Non-rigid Groupwise Registration using B-Spline Deformation Model

Release 0.00

Serdar K. Balci[1], Polina Golland[1] and William M. Wells[2]

July 17, 2007

[1]MIT CSAIL, Cambridge, MA, USA
[2]Brigham & Women's Hospital, Harvard Medical School, Cambridge, MA, USA

Abstract

In this work, we extend a previously demonstrated entropy based groupwise registration method to include a non-rigid deformation model based on B-splines. We describe an open source implementation of the groupwise registration algorithm using the Insight Toolkit ITK www.itk.org. We provide the source code, parameters, input and output data that we used for validation.

We describe an efficient implementation of the algorithm by using a stochastic optimization scheme embedded in a multi-resolution setting. The objective function is optimized using gradient descent algorithm combined with line search for the step size. The derivative of the objective function is evaluated efficiently by computing Jacobian of B-spline deformation field locally.

We demonstrate the algorithm in application to different imaging modalities including proton density, FA, T1 and T2 MR images. We validate the algorithm on synthetic datasets varying from 2 to 30 images by recovering randomly applied affine and B-spline transforms.

Contents

1 Introduction

Groupwise registration is an important tool in medical image analysis for establishing anatomical correspondences among subjects in a population [10, 2]. A groupwise registration scheme can be used to characterize anatomical shape differences within and across populations [9].

Miller et al. [7] introduce an efficient groupwise registration method in which they consider sum of univariate entropies along pixel stacks as a joint alignment criterion. They provide a template-free approach to groupwise registration by simultaneously driving all subjects to the common tendency of population. Zollei et al. [11] successfully applied this method to groupwise registration of medical images using affine transforms and used stochastic gradient descent algorithm for optimization.

In our work, we extend Miller et al.'s [7] method to include free-form deformations based on B-splines in 3D. We describe an efficient implementation using stochastic optimization in a multi-resolution setting. To optimize the objective function we use gradient descent algorithm combined with line search for the step size. We validate the algorithm on synthetic datasets by recovering randomly applied affine and B-spline transforms. All experiments that we present are easily reproducable as we provide our source code, parameters, input data and an automated procedure to obtain the results.

This paper is organized as follows. In the next section, we describe the groupwise registration algorithm by describing the stack entropy cost function and B-spline based deformation model. In Section 3, we present our registration framework and describe implementation of the metric, deformation model and optimization. In Section 4, we demonstrate the groupwise registration algorithm in application to different sets of input data. In Section 5, we describe an automated procedure to reproduce the test results. In Appendix A, we give all test results and in Appendix B we go over a groupwise registration example. In Appendix C we discuss important registration parameters.

2 Algorithm

Given a set of images $\{I_1, \ldots, I_N\}$, each described by intensity values $I_n(\mathbf{x}_n)$ across the image space $\mathbf{x}_n \in \mathcal{X}_n$, we can define a common reference frame $\mathbf{x}_R \in \mathcal{X}_R$ and a set of transforms $\mathcal{T}$ which maps points from the reference frame to points in the image space

$$\mathcal{T} = \{T_n : \mathbf{x}_n = T_n(\mathbf{x}_R), n = 1, \ldots, N\} \tag{1}$$

In the following section, we describe the stack entropy cost function as introduced by Miller et al. [7] and applied to groupwise registration of medical images using affine transforms by Zollei et al. [11]. We

continue by describing the transformation model and extend Miller et al.'s [7] method to include free-form deformations based on B-splines.

2.1 Stack Entropy Cost Function

In order to align all subjects in the population, we consider sum of univariate entropies as a joint registration criterion. We let $x_v \in \mathcal{X}_R$ be a sample from a spatial location in the reference frame and $H(I(T(x_v)))$ be the univariate entropy of the stack of pixel intensities $\{I_1(T_1(x_v)),\ldots,I_N(T_N(x_v))\}$ at spatial location x_v. The objective function for the groupwise registration can be given as follows

$$f = \sum_{v=1}^{V} H(I(T(x_v))). \tag{2}$$

We employ a Parzen window based density estimation scheme to estimate univariate entropies [3]:

