1.1.4 Vector field transformations

1.1.5 Initial transformations

There are two (not exclusive) ways to specify an initial transformation. These two ways of instanciating an initial transformation are related to the way the result transformation is calculated:

$$T_{res} = T_{init} \circ \overbrace{\delta T \circ \ldots \circ \delta T}^{\substack{\text{incremental transformations} \\ \text{computed during} \\ r\text{egistration}}} \circ T^0 \tag{1.1}$$

T_{init} and T_0 are respectively initialized by `'-initial-transformation'` and `'-initial-result-transformation'`, but T_{init} will remain unchanged during the computation while T_0 will be iteratively updated.

- the option `'-initial-[voxel-]transformation'` is used to specify a transformation T_{init} that is applied to the floating image I_{flo}. Therefore, it comes to register $I_{flo} \circ T_{init}$ with I_{ref}. In other words, the transformation T_{res} obtained with

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

  is comparable[1] to the one, $T_{res,2}$ obtained with the following commands

  % `applyTrsf` I_{flo} $I_{flo,2}$ `-trsf` T_{init}
  % `blockmatching -flo` $I_{flo,2}$ `-ref` I_{ref} `...  -res-trsf` $T_{intermediary}$
  % `composeTrsf -res` $T_{res,2}$ `-trsfs` T_{init} $T_{intermediary}$

[1]should be equal, but due to resampling effect, differences can occur.

- 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`

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

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

  [Note] This can somewhow be viewed as registering I_{flo} with $I_{ref} \circ T_0^{-1}$, but with blocks that would not be square but deformed by T_0^{-1}.

1.1.6 Result transformation

`'-result-[voxel-]transformation'` is used to specify the name of the output transfomration $T_{res} = \circ T_{I_{flo} \leftarrow I_{ref}}$ that allows to resample I_{flo} onto I_{ref}. When initial transformations are given, they are "included" in the result transformation (see Eq. 1.1).

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

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

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

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

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

1.3 applyTrsf

`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,

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

1.4 `composeTrsf`

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

```
% composeTrsf ...  -res Tres -trsfs T1 T2 ...  TN
```

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 Tres -trsfs T1 T2 ...  TN
```

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 I0 I1 -trsf T0 ...
% applyTrsf I1 I2 -trsf T1 ...
% applyTrsf I2 I3 -trsf T2 ...
```

is equivalent to the transformation composition

```
% composeTrsf -res T_{I0<-I3} -trsfs T0 T1 T2
```

that allows to get I_3 directly from I_0

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

1.5 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]
```

1.6 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}.

1.7 `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 Iref Jref -origin 25 35 45 -dim 100 90 80 -res-trsf Cref
```

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 Iref Jref -trsf Cref -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 Iref Jref ...  -res-trsf Cref
   % cropImage Iflo Jflo ...  -res-trsf Cflo
   ```
   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 Jflo -ref Jref -res Jres -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
   $$\begin{aligned} J_{res} &= J_{flo} \circ T'_{res} \\ &= I_{flo} \circ C_{flo} \circ T'_{res} \end{aligned}$$
   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 Cflo T'res
   % applyTrsf Iflo Jres -trsf T''res -template Jref
   ```

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}

1.8 `invTrsf`

1.9 `printTrsf`

Chapter 2

Developper documentation / Notes

2.1 Frame conversion: *real* $\leftrightarrow$ *voxel*

2.1.1 Image

Every image I is associated with a conversion matrix. Basically, it is simply a diagonal matrix 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

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

Accordingly,

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

Obviously, we have $H_{I,\mathbb{Z}\leftarrow\mathbb{R}} = H^{-1}_{I,\mathbb{R}\leftarrow\mathbb{Z}}$.

