Hierarchical Attribute Matching Mechanism for Elastic Registration (HAMMER)

Release 0.03

Guorong Wu[1], Xiaodong Tao[2], James V. Miller[2], and Dinggang Shen[1]

November 05, 2011

[1]Department of Radiology and BRIC, University of North Carolina at Chapel Hill, U.S.A.
[2]Visualization and Computer Vision Laboratory, GE Research, U.S.A.

Abstract

This article provides details on Hierarchical Attribute Matching Mechanism for Elastic Registration (HAMMER) – a deformable MR brain registration algorithm that is now also implemented as a 3D Slicer extension in an Insight Toolkit (ITK) framework. HAMMER has been applied in a large number of studies involving over 10,000 brains. Our implementation uses the framework from ITK. Specifically, we implemented a new ITK image-to-image filter, called "HammerDeformableRegistrationImageFilter", providing a general feature-based registration framework, which is extensible to incorporate additional image attributes for more precise registration.

Contents

1 Introduction

Image registration is a critical prerequisite for many neuroimaging studies. The main objective of image registration is to determine the spatial correspondences between images. Achieving this allows the removal of confounding morphological differences between images for more precise statistical analysis (i.e., voxel-based morphometry, deformable-based morphometry, etc.). There are typically two types of registration algorithms: affine and deformable. Affine registration determines the relative translation, rotation, scaling, and shearing between images. Deformable registration determines a high-dimensional mapping between images.

HAMMER [1, 2] is a deformable registration algorithm that leverages multi-scale attribute matching. The attribute vector is a morphological signature that encapsulates information such as image intensity, tissue type, and multi-scale regional geometric moment invariants (GMIs) [3] of WM, GM, VN, and CSF segmentations. To avoid matching ambiguity, voxels with the most distinctive attribute vectors are selected to drive the registration. The number of driving voxels is progressively increased to refine registration accuracy. Forward-backward matching is employed for ensuring inverse consistency. Matching was performed in hierarchical multi-scale fashion for speed improvement and for avoiding local minima.

2 Overview of the Algorithm

The goal HAMMER is to determine a transformation $\{h(x) = x + u(x), x \in \Omega_T, y \in \Omega_S\}$ that best matches the attribute vectors $a_T(x)$ at location x in the domain Ω_T of a fixed template T and $a_S(y)$ at location y in the domain Ω_S of a moving subject S. Function $h(x)$ displaces x to y. Instead of determining the correspondence for every voxel, HAMMER hierarchically selects a set of most distinguishable voxels, called driving voxels, to guide the deformation of the less distinctive voxels. As shown in Fig. 1, the driving voxels (displayed in red) are typically the located salient points, such as points at sulcal roots, gyral crowns, and ventricular boundaries, and can hence be depended on for reliable structural matching. As shown in the bottom left panel of Fig 1, these driving voxels initially give a coarse estimation of the deformation. With progressive addition of driving voxels (shown in green and yellow), deformation of increasing complexity can be estimated.

