Image deformation model in CASToR

September 18, 2020

Foreword

CASToR is designed to be flexible, but also as generic as possible. Any new implementation should be thought to be usable in as many contexts as possible; among all modalities, all types of data, all types of algorithms, etc.

Before adding some code to CASToR, it is highly recommended to read the general documentation *CASToR_general_documentation.pdf* to get a good picture of the project, as well as the programming guidelines *CASToR__programming_guidelines.pdf*. Also, the philosophy about adding new modules in CASToR (*e.g.* projectors, optimizers, deformations, dynamic models, etc) is fully explained in *CASToR__add_new_modules.pdf*. Finally, the doxygen documentation is a very good resource to help understanding the code architecture.

1 Summary

This HowTo guide describes the image deformation class in CASToR. It provided information about how to use the image deformation models already implemented in CASToR, or how to add a new image transformation class.

CASToR can manage different type of image-based motion correction, depending on the way the transformation operation is triggered (based on timestamps or specific number of gates and events). This allows to manage several type of motion correction during reconstruction, such as physiological motion (with respiratory and/or cardiac gated acquisition), as well as involuntary body motion. It is assumed that only one type of motion correction can be enabled in a reconstruction, therefore involuntary motion correction cannot be performed with either respiratory or cardiac motion. Simultaneous respiratory and cardiac motion can be performed using a unique set of transformations handling both types of motion.

This guide begins with a description of the preprocessing of the dataset, and the command-line options to enable motion-corrected reconstruction with castor-recon. Section 3 presents the basic for the initialization of a deformation model. The class currently implemented in castor are presented in the following section. The last sections are dedicated to the incorporation of a custom deformation model in CASToR, starting with a presentation of the overall CASToR deformation architecture (section 5). Finally, section 6 provides a step-by-step guide that explains how to add a new image deformation by simply adding a new class with few mandatory requirements.

2 Before using any image-based deformation model

2.1 Pre-requisites

Motion management in CASToR differs from the type of motion. The reconstruction of respiratory or cardiac gated acquisitions require a prior reorganization of the events in the dataset into *gates*, as well as the number of events in each gate. The correction of accidental body motion in reconstruction only requires the timestamp of the transformations. Please have a look at section 7 of the general documentation for a description of how dynamic datafiles are handled by CASToR for each type of motion.

2.1.1 Motion correction for gated acquisition (-rm, -cm)

Gated-acquisition are usually performed to adress physiological motion such as respiratory and cardiac motions. Due to the cyclic nature of these motions, the CASToR implementation requires a specific organization of the datafile. The implementation of image-based correction for these types of motion assumes the elements of the raw datafile (list-mode lines of response, or histogram bins) has previously been sorted into different subsets (*gates*), as described in section 7 of the general documentation file. Each *gate* corresponding to a specific phase of the motion surrogate (e.g, to a respiratory or cardiac phase/amplitude).

The command-line options to enable motion-correction for gated acquisitions depend on the nature of motion. The *-rm*, *-cm* command-line options allow to set up a deformation model for respiratory or cardiac motion-corrected reconstruction respectively. These option must be used along with the *-g* option, which provides a configuration file containing the number of gates and the number of events in each gates of the dataset. Section 3 describes how to set the deformation model with *-rm* or *-cm*.

2.1.2 Motion correction for involuntary patient motion (-im)

The implementation of image-based body motion correction requires the timestamp of the transformations to correct the data. During image reconstruction, the transformation operations will be performed when the timestamp of the event is greater than the next timestamp of the deformation. To set up a deformation model for this kind of motion, the command-line option *-rm* must be used. The timestamp of the transformations must be provided from a file with the *-g* option, as described in section 7 of the general documentation. The next section of the document describes how to set the deformation model with *-im*.