2.1.2 Transformations

We simply use the composition rules to express the *voxel* transformation from image I_{ref} to image I_{flo}, $T_{flo\leftarrow ref,\mathbb{Z}} = T_{flo_{\mathbb{Z}}\leftarrow ref_{\mathbb{Z}}}$ from the same transformation in the real frame, $T_{flo\leftarrow ref,\mathbb{R}} = T_{flo_{\mathbb{R}}\leftarrow ref_{\mathbb{R}}}$.

$$T_{flo\leftarrow ref,\mathbb{Z}} = H_{flo,\mathbb{Z}\leftarrow\mathbb{R}} \circ T_{flo\leftarrow ref,\mathbb{R}} \circ H_{ref,\mathbb{R}\leftarrow\mathbb{Z}}$$

Conversely,

$$T_{flo\leftarrow ref,\mathbb{R}} = H_{flo,\mathbb{R}\leftarrow\mathbb{Z}} \circ T_{flo\leftarrow ref,\mathbb{Z}} \circ H_{ref,\mathbb{Z}\leftarrow\mathbb{R}}$$

2.1.3 Linear transformations

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

2.1.4 Vector fields

The transformation is encoded by a vector field $\mathbf{V}$ that 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 $\mathbf{V}_{\mathbb{R}}$ at $M_{\mathbb{R}}$ gives the displacement of the point $M_{\mathbb{R}}$.

However, since $\mathbf{V}$ 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 $\mathbf{V}_{\mathbb{R}}(H_{\mathbb{Z}\leftarrow\mathbb{R}} M_{\mathbb{R}})$.

The transformation T from image I_{ref} to image I_{flo} is then defined by

$$\begin{array}{rl} & M_{flo,\mathbb{R}} = T(M_{ref,\mathbb{R}}) = M_{ref,\mathbb{R}} + \mathbf{V}_{\mathbb{R}}(H_{ref,\mathbb{Z}\leftarrow\mathbb{R}} M_{ref,\mathbb{R}}) \\ \Leftrightarrow & \mathbf{V}_{\mathbb{R}}(H_{ref,\mathbb{Z}\leftarrow\mathbb{R}} M_{ref,\mathbb{R}}) = M_{flo,\mathbb{R}} - M_{ref,\mathbb{R}} = T(M_{ref,\mathbb{R}}) - M_{ref,\mathbb{R}} \end{array}$$

When expressing this formula in the discrete lattice, it comes

$$\begin{array}{rcl} H_{flo,\mathbb{R}\leftarrow\mathbb{Z}} M_{flo,\mathbb{Z}} & = & H_{ref,\mathbb{R}\leftarrow\mathbb{Z}} M_{ref,\mathbb{Z}} + \mathbf{V}_{\mathbb{R}}(M_{ref,\mathbb{Z}}) \\ M_{flo,\mathbb{Z}} & = & H^{-1}_{flo,\mathbb{R}\leftarrow\mathbb{Z}} \circ H_{ref,\mathbb{R}\leftarrow\mathbb{Z}} M_{ref,\mathbb{Z}} + H^{-1}_{flo,\mathbb{R}\leftarrow\mathbb{Z}} \circ \mathbf{V}_{\mathbb{R}}(M_{ref,\mathbb{Z}}) \end{array}$$

The displacement in *voxel* coordinates $\mathbf{V}_{\mathbb{Z}}$ is then defined by

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

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

2.2 `blockmatching` versus `baladin`

2.2.1 Blocks management

There are two issues.

- The number of blocks calculated in the earlier versions of `baladin` was erroneous (and may yield an overflow).
- As a consequence, some blocks (that should have been considered) were discarded.
- the ordering of the blocks in the earlier versions of `baladin` was z varies first, then y and finally x, so that the index of a block is given by $z + (y + x * dim_y) * dim_z$.

This behavior can be mimicked with the define `_ORIGINAL_BALADIN_BLOCKS_MANAGEMENT_` in the latest version of `baladin` (except for the overflow).

It seems that the indexing order is important (experiments have been conducted by changing the indexing, with the same blocks being discarded), since results may numerically differ. It can be suspected (but has not be proven) that it may change the "best" pairing, because of the scan order, when several blocks result in the same criteria value.