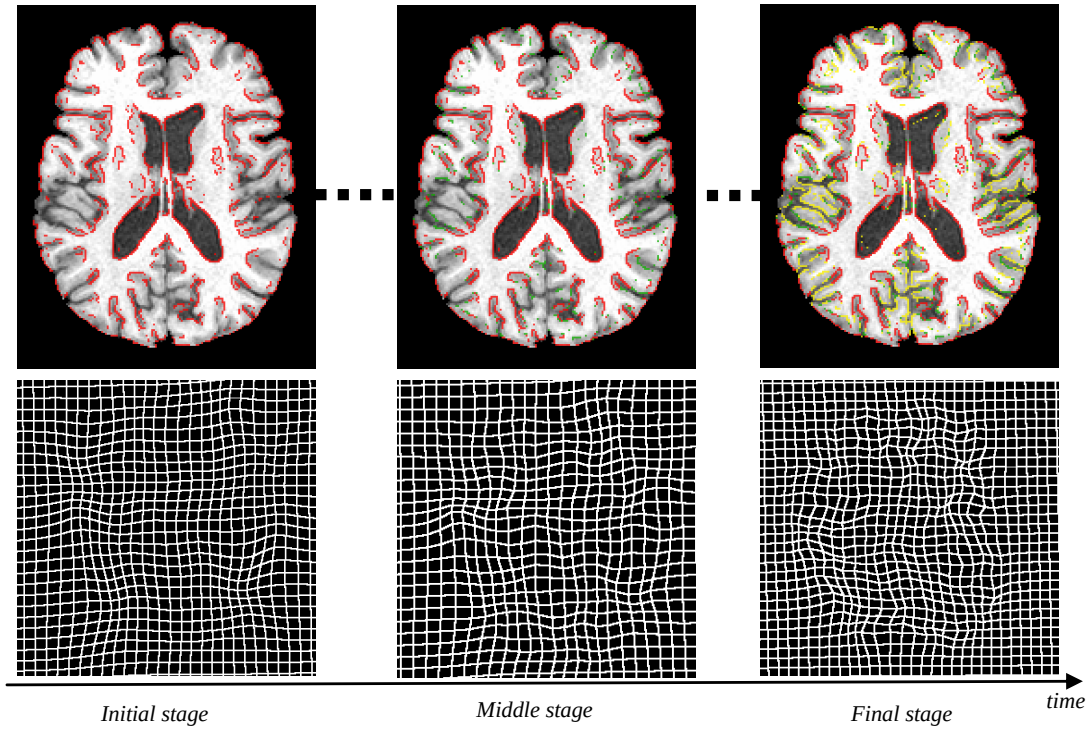

Fig. 1. Driving voxels and deformation field estimation at different stages of registration. In the initial stage, only a small number of voxels with distinctive attribute vectors (displayed in red), which are usually located at sulcal roots, gyral crowns, and ventricular boundaries, are selected to steer the registration and guide deformation of less distinctive voxels. With the progress of registration, more and more driving voxels are gradually added (shown in green and yellow). The bottom row shows the deformation fields estimated at three different stages. It can be observed that, as the number of driving voxel increases, the deformation field is progressively refined.

3 Implementation Notes

The implementation follows the ITK framework and the filters are templated over pixel type and deformation type. The implementation is modular, reusable, and extensible.

There are two image filters that are derived from `itk::ImageToImageFilter`: "`HammerTissueAttributeVectorImageFilter`", and "`HammerDeformableRegistrationImageFilter`".

"`HammerTissueAttributeVectorImageFilter`" calculates the attribute vectors used in HAMMER. The input image is required to be segmented to White Matter (WM), Gray Matter (GM), and Cerebrospinal Fluid (CSF), which are labeled to 250, 150, and 10, respectively. The neighborhood size used to compute the geometric moment invariant (GMI) features needs to be specified in

"HammerDeformableRegistrationImageFilter". The output is a vector image as defined in "`HammerTissueAttributeVector.h`".

"`HammerDeformableRegistrationImageFilter`" encapsulates the core registration algorithm as described in Section 2. By default, HAMMER will run in three resolutions: low, middle, and high resolution. `SetNumberOfIterations` sets the number of iterations for each resolution (default: 50). `SetSearchRadius` sets the radius of the initial search neighborhood (default: 12, 10, and 8 in low, middle, and high resolution, respectively). `SetSubvolumnSimilarityThreshold` sets the threshold of subvolume similarities (default: 0.6). `SetDeformRate` sets the allowed percentage of deformation in each iteration (default: 0.05). `SetDeformationFieldSigma` sets the smoothness of the deformation field (default: 1.0).

In brief, the filter performs the following:

1) Calculate the GMIs for both template and subject images;
2) Select the driving voxels X_T and Y_S for template and subject images, respectively;
3) Determine the correspondence of each driving voxel;
4) Calculate the global transformation matrix;
5) Obtain the dense deformation h by Gaussian propagation;
6) Smooth the deformation field h.
7) If not converged, relax the criteria of driving voxel selection (i.e., relax the threshold on the zero-order GMIs) and go to step 2.

Below is the list of member function implemented for this filter:

void CreatePointMatchingNeighbor(IndexArrayType &Neighbor, int Radius):
Sets the neighborhood size for subvolume matching of GMI features.

void CreateSubvolumnNeighbor(IndexArrayType &Neighbor, int Radius):
Sets the neighborhood size for warping subvolume according to tentative displacement.

void CreateSearchNeighbor(IndexArrayType &Neighbor, int Radius):
Calculates the offset of searching neighborhood.

void CalculateNeighborhoodbyIncreasingRadius(IndexArrayType &Neighbor, int Radius) throw (InvalidRequestedRegionError):
Records the location of each neighborhood point.

float SimilarityBetweenTwoImgAttribute(AttributeVectorType Template_Feature, AttributeVectorType Subject_Feature) const:
Computes the similarity two attribute vectors.