3 Deformation model initialization

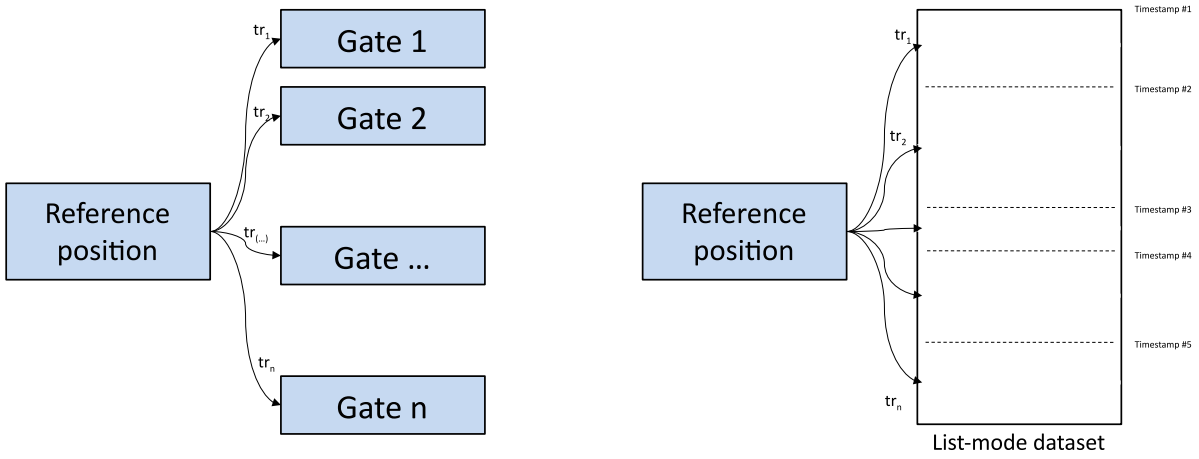
A deformation module can be enabled in order to perform image-based transformations. The **-rm**, **-cm** and **-im** command line options (for respiratory, cardiac and involuntary patient motion correction respectively) must be used to provide the image deformation information. Depending on the model, the initialization can be performed with a set of parameters, or with a configuration file, using the following syntaxes:

```
-rm model,option1,option2,...  -rm model:path/to/configuration/file
```

The first line corresponds to initialization with a set of options (such as transformation parameters). The alias of the deformation model must be written first, followed by the parameters, separated by commas. The second line corresponds to an initialization with a configuration file containing and associated parameters. The alias of the deformation model must be written first, followed by a semi-colon, and the path to the configuration file.

Transformation vectors: During image reconstruction, the image-based motion correction operations will be applied when the code reach a trigger computed from the number elements in each gate (for gated dataset), or from a timestamp (for patient motion). These operations requires a set of *forward* transformations (from a reference position, corresponding to the position in which the images will be reconstructed, to any other position (gate or patient motion time subset) of the dataset), and *backward* transformations (from any position to the reference position). The reference position can be any of the gate (or part of data corresponding to a time subsets in the case of accidental motion).

Figure 1 illustrates the transformations required for motion correction. For gated dataset (fig. 1(a)), both *forward* and *backward* transformations must be provided between the reference position and each gate position. In the case one gate corresponds to the reference position, identity parameters must be provided. Likewise, for timestamp based motion, transformation parameters must be provided for each subset of data between two timestamps.



(a) Gated based motion

(b) Timestamp based motion

Figure 1: Definition of *forward* (left arrows) and *backward* (right arrows) transformations between the gates of the dataset

This option allows to initialize a rigid or elastic deformation model and associated parameters (see details in section 7). CASToR contains a rigid (**deformationRigid**, section4.1) in order to perform image-based transformation, as described below.

4 Implemented deformation model

This section lists the deformation model implemented in CASToR. Currently, only a rigid deformation model is implemented. Details about the available deformation models and information regarding their initialization can be displayed using the command **-help-motion**.

4.1 deformationRigid:

This class uses rigid deformations whose transformation vectors are parameterized by 3 translations and 3 rotations. As most modules, it can be initialized from the command-line options:

```
-im deformationRigid,tx1,ty1,tz1,ra1,rb1,rc1,tx2,ty2,tz2,ra2,rb2,rc2,...
```

where tx, ty, tz represents the translations along the different axis, and ra, rb, rc represents the 3 rotations to perform along specific axis (x, y, and z) by default.

Or using a configuration file : `-im deformationRigid:/path/to/conf/file.txt`

The listing below displays an example of configuration file. Each transformation parameters (3 translations (mm), 3 rotations (degrees)) must be entered on a separate line.

The *Rotation_convention* parameter defines the axis for the rotation of the 3 angles according to the different conventions (*XYZ*, *ZYX*, *XYX*, etc.. default is *XYZ*).