Chapter 3

Misc

3.1 Blocs

3.1.1 Définition des blocs

La dimension d'une image est D (les coordonnées des points vont de 0 à $D-1$), on ne veut pas de blocs dans les offset_p premiers points, ni dans les offset_d derniers points. La dimension d'un bloc est B et les blocs sont espacés de S. On a donc

- coordonnée du premier point inclus dans le bloc $i, i \geq 0$:

  $$x_0 = \mathit{offset}_p + i * S$$

  Donc le point x est le premier point (origine) d'un bloc si

  $$(x - \mathit{offset}_p)\%S = 0$$

  (le reste de la division euclidienne est 0), et l'indice du bloc est alors

  $$(x - \mathit{offset}_p)/S$$

- coordonnée du dernier point inclus dans le bloc i :

  $$x_1 = \mathit{offset}_p + i * S + (B - 1)$$

Pour qu'un bloc soit valide, il faut donc

$$\mathit{offset}_p + i * S + (B-1) \leq D - 1 - \mathit{offset}_d \iff i * S \leq D - 1 - \mathit{offset}_d - B + 1 - \mathit{offset}_p$$
$$i * S \leq D - B - \mathit{offset}_d - \mathit{offset}_p$$

Le dernier indice valide est donc

$$i_d = \left(D - B - \mathit{offset}_d - \mathit{offset}_p\right)/S$$

Comme les indices commencent à 0, il y a donc $i_d + 1$ blocs avec

$$i_d + 1 = \left(D - B - \mathit{offset}_d - \mathit{offset}_p\right)/S + 1$$

3.2 Recalage

3.2.1 Transformation initiale

Si une transformation initiale T_{init} est donnée, elle est utilisée pour une première transformation de l'image flottante. Typiquement, c'est la transformation résultat d'un premier recalage. Cela revient donc à recaler $I_{flo} \circ T_{init}$ avec I_{ref}.

3.2.2 Boucle de recalage

Dans chaque boucle de recalage, à l'itération i, on cherche à recaler $I^i_{flo} = I^0_{flo} \circ T^i$ avec T_{ref}, et on calcule $\delta T^i = T_{I^i_{flo} \leftarrow I_{ref}}$. On a donc

$$T^{i+1} = \delta T^i \circ T^i$$

S'il y a une transformation initiale, on a $I^0_{flo} = I_{flo} \circ T_{init}$, donc la transformation pour rééchantillonner I^i_{flo} à partir de I_{flo} est $T_{init} \circ T^i$.

3.3 Transformations

3.4 Passage d'une transformation "réelle" en "voxels"

3.4.1 Passage d'un référentiel voxel à un référentiel réel

$H_{ref_{\mathbb{R}} \leftarrow ref_{\mathbb{Z}}}$: matrice diagonale des tailles de voxel, permet de passer d'un référentiel voxel à un référentiel réel

$$H_{ref_{\mathbb{R}} \leftarrow ref_{\mathbb{Z}}} = \begin{pmatrix} v_x & . & . \\ . & v_y & . \\ . & . & v_z \end{pmatrix} \quad \text{et} \quad H_{ref_{\mathbb{Z}} \leftarrow ref_{\mathbb{R}}} = \begin{pmatrix} 1/v_x & . & . \\ . & 1/v_y & . \\ . & . & 1/v_z \end{pmatrix}$$

où v_x, v_y et v_z sont les dimensions du voxels selon les 3 directions. On a donc

$$(x, y, z)^t = H_{ref_{\mathbb{R}} \leftarrow ref_{\mathbb{Z}}}(i, j, k)^t$$

3.4.2 Transformation linéaire

On passe de la transformation "réelle" $T_{flo_{\mathbb{R}} \leftarrow ref_{\mathbb{R}}}$ à une transformation "voxels" $T_{flo_{\mathbb{Z}} \leftarrow ref_{\mathbb{Z}}}$ par

$$T_{flo_{\mathbb{Z}} \leftarrow ref_{\mathbb{Z}}} = H_{flo_{\mathbb{Z}} \leftarrow flo_{\mathbb{R}}} \circ T_{flo_{\mathbb{R}} \leftarrow ref_{\mathbb{R}}} \circ H_{ref_{\mathbb{R}} \leftarrow ref_{\mathbb{Z}}}$$

C'est la transformation qui permet, en pratique, de rééchantillonner l'image flo dans la géométrie de l'image ref.

