BRAINSFit: Mutual Information Rigid Registrations of Whole-Brain 3D Images, Using the Insight Toolkit

Release 0.00

Hans Johnson[1], Greg Harris[2] and Kent Williams[3]

October 5, 2007

[1]hans-johnson@uiowa.edu
[2]gregory-harris@uiowa.edu
[3]norman-k-williams@uiowa.edu
The University of Iowa Carver College of Medicine,
Department of Psychiatry NeuroImaging Center
200 Hawkins Drive, Iowa City, IA 52242

Abstract

The University of Iowa's Psychiatric Iowa Neuroimaging Consortium (PINC) has developed a program for mutual information registration of `BRAINS2` [2] data using `ITK` [1] classes, called `BRAINSFit`.

We have written a helper class, `itk::MultiModal3DMutualRegistrationHelper` to simplify implementation and testing of different transform representations and optimizers. We have added a transform meeting the `ITK` standard, `itk::ScaleVersor3DTransform`. `BRAINSFit` is based on the registration examples from `ITK`, but adds new features, including the ability to employ different transform representations and optimization functions.

Our goal was to determine best practices for registering 3D rigid multimodal MRI of the human brain. A version of the current program is employed here at PINC daily for automated processing of acquired brain images.

*Acknowledgments

The following people should be recognized for their contributions: Vincent A. Magnotta, Norman Kent Williams, Hans J. Johnson, Gregory Harris, Steven Pieper.

The `BRAINS2` software was developed with the leadership of Nancy C. Andreasen, M.D., Ph.D. This work is supported by NINDS grant 5R01NS050568, and NIMH Grants: MH31593, MH40856, and MHCRC43271.

Contents

1 Automating the Image Registration Process

We have developed a program for mutual information registration of brain imaging data using `ITK` [1] classes. Our program, `BRAINSFit`, was based on an example program included in the `ITK` distribution,

`Insight/Examples/Registration/ImageRegistration8.cxx`

This program is the most functional example of multi-modal 3D rigid image registration provided with `ITK`. `ImageRegistration8` is in the `Examples` directory, and also sec. 8.5.3 in the `ITK` manual.

We have modified and extended this example in several ways:

- defined a new itk Transform class, based on `itkScaleSkewVersor3DTransform` which has 3 dimensions of scale but no skew aspect.
- implemented a set of functions to convert from `Versor` Transforms to `itk::AffineTransform` before saving to file.
- Added a template class `itkMultiModal3DMutualRegistrationHelper` which is templated over the type of `ITK` transform generated, and the optimizer used.

- Added image masks as an optional input to the Registration algorithm, limiting the volume considered during registration to voxels within the brain.
- Defined the command line parameters using tools from the Slicer [3] program, in order to conform to the `Slicer3` Execution model.
- Added the ability to write output images in any `ITK`-supported scalar image format.
- Through extensive testing as part of the `BRAINS2` determined reasonable defaults for registration algorithm parameters.

1.1 A new transform class

`itkScaleVersor3DTransform` is a new `ITK Transform` class, which is a modification of `itkScaleSkewVersor3DTransform` to remove the skew factors from the transform. The result is a 9-parameter transform, comprising three dimensions each of rotation, translation and scale. `itkScaleVersor3DTransform` is particularly useful in registration of brain images, which commonly have symmetric size variations in all three dimensions.

1.2 Conversion Routines for Versor transform types to Affine

We implemented a `vnl_svd`-based 3×3 matrix orthonormalization routine and use it when coercing our `itkScaleVersor3DTransform` to an `itkVersorRigid3DTransform`. In addition, we implemented assignment functions for converting all supported transform types to `itk::AffineTransform`. This was originally a requirement for integration with `BRAINS2`, but could be used in other programs as well.

1.3 A helper class to build an ITK pipeline

The new class `itkMultiModal3DMutualRegistrationHelper` encapsulates the complete processing pipeline for mutual information registration. This class is parameterized over the output transform type, optimizer type, input image type and output image type. This class captures all of the code common to all forms of the Mutual Information Registration algorithm, such that only high-level configuration parameters need be specified by the calling program.

This allows the `MattesMutualInformation` program to be a concise workbench for evaluating and using different transform and optimizer types.

1.4 Using Masks for Registration

`itk::ImageToImageMetric` and all of its descendents (`itk::MattesMutualInformationImageToImageMetric`, for example) can use mask images to limit the voxesl considered during registration to the relevant region of the input and output regions. This can improve performance somewhat, both in time consumed and the quality of the resulting registration. The origin `ImageRegistration8` program didn't include a provision for using masks, so we added it.