The forward and backward deformations will be automatically computed from the parameters provided by *Transformation_parameters*. By default, the transformation parameters must correspond to the transformation between the reference position i to each subset (or gate), as illustrated in figure 1.

One can also provide parameters from the position i to the next position *i+1* (usually assuming that the first gate/subset is the reference). In this case, the *Transformation_mode* parameter must be set to 1.

Listing 1: Example of a configuration file for rigid transformation.

```
1
2 // Parameters for rigid deformation (tx, ty, tz, rx, ry, rz)
3 Transformation_parameters:
4 0,0,0,0,0,0
5 1,2,-3.5,-3.5,-2,4.5
6 3,-0.5,-4.25,3.75,-2,4.25
7 1,2,1.5,-0.75,-3.5,-3
8 -0.75,0.5,-2,-4,-3.25,-3
9 -0.5,3,2.5,1.75,-0.5,-3.5
10
11 // Convention of rotation (any combination of XYZ axis)
12 Rotation_convention: XYZ
13
14 // Parameters computation:
15 // 0: Each line of parameters represent the actual deformation in ↩
    comparison with the reference position
16 // 1: Each line of parameters represent the deformation with the previous↩
     position.
17 Transformation_mode: 0
```

5 The image deformation architecture

The image deformation part of the code is based on 2 main classes: *oDeformationManager* and *vDeformation*. The main program will instantiate and initialize the *oDeformationManager*, and during the iterative reconstruction process, the *oDeformationManager::PerformDeformation()* function will apply the required deformations on the different image matrices when the event counter hits a trigger (a certain number of events in the case of gated dataset, or a certain timestamp for involuntary patient motion). The various steps of application of the deformations on the different image matrices of the algorithm (forward image containing the image to project, backward image containing the correction factors, etc...) is natively managed by the *vDeformation* mother class, so that the developper doesn't need to handle this in his deformation class.

The *vDeformation* class only requires that the child implements the *ApplyDeformations()* function, which will perform the deformation on an input image matrix, and recover the result in an output image matrice, both passed in argument. Please have a look to the section **??** for more help about the deformation model implementation itself.

It is worthy to note that the deformation process itself is NOT multithreaded, therefore a synchronization of the threads will be performed before each image deformation. This is to limit the RAM requirements on some image matrices which should not be multithreaded a-priori.

Regarding the sensitivity image of list-mode reconstruction, the algorithm will generate an image corresponding to the reference position using the *Forward* and *Backward* transformations. The algorithm will simply loop on the total number of transformations, generate a reference image for each set of *Forward* and *Backward* transformation, and average all these images to get a standard reference sensitivity image.

6 Add your own deformation model

6.1 Basic concept

To add a new image-based deformation module, the user only has to build a specific class that inherits from the abstract class *vDeformation*. Then, one just has to implement a set of pure virtual functions for the initialization and application of the model. Please refer to the *CASToR__add_new_modules.pdf* guide in order to fill up the mandatory parts of adding a new module; namely the auto-inclusion mechanism, the interface-related functions and the management functions. Right below are some instructions to help you fill the specific pure virtual projection functions of deformation model.

To ease the implementation, a template class is provided in the source code and already implements all the squeleton. Basically, one will have to change the name of the class and fill the related functions up in his own code. The actual files are *include/image/iDeformationTemplate.hh* and *src/image/iDeformationTemplate.cc* and are actually already part of the source code.

6.2 Implementation of the deformation functions

Several mandatory functions should be implemented (or return an error by default) in a new deformation model class :

ShowHelp(): Simply output some help and guidance to describe what the deformation model does and how to use it.

ReadAndCheckOptionsList(): Implement here the reading of any options specific to this deformation model passed through the argument `const string& a_listOptions`. The user can make use of the *ReadStringOption()* function to parse the list of parameters. If the function is not mandatory (e.g initialization of the model using a file is required tather than with command-line parameters), just send an error message and return 1.

ReadAndCheckConfigurationFile(): Implement here the reading of any configuration file specific to this deformation model, passed through the argument `const string& a_fileOptions`. The user can make use of the *ReadDataASCIIFile()* function to read data from a file. If the function is not mandatory (e.g initialization of the model using command line options is required rather than with a file), just send an error message and return 1.