$$f = -\sum_{v=1}^{V} \frac{1}{N} \sum_{i=1}^{N} \log \frac{1}{N} \sum_{j=1}^{N} G_\sigma(d_{ij}(x_v)) \tag{3}$$

where $d_{ij}(x) = I_i(T_i(x)) - I_j(T_j(x))$ is the distance between intensity values of a pair of images evaluated at a point in the reference frame and G_σ is a Gaussian kernel with variance σ^2. Parzen window density estimator allows us to obtain analytical expressions for the derivative of the objective function with respect to the transformation parameters

$$\frac{\partial f}{\partial T_n} = \sum_{v=1}^{V} \frac{1}{\sigma^2 N} \sum_{i=1}^{N} \sum_{j=1}^{N} \frac{G_\sigma(d_{ij}(x_v)) d_{ij}(x_v)}{\sum_{k=1}^{N} G_\sigma(d_{ik}(x_v))} \frac{\partial d_{ij}(x_v)}{\partial T_n}. \tag{4}$$

2.2 Free-Form B-spline Deformation Model

For the nonrigid deformation model, we define a combined transformation consisting of a global and a local component

$$T(\mathbf{x}) = T_{local}(T_{global}(\mathbf{x})) \tag{5}$$

where T_{global} is a twelve parameter affine transform and T_{local} is a deformation model based on B-splines.

Following Rueckert et al.'s formulation [8], we denote $\Phi_{i,j,k}$ an $n_x \times n_y \times n_z$ grid of control points with uniform spacing. The free form deformation can be written as a 3-D tensor product of 1-D cubic B-splines.

$$T_{local}(\mathbf{x}) = \mathbf{x} + \sum_{l=0}^{3} \sum_{m=0}^{3} \sum_{n=0}^{3} B_l(u) B_m(v) B_n(w) \Phi_{i+l,j+m,k+n} \tag{6}$$

where $\mathbf{x} = (x,y,z)$, $i = \lfloor x/n_x \rfloor - 1$, $j = \lfloor y/n_y \rfloor - 1$, $k = \lfloor z/n_z \rfloor - 1$, $u = x/n_x - \lfloor x/n_x \rfloor$, $v = y/n_y - \lfloor y/n_y \rfloor$, $w = z/n_z - \lfloor z/n_z \rfloor$ and where B_l is l'th cubic B-spline basis function. Using the same expressions for u,v and w as above, the derivative of the deformation field with respect to B-spline coefficients can be given by

$$\frac{\partial T_{local}(x,y,z)}{\partial \Phi_{i,j,k}} = B_l(u) B_m(v) B_n(w) \tag{7}$$

where $l = i - \lfloor x/n_x \rfloor + 1$, $m = j - \lfloor y/n_y \rfloor + 1$ and $n = k - \lfloor z/n_z \rfloor + 1$. We consider $B_l(u) = 0$ for $l < 0$ and $l > 3$. The derivative terms are nonzero only in the neighborhood of a given point. Therefore, optimization of the objective function using gradient descent can be implemented efficiently.