1.5 Using the Slicer3 Execution Model for command line parameters

The Slicer3 Execution Model is a way for a program to function as both a command line program, as as a subroutine within the Slicer3 environment. The command line parameters are specified in an XML file that includes documentation/help strings. This file is read by the `Slicer3` utility `GenerateCLP` which generates the code for handling the program's command line. In addition to command line parsing, it can reproduce the XML describing all parameters, which Slicer3 can us to build a user interface panel for the program.

1.6 Output Image Pixel Types

We added the command line parameter `--OutputImagePixelType`, which specifies one of `float`,`short`,`ushort`,`int`,`uint`,`char`, or `uchar`. The most common image pixel types for MRI brain scans is 16 bit integers (bcodeshort) or unsigned 8 bit char (`uchar`), but `BRAINSFit` internally uses single precision floating point pixels for the registration process. Consequently, by default `BRAINSFit` writes out images with single precision pixels.

1.7 Tuning of default program parameters

The program presented in this project has been used for some time as part of the standard image processing pipeline used at the University of Iowa for brain imaging studies. Using image registration programs is in general somewhat difficult because of the large number of parameters to the registration algorithm that need to be tweaked to get meaningful results.

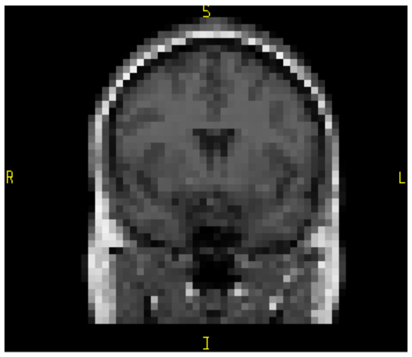
As a result of the Iowa experience with using this MutualRegistration program, we have set program default parameters to the set of parameters deemed the best behaved over the course of registering many scans. A good registration fit is in most cases attained with no more program input than the fixed and moving image file names and the output transform and/or image filename.

2 Example Run

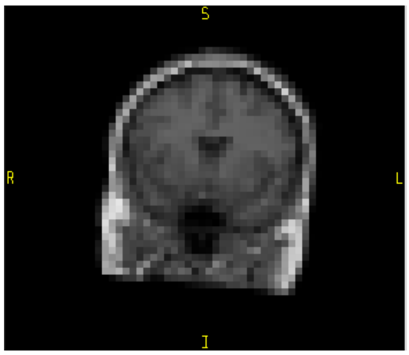
The `BRAINSFit` distribution contains a directory named `TestData`, which contains two example images. The first, `test.nii.gz` is a `NIfTI` format image volume, which is used the input for the `CTest`-managed regression test program. The program `makexfrmedImage.cxx`, included in the `BRAINSFit` distribution was used to generate `test2.nii.gz`, by scaling, rotating and translating `test.nii.gz`.

You can see representative Sagittal slices of `test.nii.gz` (in this case, the fixed image, `test2.nii.gz` (the moving image), and the two images 'checkerboarded' together in Figure 1. To register `test2.nii.gz` to `test.nii.gz`, you can use the following command:

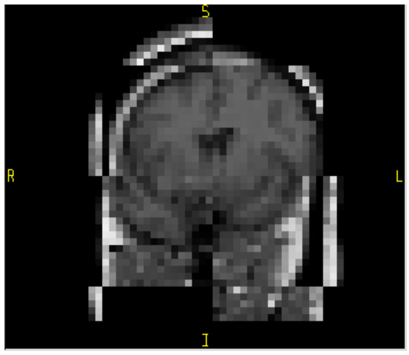
```
BRAINSFit --FixedImage test.nii.gz \
--MovingImage test2.nii.gz \
--OutputImage registered.nii.gz \
--FitTransformtype Affine
```



