

Image processing modules in CASToR

October 26, 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 overview 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 very good resource to help understanding the code architecture.

1 Summary

This HowTo guide describes how to add your own image processing module into CASToR. 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 image processing part of the CASToR architecture that explains the chosen philosophy. Then follows a step-by-step guide that explains how to add a new image processing module by simply adding a new class with few mandatory requirements.

2 The image processing architecture

The image processing part of the code is based on 2 main classes: *oImageProcessingManager* and *vImageProcessingModule*. The main program will instantiate and initialize the *oImageProcessingManager*. It is in charge of reading command line options and instantiating the different children of *vImageProcessingModule*. As many image processing modules as desired can be used/combined. Each one is declared by the option *-proc* when executing the program. To get some help on how to use it and a list of the implemented processing modules, execute the program with the option *-help-proc*. The option format follows the philosophy described in *CASToR_HowTo_add_new_modules.pdf*. Any image processing module can be used at different places during the program execution. For iterative algorithms, it can be used on the image to be forward-projected, on the back-projected correction terms, on the current estimated image either as a post-processing step (applied to the image to be saved), or as an intra-iterations processing (the processed image is put back into the next update as the current estimate).

During the initialization the *oImageProcessingManager* will call the *InitializeSpecific()* function of each image processing module. This function is pure virtual and should implement everything that needs to be done to be able to apply the processing. Then, during the execution of the program, the function *Process()* is called to actually apply the processing onto the image provided as a parameter. This is also a pure virtual function.

Below is a more detailed description of how the image processing modules are used and how to add your own.

3 Add your own image convolver

3.1 Basic concept

To add your own image processing module, you only have to build a specific class that inherits from the abstract class *vImageProcessingModule*. Then, you just have to implement a bunch of pure virtual functions that will correspond to the specific stuff you want your new image processing module to do. 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 functions of your image processing module.

To make things easier, we provide an example of template class that already implements all the skeleton. 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/image/iImageProcessingTemplate.hh* and *src/image/iImageProcessingTemplate.cc* and are actually already part of the source code.

Based on what your processing module will do, the following boolean members have to be set in the constructor of the module:

Listing 1: Boolean members specifying the actions of the module on the different dynamic dimensions.

```
1  // A boolean that specify if the module is affecting the time frame ↔
   dynamic dimension
2  bool m_affectTimeDimensionFlag;
3  // A boolean that specify if the module is affecting the respiratory ↔
   dynamic dimension
4  bool m_affectRespDimensionFlag;
5  // A boolean that specify if the module is affecting the cardiac ↔
   dynamic dimension
6  bool m_affectCardDimensionFlag;
```

The *oImageProcessingManager* will use these booleans during the initialization to check the consistency between how the dynamic dimensions are modeled and the processing modules in use. For instance, if your processing module is acting on the time frames and modifying them, then the boolean *m_affectTimeDimensionFlag* should be set to *true* in the constructor. In that case, if the frame dimension is modeled by some basis functions, then the use of this processing module will automatically be forbidden.

3.2 Implementation of the specific functions

In addition to the initialization functions, there is only one pure virtual function to implement: *Process()*. This function will actually perform the processing. It has only one parameter which is a pointer to the dynamic image to be processed. Note that the result of the processing should also be put into this image. The dynamic dimensions of the image are the standard ones used in CASToR. All information and the tools needed to implement this function are fully described in the template source file *src/image/iImageProcessingTemplate.cc*, so please refer to it.