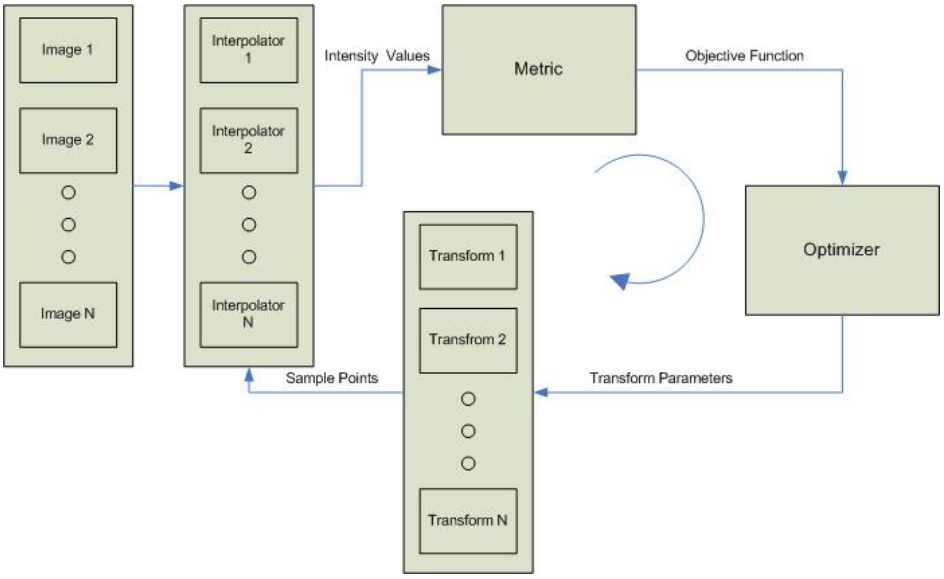

Figure 1: The basic components of the registration framework are a metric, an optimizer and arrays of input images, transforms and interpolators.

As none of the images are chosen as an anatomical reference, it is necessary to add a geometric constraint to define the reference coordinate frame. Similar to Bhatia et al. [1], we define the reference frame by constraining the average deformation to be the identity transform:

$$\frac{1}{N}\sum_{n=1}^{N} T_n(\mathbf{x}) = \mathbf{x} \tag{8}$$

This constraint assures that the reference frame lies in the center of the population. In the case of B-splines, the constraint can be satisfied by constraining the sum of B-spline coefficients across images to be zero. In the gradient descent optimization scheme, the constraint can be forced by subtracting the mean from each update vector [1].

3 Implementation

Figure 1 displays the basic components of a groupwise registration framework and the interactions between them. The framework is an extension of ITK's pairwise registration framework described in ITK software guide [4].

The basic input to the registration process is an array of images. In our setting all images are considered to be moving images. None of the images are chosen as a fixed image to avoid anatomical bias to the chosen reference. Instead, the reference frame is defined by constraining the sum of all transforms to be the identity transform as shown in equation 8. The constraint is forced inside the metric classes by subtracting the mean from transform parameter updates.

Every image is associated with an interpolator and a transform. Each transform map points from the common coordinate frame to the corresponding space of the input image. Interpolators allow to evaluate intensity values at non-grid locations. The metric computes an objective function which measures how well the

group of images are registered. The optimizer component optimizes this objective function by searching over the parameters of the transforms.

In following sections, we introduce registration components and the interactions between them.

3.1 Metric

We extended `itk::ImageToImageMetric` base class to `itk::MultiImageMetric` to compute an objective function on a group of images. The base class is multi-threaded and provides generic methods to be used by the optimizer.

`itk::UnivariateEntropyMultiImageMetric` derive from this class and compute the sum of univariate entropies as given in equation 3. This metric uses a stochastic subsampling scheme to efficiently evaluate the objective function [6]. The function `SampleFixedImageDomain` takes random samples from the image domain in a multi-threaded fashion. In each iteration of the registration process, random samples are taken and the objective function is evaluated only on this sample set. The metric makes use of the B-spline optimization for Jacobian computations if a B-spline transform is connected to it.

`itk::VarianceMultiImageMetric` computes sum of variances along pixel stacks. This class matches each image to a mean template image using sum of squared errors as an objective function similar to Joshi et al. [5]. This class also follows a stochastic subsampling procedure and performs multi-threaded execution.

3.2 Optimization

We provide an efficient optimization scheme by using line search with the gradient descent algorithm. For each iteration, the class `itk::GradientDescentLineSearchOptimizer` performs a line search to determine the step size of the gradient descent algorithm. Empirically we observed that this optimization method is more robust to optimization parameters and increases the convergence rate.

As in every iterative search algorithm, local minima pose a significant problem. To avoid local minima we use a multi-resolution optimization scheme. The registration is first performed at a coarse scale by downsampling the input. Results from coarser scales are used to initialize optimization at finer scales.

