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]

August 21, 2009

[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 between `Versor` Transforms and the general `itk::AffineTransform` and deferred converting from specific to more general representations to preserve transform information specificity as long as possible. Our Rigid transform is the narrowest, a Versor rotation plus separate translation.
- 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.
- Added image mask generation as an optional input to the Registration algorithm when meaningful masks such as for whole brain are not available, allowing the fit to at least be focussed on whole head tissue.
- Added the ability to use one transform result, such as the Rigid transform, to initialize a more adaptive transform
- 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.

Masks were so useful in obtaining a more accurate optimizing convergence that we added automatic generation of whole head tissue masks as well.

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.

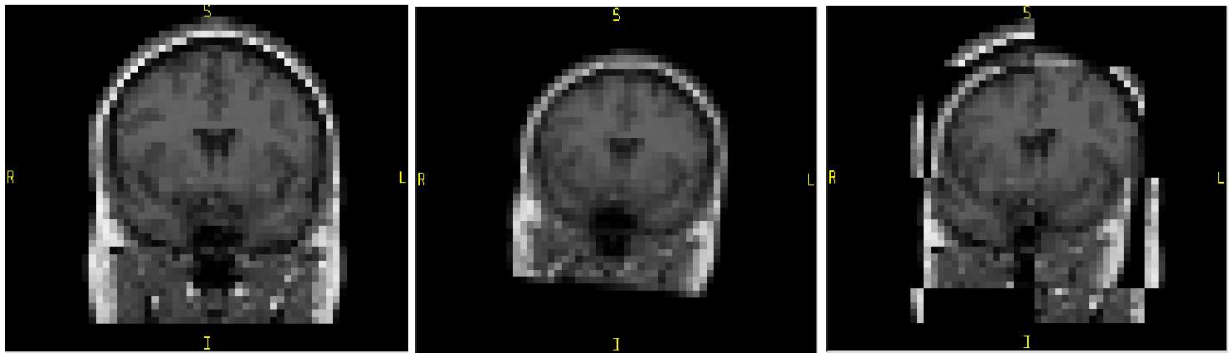
1.6 Output Image Pixel Types

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.