float DetermineCorrespondenceOnOneDrivingVoxel(ImageAttributePointerType &FixedAttributeImage, ImageAttributePointerType &MovingAttributeImage, DeformationFieldPointer &DeformFld, int DrivingPointIndex, DeformationFieldPointer DeformFld_Last, DeformationVectorType &DeformationUpdate, int SearchRadius, int Step) const:
Core algorithm of HAMMER. Outputs the estimated displacement vector and the subvolume matching degree based on subvolume matching. The input is the index m_ModelDrivingPoint.

float SubVolumeMatching(ImageAttributePointerType &FixedAttributeImage, ImageAttributePointerType &MovingAttributeImage, DeformationFieldPointer &DeformFld, IndexType &ImageIndex, DeformationVectorType TentativeWarp, IndexArrayType CertainNeighborhood, int NeighborhoodStep, float *MinDist, float MinDist_Threshold) const:
Performs subvolume matching. Returns the overall degree of similarity.

float ComputeVectorMagnitude(DeformationVectorType Deform_Vector) const:
Computes the norm of a vector.

void FindingInverseForceFromSubject(ImageAttributePointerType &FixedAttributeImage, ImageAttributePointerType &MovingAttributeImage, DeformationFieldPointer &DeformFld, int SearchRadius) const:
Computes the inverse force from subject image to template image.

void DisseminateDeformation(DeformationFieldPointer &DeformFld, const int &DrivingPointIndex, DeformationVectorType TentativeWarp, IndexArrayType CertainNeighborhood, int NeighborhoodSize, int GaussianSigma):
Warps the subvolume w.r.t. tentative displacement vector using Gaussian weighting.

void IdentifyDrivingVoxelsInFixedImage(ImageAttributePointerType &FixedAttributeImage, std::vector<float> &FixedImageCriteria):
Determines driving voxels with given criterion in the fixed image.

void IdentifyDrivingVoxelsInMovingImage(ImageAttributePointerType &MovingAttributeImage, std::vector<float> &MovingImageCriteria):
Determines the driving voxels with the given criterion in the moving image.

unsigned int IdentifyDrivingVoxels(ImageAttributePointerType avPointer, std::vector<IndexType> &drivingVoxels, std::vector<float> &Criteria):
Determines whether a voxel is qualified as a driving voxel.

void SmoothDeformationField(ImageAttributePointerType &FixedAttributeImage, DeformationFieldPointer DeformFld, DeformationFieldPointer DeformFld_Last, float LocalRatio):
Gaussian smoothing of the deformation field.

void FitGlobalAffineTransform(DeformationVectorArrayType CurrentDeformationOnDrivingPoint, DeformationVectorArrayType PreviousDeformationOnDrivingPoint):
Estimate the affine transformation matrix from the correspondences of driving voxels by least square fitting.

void SetStandardDeviations(double value):
Sets the standard deviations for Gaussian smoothing

void SmoothDeformation_OneTime(ImageAttributePointerType &FixedAttributeImage, DeformationFieldPointer DeformFld, int Time):

Smooths the deformation field.

void EdgePreserveSmoothDeformation_OneTime(ImageAttributePointerType &FixedAttributeImage, DeformationFieldPointer DeformFld, int Time):
Smooths of the deformation field with edge preserving.

void HAMMERRegistrationOneRound(ImageAttributePointerType &FixedAttributeImage, ImageAttributePointerType &MovingAttributeImage, int resolution, float ratio_iteration, float LocalRatio, DeformationFieldPointer DeformFld):
Performs one round of HAMMER registration

void HAMMERMainLoop(ImageAttributePointerType &FixedAttributeImage, ImageAttributePointerType &MovingAttributeImage, DeformationFieldPointer &DeformFld, itk::ProgressReporter & progress):
The core of the HAMMER algorithm.

4 User Guide

4.1 Software Required to Compile HammerDeformableRegistrationFilter

- Insight Toolkit version 2.4 or higher
- CMake version 2.2 or higher

4.2 Using the HAMMER Registration Filter

4.2.1 Preprocessing

Remove extra-cranial tissue and segment brain tissue into white matter, gray matter, and cerebrospinal fluid. Set the label values for white matter, gray matter and cerebrospinal fluid to 250, 150, and 10, respectively.

4.2.2 Using the filter

Here we demonstrate how to use HAMMER registration filter for deformable registration of the two T1 MR brains. The source code can be found in /src/test/HammerRegistrationTest.cxx.