Efficient B-spline Implementation

To compute B-spline deformation fields efficiently we modified `itk::BSplineDeformableTransform` to take into account the locality of B-splines for Jacobian computations. The new class `itk::BSplineDeformableTransformOpt` computes the Jacobian field locally by using the function `GetJacobian( point, indexes, weights )`. This function returns the nonzero `indexes` of the Jacobian field and the `weights` associated with them. The metric class can make use of this optimization in `GetValueAndDerivative()` method with an `if` statement for B-splines. `itk::MultiImageMetric` has a member `bool m_BsplineDefined`, which is turned on if the metric class is used with `itk::BSplineDeformableTransformOpt`. As only a fixed number of control points in the Jacobian field are nonzero, the computational gain is significant, especially if the deformation field is dense. The changes we made to `itk::BSplineDeformableTransform` are backward compatible as we keep the default implementation of the member function `GetJacobian( point)`.

3.3 Component Interaction

Interconnections between the components should be handled before starting the registration. We modified the pairwise registration method `itk::MultiResolutionImageRegistrationMethod` to create a groupwise registration method `itk::MultiResolutionMultiImageRegistrationMethod` that automatically initializes connections between registration components and provides a multiresolution optimization scheme. The Registration method has several functions to describe a multiresolution groupwise registration setting.

- `SetNumberOfImages()`: Sets number of images used in registration
- `SetNumberOfLevels()`: Sets number of multiresolution levels
- `SetOptimizer()`: Sets the optimizer
- `SetMetric()`: Sets the registration metric
- `SetImagePyramidArray(int, imagePyramid)`: Uses the given image pyramid in the registration
- `SetTransformArray(int,transform)`: Connects the given transform to corresponding images
- `SetInterpolatorArray( int,interpolator)`: Connects the given interpolator to corresponding images
- `SetTransformParametersLength()`: Allocates space for transform parameters
- `StartRegistration()`: Connects the components and starts the registration

Using the functions above the registration method initializes registration components, handles the interconnections between them and performs a multi-resolution optimization. Appendix B provides a registration example using this class.

3.4 Summary of New Classes

We provide the following classes as part of the groupwise registration framework.

- `itk::MultiImageMetric<TImage>`
  A multi-threaded base class templated over the input image for groupwise registration metrics.
  - `itk::UnivariateEntropyMultiImageMetric<TImage>`
    Computes sum of univariate entropies along pixel stacks.
  - `itk::VarianceMultiImageMetric<TImage>`
    Computes sum of variances along pixel stacks.
- `itk::MultiResolutionMultiImageRegistrationMethod<TImage>`
  Provides a generic interface for multi-resolution registration using components of the registration framework.
- `itk::GradientDescentLineSearchOptimizer`
  Gradient descent optimizer combined with line search for determining the step size.
- `itk::BSplineDeformableTransformOpt<TScalarType, NDimensions, VSplineOrder >`
  B-spline deformable transform optimized for Jacobian computations.

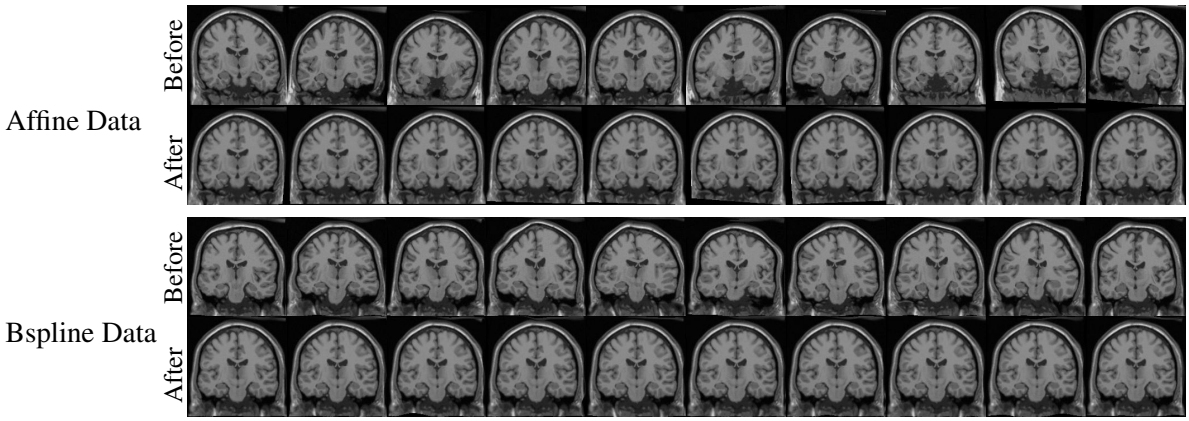

Figure 2: Central slices of synthetic affine and B-spline 3D volumes before and after registration.