Fixed Image

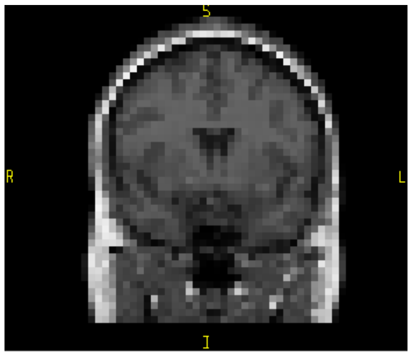


Moving Image

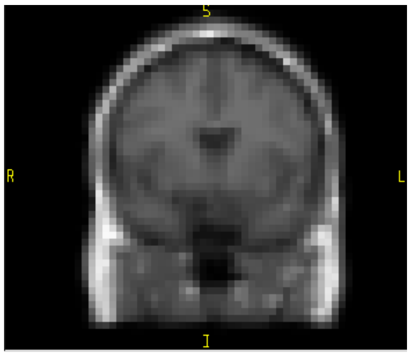


Checkerboard

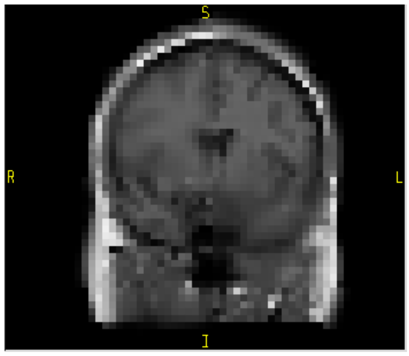
Figure 1: Registration Inputs



Fixed Image



Registered Image



Checkerboard

Figure 2: Registration Outputs

A representative slice of the registered results image `registered.nii.gz` is in the center of Figure 2. You can see from the Checkerboard of the Fixed and Registered images that the fit is quite good with Affine transform-based registration. The blurring of the registered images is a consequence of the initial scaling used in generating the moving image from the fixed image, compounded by the interpolation necessitated by the transform operation.

You can see the differences in results if you re-run `BRAINSFit` using `Rigid`, `ScaleVersor3D`, or `ScaleSkewVersor3D` as the `--FitTransformType` parameter. In this case, the authors judged `Affine` the best method for registering these particular two images; in the `BRAINS2` automated processing pipeline, `Rigid` usually works well for registering research scans.

3 BRAINSFit Usage

3.1 Required Parameters

`--MovingImage <filename>`	The moving image for registration
`--FixedImage <filename>`	The fixed image for registration
`--OutputTransform <filename>`	Filename to which save the estimated transform
`--StrippedOutputTransform <filename>`	File name for the estimated transform, stripped of scaling, to register the moving image to the fixed image.
`--OutputImage <filename>`	The (optional) output image for registration

3.2 Optional Parameters

`--OutputImagePixelType <pixel type>`	Pixel type for output image. One of `float`, `short`, `ushort`, `int`, `uint`, `char`, `uchar`(default: float)
`--StudyID <study name>`	Identifier for the scanner encounter (MRQID) (default: ANON)
`--PatientID <patient ID>`	Identifier for the research subject (default: ANON)
`--BackgroundFillValue <double>`	Background fill value for output image (default: 0)
`--MaximumStepLength <double>`	Internal debugging parameter (default: 0.2)
`--MinimumStepLength <double>`	Internal debugging parameter (default: 0.005)
`--MedianFilterRadius <double>`	The radius for the optional MedianImageFilter preprocessing. (default: 0)
`--RelaxationFactor <double>`	Internal debugging parameter (default: 0.5)

`--SkewScale <double>`	ScaleSkewVersor3D Skew compensation factor. Increase this to put more skew in a ScaleSkewVersor3D search pattern. (default: 10)
`--ReproportionScale <double>`	ScaleVersor3D 'Scale' compensation factor. Increase this to put more rescaling in a ScaleVersor3D or ScaleSkewVersor3D search pattern. (default: 25)
`--TranslationScale <double>`	Translation scale compensation factor. Decrease this to put more rotation in the search pattern (default: 1000)
`--NumberOfSpatialSamples <int>`	The number of spatial samples. Increase this for a slower, more careful fit. (default: 100000)
`--MaxNumberOfIterations <int>`	The maximum number of iterations to try beofe failing to converge. Use an explicit limit like 500 or 1000 to manage risk of divergence (default: 1500)
`--MovingImageOrigin <x,y,z>`	The coordinates of the AC point of the moving image (default: 0,0,0)
`--FixedImageOrigin <x,y,z>`	The coordinates of the AC point of the fixed image (default: 0,0,0)
`--OriginsProvided`	Use provided AC point origins for initial alignment (default: 0)
`--MovingImageTimeIndex <int>`	The index in the time series for the 3D moving image to fit, if 4-dimensional. (default: 0)
`--FixedImageTimeIndex <int>`	The index in the time series for the 3D fixed image to fit, if 4-dimensional. (default: 0)
`--ForceCoronalOrientation`	Permute axes of both the moving and fixed images so that they are both in coronal orientation prior to estimating registration. The resulting transform will be defined in terms of the coronal orientations and will only be valid when applied to images that are in coronal orientation. (default: 0)
`--MovingMask <filename`	Moving Image Mask
`--FixedMask <filename>`	Fixed Image Mask
`--InitialTransform <filename>`	Filename of transform used to initialize for registration
`--FitTransformType <type name>`	Specifies the ITK transform type generated. One of `<Rigid, ScaleVersor3D, Affine, ScaleSkewVersor3D` (default: Rigid)