3.4.3 Transformation non-linéaire

Elle est codée par une image de vecteurs $\mathbf{V}_{flo_{\mathbb{R}} \leftarrow ref_{\mathbb{Z}}}$, donc définie sur une image, a priori, I_{ref} pour pouvoir rééchantillonner l'image flottante dans la géométrie de l'image référence.

Ces vecteurs sont un déplacement dans le monde réel, qui indique de combien un voxel se déplace. La transformation dans le référentiel réel se définit donc par

$$T_{flo_{\mathbb{R}} \leftarrow ref_{\mathbb{R}}}(x,y,z) = (x,y,z) + \mathbf{V}_{flo_{\mathbb{R}} \leftarrow ref_{\mathbb{Z}}} \left(H^{-1}_{ref_{\mathbb{R}} \leftarrow ref_{\mathbb{Z}}}(x,y,z) \right)$$

La transformation dans le référentiel voxel se définit donc par

$$T_{flo_{\mathbb{Z}} \leftarrow ref_{\mathbb{Z}}}(i,j,k) = H^{-1}_{flo_{\mathbb{Z}} \leftarrow flo_{\mathbb{R}}} \circ H_{ref_{\mathbb{R}} \leftarrow ref_{\mathbb{Z}}}(i,j,k) + H^{-1}_{flo_{\mathbb{Z}} \leftarrow flo_{\mathbb{R}}} \mathbf{V}_{flo_{\mathbb{R}} \leftarrow ref_{\mathbb{Z}}}(i,j,k)$$

3.5 Calcul d'une transformation aux moindres carres

3.5.1 Rappels

$$\begin{aligned} \sum_i (x_i - \bar{x})^2 &= \sum_i \left(x_i^2 - 2x_i\bar{x} + \bar{x}^2 \right) \\ &= \sum_i x_i^2 - 2\bar{x} \sum_i x_i + N\bar{x}^2 \\ &= \sum_i x_i^2 - 2\bar{x}N\bar{x} + N\bar{x}^2 \\ &= \sum_i x_i^2 - N\bar{x}^2 \end{aligned}$$

$$\begin{aligned} \sum_i (x_i - \bar{x})(x'_i - \bar{x}') &= \sum_i x_i x'_i - \bar{x} \sum_i x'_i - \bar{x}' \sum_i x_i + N\bar{x}\bar{x}' \\ &= \sum_i x_i x'_i - \bar{x}N\bar{x}' - \bar{x}'N\bar{x} + N\bar{x}\bar{x}' \\ &= \sum_i x_i x'_i - N\bar{x}\bar{x}' \end{aligned}$$

3.5.2 Translation + scaling

Le critère est de la forme

$$\sum_i (a_x x_i + t_x - x'_i)^2 + \dots$$

La dérivée par rapport à t_x donne

$$\begin{aligned}\sum_i (a_x x_i + t_x - x_i') = 0 \quad &\iff \quad a_x \sum_i x_i + N t_x - \sum_i x_i' = 0 \\ &\iff \quad t_x = \bar{x}' - a_x \bar{x}\end{aligned}$$