4 Results

To validate the algorithm we run synthetic experiments using four different imaging modalities: proton density data with $181 \times 217 \times 180$ voxels and $1 \times 1 \times 1$ mm spacing, T1 data with $181 \times 217 \times 180$ voxels and $1 \times 1 \times 1$ mm spacing, T2 data with $181 \times 217 \times 180$ voxels and $1 \times 1 \times 1$ mm spacing, FA data with $256 \times 256 \times 50$ voxels and $0.9375 \times 0.9375 \times 2.5$ mm spacing. The input data can be obtained from ITK's Data directory at http://public.kitware.com/pub/itk/Data/.

For each sample medical data we perform two different sets of experiments. First, we apply random affine transforms to input data and register this synthetic dataset using global affine registration. In the second setting, we apply random B-spline transforms to input data and recover applied transforms using B-spline registration. Figure 2 show the central slices of 10 random 3D images from the T1 MR image before and after registration. The figure indicate that the images are well aligned after registration.

To evaluate registration accuracy visually, we compute mean and standard deviation images before and after registration. Figure 3 show the central slices of mean and standard deviation images for all four modalities using an input data of 10 images. Visually we can observe that the mean images get sharper and the standard deviation images get darker. In a perfect registration, the standard deviation images should be all zero images.

From the images in figure 3, we can note some registration artifacts at image boundaries. These occur because during random image generation images get cropped at boundaries. For datasets generated with affine transforms, standard deviation images get close zero after affine registration. Therefore, we can state that for all imaging modalities transform components are recovered successfully.

For datasets generated with B-spline transforms, we note some residuals at central locations in standard deviation images after non-rigid registration. This occur because we used the same transform complexity with $8 \times 8 \times 8$ B-spline control points for both image generation and registration. The residuals can be made smaller by using B-splines with higher number of control points, as a dense deformation field can capture larger shape variations. The main reason for the residuals is that the inverse of a B-spline transform is generally not a B-spline transform and the registration results show the best fit to the inverse transforms.

To examine the performance of the algorithm with respect to number of input images we run experiments with 2, 10 and 30 images. For the results of these experiments see Appendix A. The results in Appendix A suggest that the groupwise registration algorithm is robust to varying number of input images. Visually, standard deviation values are close to zero which indicates successful recovery of applied transforms.