CheckParameters(): Use this function to check if the private parameters of your class have correctly been initialized. It might be a good idea to initialize all private parameters in the constructor with default erroneous value to properly check their initialization.

Initialize(): Use this function to Instanciate/Initialize any member variables/arrays after the parameters have been checked in the previous function.

ApplyDeformation(): The deformation model must be implemented here, with the help of any private functions if required. This function is described in the pseudocode below. This function has 4 arguments:

`ap_inputImage` : vector containing the input image to deform

`ap_outputImage` : vector in which the transformed image must be recovered

`a_direction` : integer which indicates the direction of the deformation to perform, i.e:

`FORWARD_DEFORMATION` (from the reference position to the `a_defIdx` position),

`BACKWARD_DEFORMATION` (from the `a_defIdx` position to the reference position).

The integers `FORWARD_DEFORMATION` and `BACKWARD_DEFORMATION` are defined in the beginning of vDeformation.hh as macros

`a_defIdx` : defines the index of the transformation

All information and the tools needed to implement these functions are fully described in the template source file *src/projector/iDeformationTemplate.cc*, so please refer to it.

Listing 2: Description of ApplyDeformations function.

```
1
2  int iDeformationTemplate::ApplyDeformations(FLTNB* ap_inputImage,
3                                              FLTNB* ap_outputImage,
4                                              string a_direction,
5                                              int a_defIdx)
6  {
7    // This function is an implementation of the pure virtual mother ↩
       function. The actual deformation should be implemented here, with the ↩
       help of any private functions if required
8
9    // IMAGE DIMENSIONS:
10   // For code efficiency and readability, the spatial index of a voxel is↩
       a cumulative 1D index.
11   // That is to say, given a voxel [indexX,indexY,indexZ],
12   // its cumulative 1D index is computed by 'index = indexZ*nbVoxXY + ↩
       indexY*nbVoxX + indexX'.
13
14   // The image dimensions can be recovered from the ↩
       mp_ImageDimensionsAndQuantification class
15   // Total number of voxels         :
16   mp_ImageDimensionsAndQuantification->GetNbVoxXYZ()
17   // Number of voxels in a slice    :
18   mp_ImageDimensionsAndQuantification->GetNbVoxXY()
19   // Number of voxels on the X-axis :
20   mp_ImageDimensionsAndQuantification->GetNbVoxX()
21
22
23   // Any error should return a value >0.
24
25   return 0;
26 }
```

7 Meta-data command line options

List of the different set of options related to image-based deformations :

- **-g** *file*: Give a text file defining the dynamic data splitting (due to respiratory/cardiac gating or involuntary patient motion correction). The file should contain the some of the following keywords :

  `nb_respiratory_gates`: number of respiratory gates in the data

  `nb_events_respiratory_gates`: enter the number of events within each gate, separated by commas. If the data contains several frame (dynamic acquisition), the data splitting of each frame should be entered on a new line (1 line by frame).

  `nb_cardiac_gates`: number of cardiac gates in the data

  `nb_events_cardiac_gates`: enter the number of events within each gate, separated by commas. If the data contains several frame (dynamic acquisition), the data splitting of each frame should be entered on a new line (1 line by frame).

  `nb_involuntary_motion_triggers`: number of involuntary patient motion triggers in the data

  `list_involuntary_motion_triggers`: enter the timestamp of each transformation, separated by commas. No specific nomenclature is required for dynamic acquisitions (data containing several frames). However, if the data contains several bed positions, the data splitting of each bed should be entered on a new line.

  `duration_gate`: (optional) enter the duration (seconds) of each gate, separated by commas. If the data contains several frames (dynamic acquisition), the gate durations within each frame should be entered on different line (1 line by frame).

- **-rm** *param*: Give the deformation to be used for respiratory motion correction, along with a configuration file (deformationModel:file), or the list of parameters associated to the deformation model (deformationModel,param1,param2,...).

- **-cm** *param*: Give the deformation to be used for cardiac motion correction, along with a configuration file (deformationModel:file), or the list of parameters associated to the deformation model (deformationModel,param1,param2,...).

- **-im** *param*: Give the deformation to be used for patient motion correction, along with a configuration file (deformationModel:file), or the list of parameters associated to the deformation model (deformationModel,param1,param2,...).