La dérivée par rapport à a_x donne

$$\begin{aligned}\sum_i x_i \left(a_x x_i + t_x - x_i'\right) = 0 \quad &\iff \quad a_x \sum_i x_i^2 + t_x \sum_i x_i - \sum_i x_i x_i' = 0 \\ &\iff \quad a_x \sum_i x_i^2 + N\bar{x}\left(\bar{x}' - a_x \bar{x}\right) - \sum_i x_i x_i' = 0 \\ &\iff \quad a_x \left(\sum_i x_i^2 - N\bar{x}^2\right) + N\bar{x}\bar{x}' - \sum_i x_i x_i' = 0 \\ &\iff \quad a_x \sum_i \left(x_i - \bar{x}\right)^2 - \sum_i \left(x_i - \bar{x}\right)\left(x_i' - \bar{x}'\right) \\ &\iff \quad a_x = \frac{\sum_i \left(x_i - \bar{x}\right)\left(x_i' - \bar{x}'\right)}{\sum_i \left(x_i - \bar{x}\right)^2}\end{aligned}$$

Chapter 4

Misc

4.1 Notes

4.1.1 Block_Matching()

Arguments

- image I^p_{ref} dans la géométrie sous-échantillonnée (niveau p de la pyramide)
- image I_flo dans la géométrie initiale
- transformation permettant de passer de I^p_{ref} à I_{flo}.
  - cas linéaire : c'est
    $$T_{flo_{\mathbb{R}} \leftarrow ref^p{}_{\mathbb{R}}}$$
    et le rééchantillonnage se fait avec
    $$H_{flo_{\mathbb{Z}} \leftarrow flo_{\mathbb{R}}} \circ T_{flo_{\mathbb{R}} \leftarrow ref^p{}_{\mathbb{R}}} \circ H_{ref^p{}_{\mathbb{R}} \leftarrow ref^p{}_{\mathbb{Z}}}$$
  - cas non-linéaire : c'est une image vectorielle $\mathbf{V}^p$ définie sur la même grille que I^p_{ref}
    $$T_{flo_{\mathbb{Z}} \leftarrow ref_{\mathbb{Z}}}(i,j,k) = H^{-1}_{flo^p{}_{\mathbb{Z}} \leftarrow flo_{\mathbb{R}}} \circ H_{ref_{\mathbb{R}} \leftarrow ref^p{}_{\mathbb{Z}}}(i,j,k) + H^{-1}_{flo_{\mathbb{Z}} \leftarrow flo_{\mathbb{R}}} \mathbf{V}^p_{flo_{\mathbb{R}} \leftarrow ref^p{}_{\mathbb{Z}}}(i,j,k)$$
- Allocation d'une image I^p_{flo} de même géometrie que I^p_{ref}
- Allocation de la liste des blocs
  - Pour les non-linéaires, les tailles de blocs doivent être impairs
- Calcul des paramètres de blocs pour I^p_{ref}
- Boucle
  1. Rééchantillonnage de I_{flo} en I^i_{flo} (i : itération) avec

2. Calcul des paramètres de blocs pour I^i_{flo}
3. Tri des blocs pour I^i_{flo}
4. Calcul des appariements entre les blocs pour I^i_{flo} et ceux de I^p_{ref}
5. Calcul de la transformation incrmentale $\delta T_{flo^i \leftarrow ref^p}$
   - cas linéaire : c'est une transformation dans le référentiel réel
   $$\delta T_{{flo^i}_{\mathbb{R}} \leftarrow {ref^p}_{\mathbb{R}}}$$
   - cas non-linéaire : c'est un champ de vecteurs $\delta \mathbf{V}_{{flo^i}_{\mathbb{R}} \leftarrow {ref^p}_{\mathbb{Z}}}$
6. Mise à jour de la transformation $T_{flo^i \leftarrow ref^p}$
   - cas linéaire : C'est une transformation dans le référentiel réel
   $$T_{{flo^{i+1}}_{\mathbb{R}} \leftarrow {ref^p}_{\mathbb{R}}} = \delta T_{{flo^i}_{\mathbb{R}} \leftarrow {ref^p}_{\mathbb{R}}} \circ T_{flo^i \leftarrow ref^p}$$
   - cas non-linéaire :
   $$\begin{aligned} \mathbf{V}_{{flo^{i+1}}_{\mathbb{R}} \leftarrow {ref^p}_{\mathbb{Z}}}(i,j,k) &= \mathbf{V}_{{flo^i}_{\mathbb{R}} \leftarrow {ref^p}_{\mathbb{Z}}}(i,j,k) \\ &\quad + \delta \mathbf{V}_{{flo^i}_{\mathbb{R}} \leftarrow {ref^p}_{\mathbb{Z}}} \Big((i,j,k) + H^{-1}_{flo_{\mathbb{Z}} \leftarrow flo_{\mathbb{R}}} \mathbf{V}_{{flo^i}_{\mathbb{R}} \leftarrow {ref^p}_{\mathbb{Z}}}(i,j,k) \Big) \end{aligned}$$