3.3 GenerateCLP Global Options

`--processinformationaddress <std::string>`	Address of a structure to store process information (progress, abort, etc.). (default: 0)
`--xml`	Produce xml description of command line arguments (default: 0)
`--echo`	Echo the command line arguments (default: 0)
`--, --ignore_rest`	Ignores the rest of the labeled arguments following this flag.
`--version`	Displays version information and exits.
`-h,--help`	Displays usage information and exits.

4 Software Requirements

You need to have the following software installed:

- Insight Toolkit 3.4.
- CMake 2.4.

If you do not already have `ITK` and `CMake` installed, they can obtained via the Insight Toolkit Website: http://www.itk.org.

Note that other versions of the Insight Toolkit are also available in the testing framework of the Insight Journal. Please refere to the following page for details:

http://www.insightsoftwareconsortium.org/wiki/index.php/IJ-Testing-Environment

If you wish to add to or modify the command line parameters, you will need the `GenerateCLP` program which is part of the `Slicer3` application. Refer to the following pages for details:

http://www.slicer.org

http://www.na-mic.org/Wiki/index.php/Slicer3:Execution_Model_Documentation

5 Building BRAINSFit

The normal steps for building and installing `BRAINSFit` are as follows:

- Build and install CMake
- Build and (optionally) install the Insight Toolkit (`ITK`)
- Configure and build `BRAINSFit`

The first two steps are amply covered in the documentation for those prerequisite packages. Building `BRAINSFit` itself involves the following steps:

- Untar (or checkout from version control) the `BRAINSFit` distribution.
- create an empty directory to build the program in.
- run `cmake` in the build directory, pointing to the source code directory
- Once successfully configured, run `make` to build the program
- Optionally, install the program

The following shell script accomplishes all these steps:

```
# Create a directory to hold source and build directory
mkdir -p BRAINSFit-sandbox
cd BRAINSFit-sandbox

#
# check BRAINSFit out from the NITRC svn server
svn checkout https://www.nitrc.org/svn/multimodereg

# make directory for out-of-source build
mkdir BRAINSFit-build
#
# go into build directory
cd BRAINSFit-build
#
# run CMake to configure. ITK is only required prerequisite
# If you have a Slicer3 build on your system, adding
# -DGenerateCLP_DIR:PATH=${SlicerSource}/Libs/GenerateCLP
# -DGENERATECLP_EXE:FILEPATH=${SlicerBuild}/bin/GenerateCLP
#
# will allow you to change the command line description in BRAINSFit.xml
# ${SlicerSource} being the Slicer source directory and ${SlicerBuild} the
# build directory
#
# $ITKDIR -- either PREFIX/lib/InsightToolkit for an installation or the root
# of the build tree.
cmake -DITK_DIR:PATH=$ITKDIR ../multimodereg
make
make test
```

6 Project Home on NITRC.org

The `BRAINSFit` project is hosted on `http://www.nitrc.org`, The Neuroimaging Informatics Tools and Resources Clearinghouse. The project page itself is

`http://www.nitrc.org/projects/multimodereg`

NITRC is a project of the US National institutes of Health, started to give researchers access to neuroimaging software tools. Every project on NITRC has a variety of resources presented on the project page: a source code repository, bug tracker, mailing lists, forums, etc.

The most current source code (including the LaTex source files for this document) are always available from the project page given above. Users are encouraged to register on the NITRC page, so they can ask questions, make feature requests, share use experiences, and contribute bug reports.

References

[1] Luis Ibanez, Will Schroeder, Lydia Ng, and Josh Cates. *The ITK Software Guide: The Insight Segmentation and Registration Toolkit (version 1.4)*. Kitware Inc., September 2003. (document), 1

[2] V.A. Magnotta, G. Harris, N.C. Andreasen, W.T.C. Yuh, and D. Heckel. Structural MR image processing using the BRAINS2 toolbox. *Computerized Medical Imaging and Graphics*, 26:251–64, 2002. (document)

[3] Steve Pieper. The na-mic kit: Itk, vtk, pipelines, grids and 3d slicer as an open platform for the medical image computing community. *Proceedings of IEEE International Symposium on BioMedical Imaging: From Nano to Macro 2006*, pages 698–701, March 2006. 1