Projectors and system matrices in CASToR

October 22, 2024

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 new 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, image processing, etc) is fully explained in *CASToR__add_new_modules.pdf*. Finally, the doxygen documentation is a good resource to help understanding the code architecture.

1   Summary

This HowTo guide describes how to add your own projector into CASToR or how to use your own pre-computed system matrix. CASToR is mainly designed to be modular in the sense that adding a new feature should be as easy as possible. This guide begins with a brief description of the projector part of the CASToR architecture that explains the chosen philosophy. Then follows one step-by-step guide that explains how to add a new projector by simply adding a new class with few mandatory requirements, and another guide to use your own pre-computed system matrix by observing some simple rules.

2   The projector architecture

The projector part of the code is based on 4 main classes: *oProjectorManager*, *oProjectionLine*, *vProjector* and *oSystemMatrix*. To make a long story short, the main program will instantiate and initialize the *oProjectorManager*, and during the reconstruction process, the *oProjectorManager::ComputeProjectionLine()* function will be used to get a *oProjectionLine* from the current event provided as a parameter. The *oProjectionLine* is somewhat a container that holds the system matrix elements computed by a *vProjector* or loaded by the *oSystemMatrix*, with respect to the data channel associated to the event.

The *oProjectorManager*, being the manager, is in charge of reading command line options and instantiating either a *vProjector* or a *oSystemMatrix* with respect to what the user asks for. Forward and backward operators can be different and of any type. The *vProjector* is an abstract class so only its children can be used as actual projectors. It corresponds to on-the-fly projectors such as Siddon for example. The *oSystemMatrix* class can directly be used to load your own pre-computed system matrix, as long as you observe some mandatory rules about the format of the system matrix. Note that time-of-flight PET data cannot be reconstructed using a pre-computed system matrix.

When a *vProjector* is used, the *oProjectorManager* calls the *vProjector::Project()* function. In this function, the scanner is called to compute two cartesian coordinates associated to this event, providing the line-of-response (LOR). The compression is also managed (*i.e.* when multiple physical

LORs are contributing to an event) by averaging the multiple cartesian coordinates associated to each point of the LOR. Then, based on the data type (*i.e.* modality), the data mode (*i.e.* histogram or list-mode) and whether the time-of-flight information is used in the case of PET data, one over three different projection functions is called. These three different functions are the following:

ProjectWithoutTOF() : This function is used for all non-PET modalities, and for PET without TOF data. Given two points, it simply computes the path of the line through the image.

ProjectTOFListmode() : This function is used for PET list-mode data with continuous TOF information, that is to say with the original TOF measurement provided in units of time. Given two points and the TOF measurement, it should compute the path of the line through the image while applying a Gaussian kernel centered on the TOF position and of FWHM corresponding to the TOF resolution of the data.

ProjectTOFHistogram() : This function is used for PET histogrammed data with binned TOF information, that is to say with an additional TOF dimension over the histogram. Given two points and the TOF bin, it should compute the path of the line through the image while applying the TOF bin function, which is obtained by convolving the Gaussian kernel (centered on the center of the TOF bin and of FWHM corresponding to the TOF resolution of the data) convolved with a box function whose width equals the TOF bin width.

3   Implemented projectors

Several widespread projectors have already been implemented, this includes Siddon (original and incremental version), Joseph, and Distance-driven projector. For a complete and exhaustive list of all available projectors, use the related help option directly within the CASToR program. All the projectors take into account only voxels pertaining both to the provided field of view and to the line-of-response.

3.1   TOF

There are several ways for implementing TOF projection, using more or less approximations and computational tradeoffs. The following method is used for all currently implemented projectors (Siddon, Joseph, Distance-driven). The TOF uncertainty function is a Gaussian function with a given FWHM, normalized so that its integral equals 1.

The main difference between TOF and nonTOF projection coefficients (system matrix elements) lies in the coefficient component which stands for the length of the line-of-response through a voxel (integration along the LOR through the voxel). For list-mode data, this component is replaced by the integration of the TOF Gaussian uncertainty function along the LOR through the voxel. For histogram TOF bin data, this component is replaced by an approximation of the integration (value of the TOF Gaussian function multiplied by the length of the line-of-response through the voxel). The width of the TOF bin is not taken into account here, because convolution can be costly. Instead, since it should always be true that the sum of all TOF bin coefficients for a single voxel equals the non TOF coefficient for this voxel, the TOF bin coefficients are scaled by ensuring, somewhat artificially, that this condition is met. This implies that the convolution with the TOF bin width can be approximated by a single multiplicative factor for all TOF coefficients for a single voxel.

By default, the TOF Gaussian function is truncated at 3 standard deviations, but this can be changed using a specified projector parameter.

For more details about the undelying equations and their implementation, please refer to the document dedicated to TOF.

4 Add your own projector

4.1 Basic concept

To add your own projector, you only have to build a specific class that inherits from the abstract class *vProjector*. Then, you just have to implement a bunch of pure virtual functions corresponding to what you want your new projector specifically to realize. Please refer to the *CASToR__add_new_modules.pdf* guide in order to fill up the mandatory parts of adding a new module (your new projector is a 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 your projector.

To make things easier, we provide an example of a template class that already implements all the squeleton. Basically, you will have to change the name of the class and fill the functions up with your own code. The actual files are *include/projector/iProjectorTemplate.hh* and *src/projector/iProjectorTemplate.cc* and are actually already part of the source code. Also, we recommend that you take a look at other implemented projectors.

4.2 Implementation of the projection functions

The projection functions that you have to implement are the three ones mentionned in the previous section: *ProjectWithoutTOF()*, *ProjectTOFListmode()* and *ProjectTOFHistogram()*. All information and the tools needed to implement these functions are fully described in the template source file *src/projector/iProjectorTemplate.cc*, so please refer to it.

For each projector, one must specify in the constructor if the projector is compatible with SPECT attenuation correction. If all voxels contributing to a projection line are added to the *oProjectionLine* in an ordered manner, from the outside to the detector (point1 to point2), then CASToR will be able to automatically manage the attenuation correction for SPECT data (assuming obviously that an attenuation map has been provided). If your projector meets this requirement, then do not forget to set the boolean member *m_compatibleWithSPECTAttenuationCorrection* to true in the constructor of your projector; otherwise it is set to false by default in the constructor of *vProjector*.

5 Use your own pre-computed system matrix

This feature is not yet implemented in the CASToR code.