4.1.2 Pyramidal block matching

1. Construction des parametres pour chaque niveau
2. Boucle sur les niveau
   (a) Allocation de la transformation du niveau
   (b) matrice de reechantillonnage
   (c) reechantillonnage de l'image de reference
   (d) reechantillonnage de la transformation courante a la transformation du niveau
   (e) filtrage eventuel de l'image flottante
   (f) recalage
   (g) reechantillonnage de la transformation resultat a la transformation courante
   (h) liberation memoire

4.2 Pyramidal_Block_Matching()

1. Initialisation de la transformation. Rééchantillonnage de flo vers ref.

   $$Tr\_result = T_{ref \leftarrow flo}$$

   J'aurai vu l'inverse.

2. Contrôle sur la taille du voisinage d'exploration.

3. Calcul du nombre de niveau de la pyramide

4. Allocation d'une image en cas de pyramide filtrée.

5. Pyramide

   (a) Mise à jour des paramtres (pas au 1e niveau)
   (b) Calcul de l'image ref sous-échantillonnée et de la matrice de sous-'echantillonnage de ref vers ref_sub
   (c) Lissage éventuel de l'image flottante
   (d) Appel de Block_Matching() la matrice de transformation est en coordonnées voxel de ref vers flo

4.3 Block_Matching()

Pourquoi on passe le type de Inrimage_flo ?

1. Composition de la matrice de sous-échantillonnage avec

4.4 Fonctions

4.4.1 ChangeGeometryTransformation(), transformation.h

4.5 Notes

- la position du centre d'un bloc (cf CalculChampVecteur2D() et CalculChampVecteur3D()) est donnée par $x = a + param- > bl\_dx/2.0$ avec a origine du bloc, et $param- > bl\_dx$ taille du bloc. Etant donné que a est au centre du bloc, il aurait fallu faire $x = a + (param- > bl\_dx - 1)/2.0$!! [DONE]

- dans CalculChampVecteur3D(), la taille du voisinage est de -$param- > bl\_size\_neigh\_x$ a +$param- > bl\_size\_neigh\_x$, et on parcourt le voisinage a partir de $a - param- > bl\_size\_neigh\_x$ et en avançant de $param- > bl_next_neigh_x$, ce qui fait que l'on ne teste pas forcément le point central (ex: size = 3 et step = 2). A corriger.

- Le calcul des résidus devrait se faire par une distance de Mahalanobis.
  - Paires : (M_i, M'_i)
  - Transformation estimée : $T = \arg\min \sum_i \|T \circ M_i - M'_i\|^2$
  - vecteurs résidus : $\mathbf{r}_i = T \circ M_i - M'_i$
  - moyenne des résidus : $\overline{\mathbf{r}} = \frac{1}{N} \sum_i \mathbf{r}_i$
  - matrice de covariance des résidus : $C = \frac{1}{N} \sum_i (\mathbf{r}_i - \overline{\mathbf{r}})(\mathbf{r}_i - \overline{\mathbf{r}})^T$
  - la distance de Mahalanobis s'écrit $(\mathbf{r}_i - \overline{\mathbf{r}})^T C^{-1} (\mathbf{r}_i - \overline{\mathbf{r}})$

  Maintenant, il faudrait calculer la distance par rapport à un résidu nul...
- tools = outils, basic = allocation, initialisation, ...
- transformation_type n'a pas à être dans les paramètres de recalage, juste dans les paramètres locaux du 'main'