First of all, we need to initialize the HAMMER registration filter instance as follows:

```
typedef itk::HammerDeformableRegistrationImageFilter<
            ImageType,
            DeformationFieldType> RegistrationFilterType;
      RegistrationFilterType::Pointer hammer = RegistrationFilterType::New();
```

Then, the fixed subject and the moving subject should be specified:

```
      hammer->SetFixedImage( fImg0 );
      hammer->SetMovingImage( mImg0 );
```

Next, we need to specify the parameters used in HAMMER

```
hammer->SetIterations( iterations[0], iterations[1], iterations[2] );
hammer->SetDeformRate(0.05);
hammer->SetPointMatchingThreshold(0.8);
hammer->SetSubvolumeSimilarityThreshold(0.6);
hammer->SetSearchRadius(12);
```

Finally, calling `hammer->Update()` will return a dense deformation field that points from the fixed image to the moving image. The following warps the subject image with itk::WarpImageFilter.

```
typedef itk::WarpImageFilter<
          ImageType,
          ImageType,
          DeformationFieldType  >     WarperType;

typedef itk::NearestNeighborInterpolateImageFunction<
          ImageType,
          double >  InterpolatorType;

WarperType::Pointer warper = WarperType::New();
    InterpolatorType::Pointer interpolator = InterpolatorType::New();

warper->SetInput( mImg0 );
warper->SetInterpolator( interpolator );
warper->SetOutputDirection( fImg0->GetDirection() );
warper->SetOutputSpacing( fImg0->GetSpacing() );
warper->SetOutputOrigin( fImg0->GetOrigin() );
warper->SetDeformationField(hammer->GetOutput());

WriterType::Pointer      writer =  WriterType::New();
writer->SetFileName( resampledFilename.c_str() );
writer->SetInput( warper->GetOutput() );

writer->Update();
```

4.3 Using HAMMER Registration Filter in Slicer 3

We have also integrated HAMMER into 3D Slicer. Fig. 2 shows the GUI of HAMMER. A step-by-step tutorial of how to use HAMMER can be found on our NITRC website (http://www.nitrc.org/projects/hammerwml).

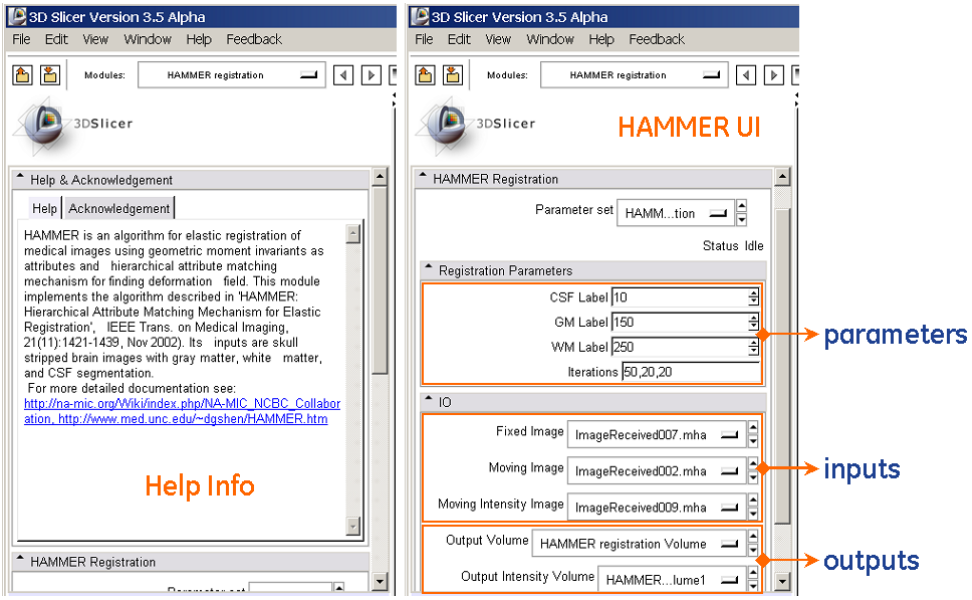

Fig. 2. The GUI of HAMMER in 3D Slicer.

5 Conclusion

We have implemented HAMMER in ITK framework. To our knowledge, it is the first hierarchical feature-based registration tool in the ITK. The algorithm has been extensively tested on different platforms and can be easily integrated and adapted for different applications.

Reference

[1] D. Shen and C. Davatzikos, "HAMMER: Hierarchical attribute matching mechanism for elastic registration," *IEEE Transactions on Medical Imaging*, vol. 21, pp. 1421-1439, November 2002.

[2] D. Shen and C. Davatzikos, "Very high resolution morphometry using mass-preserving deformations and HAMMER elastic registration," *NeuroImage*, vol. 18, pp. 28-41, January 2003.

[3] C. H. Lo and H. S. Don, "3-D Moment Forms: Their Construction and Application to Object Identification and Positioning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, pp. 1053-1064, October 1989 1989.