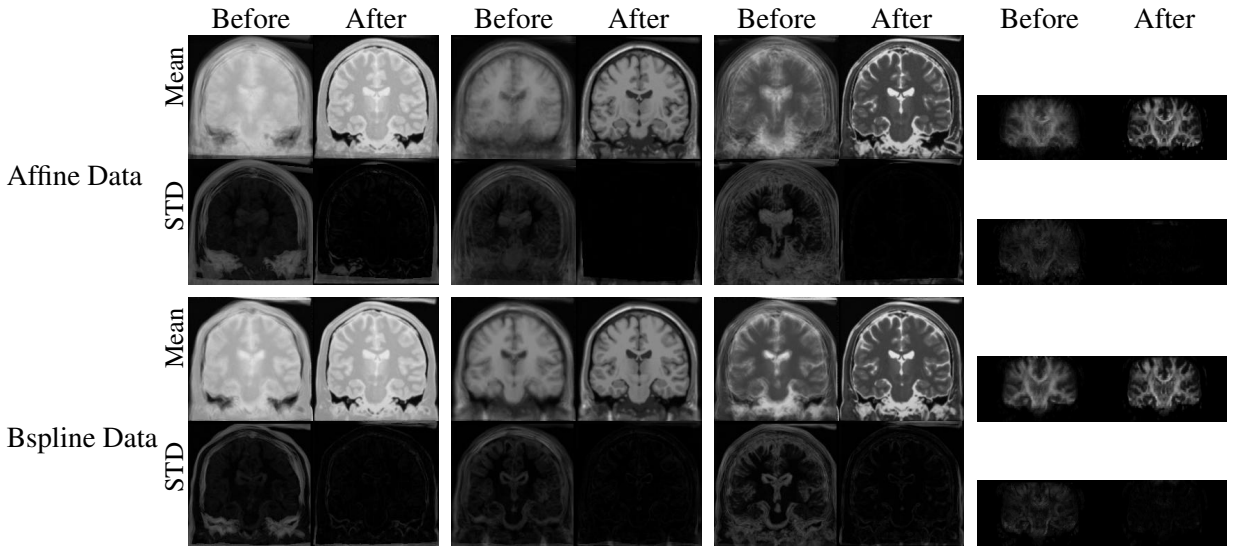

Figure 3: Central slices of 3D volumes before and after registration for (from left to right) PD, T1, T2 and FA synthetic images. Inside a block top row shows the mean images and bottom row standard deviation images before and after registration. Blocks in the top row show the results for synthetic data generated with 10 random affine transforms and blocks in the bottom row show results for synthetic data generated with 10 random B-spline transforms.

The experiments show that the groupwise registration algorithm can handle few input images including a minimum of two input images.

5 Running Tests

To reproduce the results presented in this paper please follow the directions below:

1. Compile ITK 2.8 or higher (http://www.itk.org).
2. Install CMake 2.4 or higher (http://www.cmake.org).
3. Download Groupwise Registration Project from Insight Journal (http://insight-journal.org).
4. Configure and compile Groupwise Registration Project using CMake. Point the folder containing brain MR images to Brainweb_DATA_ROOT and the folder containing FA images to FAImage_Data_ROOT. To avoid timeout errors set DART_TESTING_TIMEOUT parameter to a large value(e.g. 36000).
5. To reproduce the results presented in this document, run CTest from the project folder.
6. Check test results from the corresponding folders in Testing/Temporary. Examine registration parameters and the example code `GroupwiseRegistrationExample.cxx`.

6 Conclusion

In this paper, we describe an open source implementation of a non-rigid groupwise registration algorithm using the Insight Toolkit ITK `www.itk.org`. We provide the source code, parameters, input and output data that we used for validation.

We demonstrated the algorithm in application to different imaging modalities including proton density, FA, T1 and T2 MR images. We validated the algorithm on synthetic datasets varying from 2 to 30 images by recovering randomly applied affine and B-spline transforms.

Appendix A show the results for all experiments Appendix B present a groupwise registration example using the implemented classes and Appendix C explain how to use command line modules.

A Test Results

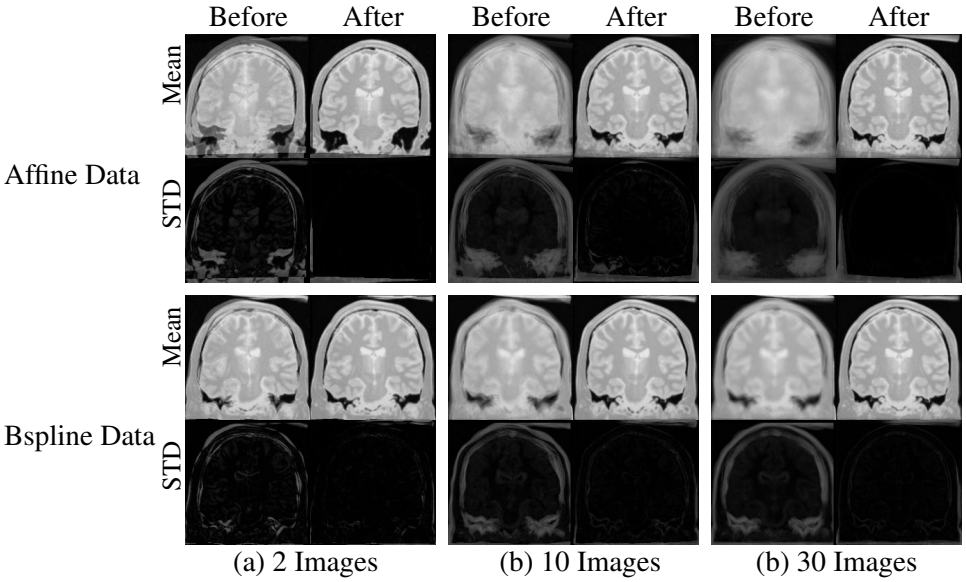

Figure 4: Central slices of 3D volumes before and after registration for proton density images with varying number of input images.