Fixed Image Moving Image Checkerboard

Figure 1: Registration Inputs

2 Example Run

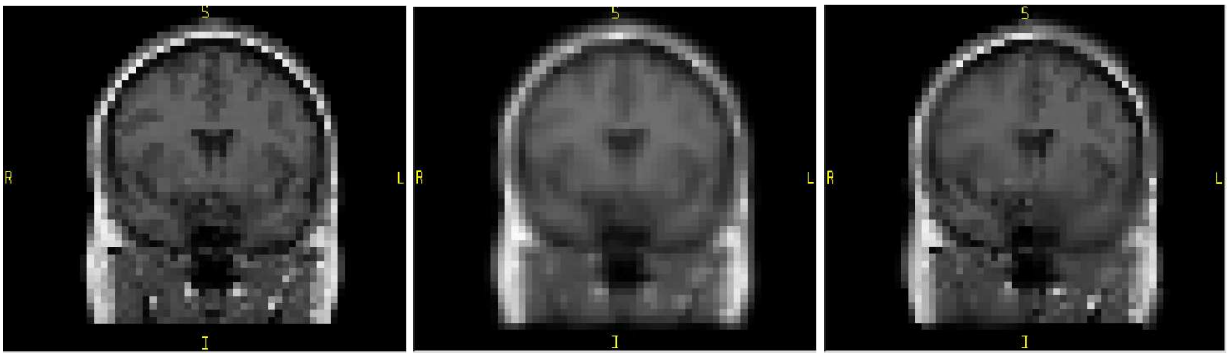
The `BRAINSFit` distribution contains a directory named `TestData`, which contains two example images. The first, `test.nii.gz` is a `NIfTI` format image volume, which is used the input for the `CTest`-managed regression test program. The program `makexfrmedImage.cxx`, included in the `BRAINSFit` distribution 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 --fixedVolume test.nii.gz \
--movingVolume test2.nii.gz \
--outputVolume registered.nii.gz \
--transformType Affine
```

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 `----transformType` 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.



Fixed Image Registered Image Checkerboard

Figure 2: Registration Outputs

3 BRAINSFit Usage

3.1 Required Input Parameters

`--fixedVolume <filename>`	The fixed image for registration by mutual information optimization.
`--movingVolume <filename>`	The moving image for registration by mutual information optimization.
`--transformType <type name>`	Specifies one of the four recognized ITK 3D transform types used in parameter optimization descent. BRAINSFit always optimizes mutual information, but the kind of descent varies with the transform type. One of `Rigid`, `ScaleVersor3D`, `Affine`, `ScaleSkewVersor3D` (default: `Rigid`)

3.2 Transform Configuration Parameters

`--initializeTransformMode <mode name>`	Determine how to initialize the transform. `GeometryOn` on assumes that the center of the voxel lattice of the images represent similar structures. `MomentsOn` assumes that the center of mass of the images represent similar structures. `Off` assumes that the physical space of the images are close, and that an identity matrix initialization is a good starting point. This flag is mutually exclusive with the initialTransform flag. One of `Off`, `MomentsOn`, `GeometryOn` (default: `Off`)
`--initialTransform <filename>`	Filename of transform used to initialize the registration

3.3 Important Registration Parameters

`--numberOfIterations <int>` The maximum number of iterations to try before failing to converge. Use an explicit limit like 500 or 1000 to manage risk of divergence (default: 1500)

`--numberOfSamples <int>` The number of voxels sampled for mutual information computation. Increase this for a slower, more careful fit. You can also limit the sampling focus with ROI masks and ROIAUTO mask generation. (default: 100000)

`--minimumStepSize <double>` Each step in the optimization takes steps at least this big. When none are possible, registration is complete. (default: 0.005)

`--spatialScale <double>` How much to scale up changes in position compared to unit rotational changes in radians – decrease this to put more rotation in the search pattern. (default: 1000.0)

`--reproportionScale <double>` ScaleVersor3D 'Scale' compensation factor. Increase this to put more rescaling in a ScaleVersor3D or ScaleSkewVersor3D search pattern. 1.0 works well with a spatialScale of 1000.0. (default: 1.0)

`--skewScale <double>` ScaleSkewVersor3D Skew compensation factor. Increase this to put more skew in a ScaleSkewVersor3D search pattern. 1.0 works well with a spatialScale of 1000.0. (default: 1.0)

3.4 Result Configuration Parameters

`--outputTransform <filename>`	Filename to which save the estimated transform.
`--strippedOutputTransform <filename>`	estimated transform, stripped of scaling, to register the moving image to the fixed image.
`--patientID <patient ID>`	Identifier for the research subject. (default: ANON)
`--studyID <study name>`	Identifier for the scanner encounter (MRQID). (default: ANON)
`--outputVolume <filename>`	The (optional) output image for registration.
`--outputVolumePixelType <pixel type>`	The output image Pixel Type is the scalar datatype for representation of the Output Volume. One of `float, short, ushort, int, uint, char, uchar` (default: `float`)
`--backgroundFillValue <double>`	Background fill value for output image. (default: 0.0)
`--scaleOutputValues`	If true, and the voxel values do not fit within the minimum and maximum values of the desired outputVolumePixelType, then linearly scale the min/max output image voxel values to fit within the min/max range of the outputVolumePixelType. (default: false)
`--useWindowedSinc`	Use windowedSinc interpolation to create output images. WARNING: This will add 8 minutes to the interpolation of the final image of size 256x256x256. (default: false)

3.5 Control of Masked Processing

`--maskProcessingMode <mode name>`	What mode to use for using the masks. If ROIAUTO is choosen, then the mask is implicitly defined using a otsu forground and hole filling algorithm. Where the Region Of Interest mode uses the masks to define what parts of the image should be used for computing the transform. One of `NOMASK, ROIAUTO, ROI` (default: `NOMASK`)
`--fixedBinaryVolume <filename>`	Fixed Image binary mask volume.
`--movingBinaryVolume <filename`	Moving Image binary mask volume.

3.6 Special Input Image Parameters

`--fixedVolumeTimeIndex <int>`	The index in the time series for the 3D fixed image to fit, if 4-dimensional. (default: 0)
`--movingVolumeTimeIndex <int>`	The index in the time series for the 3D moving image to fit, if 4-dimensional. (default: 0)
`--fixedVolumeOrigin <x,y,z>`	The coordinates of the origin of the fixed image. (default: 0,0,0)
`--movingVolumeOrigin <x,y,z>`	The coordinates of the origin of the moving image. (default: 0,0,0)
`--medianFilterSize <x,y,z>`	The radius for the optional MedianImageFilter preprocessing in all 3 directionis. (default: 0,0,0)

3.7 Registration Debugging Parameters

`--relaxationFactor <double>`	Internal debugging parameter (default: 0.5)
`--maximumStepSize <double>`	Internal debugging parameter (default: 0.2)

3.8 System Debugging Parameters

`--failureExitCode <int>`	If fit fails exit with this status code rather than 0. (default: -1)
`--writeTransformOnFailure`	Flag to save the final transform even if the numberOfIterations are reached without convergence. (default: 0)
`--debugNumberOfThreads <int>`	Explicitly specify the maximum number of execution threads to use.

3.9 GenerateCLP Global Options

`--processinformationaddress <int>`	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

BRAINSFit depends on the Insight Toolkt, version 3.4 or later, and CMake 2.4 or later. See the build instructions below; BRAINSFit is distributed with build scripts that can automatically retrieve the source code for both ITK and CMake and build them as part of the BRAINSFit build process.

You can also install ITK and CMake separately and use the usual CMake configure-build-install process if you are comfortable doing that.

The build script is written in the TCL scripting language, so Tcl needs to be installed to run that script. Ready-to-install TCL binaries are available from thtp://www.activestate.com/activetcl or http://www.equi4.com/pub/tk/downloads.html.

5 Building BRAINSFit

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

- make a 'sandbox' directory in which to build BRAINSFit.
- change to that sandbox directory
- Untar (or checkout from version control) the BRAINSFit distribution.
- run BRAINSFit/BuildScripts/getbuildtest.tcl.
- Optionally, install the program

The following shell script accomplishes all these steps, except installation:

```
# 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 BRAINSFit
#
# run the build script.
tclsh BRAINSFit/BuildScripts/getbuildtest.tcl
```

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