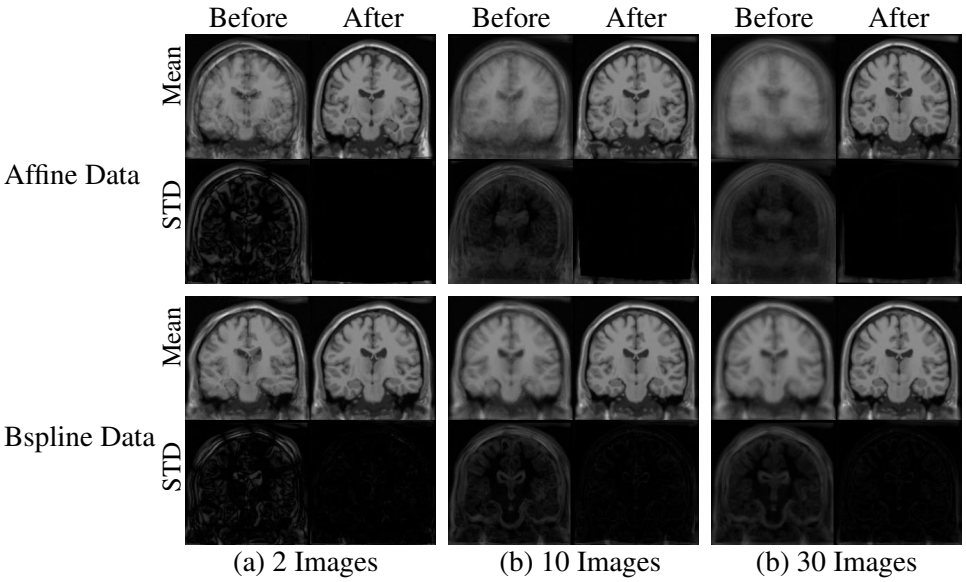

Figure 5: Central slices of 3D volumes before and after registration for T1 MR images with varying number of input images.

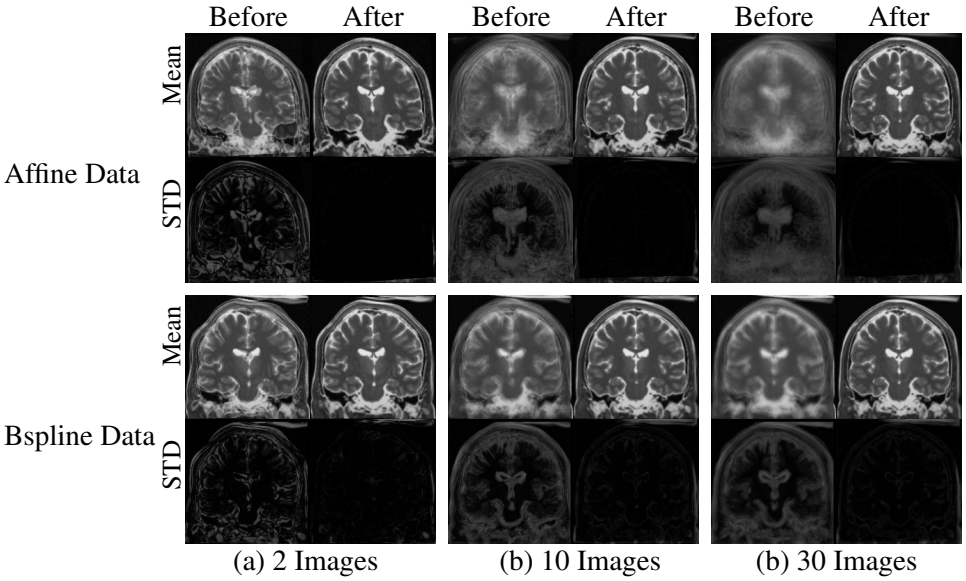

Figure 6: Central slices of 3D volumes before and after registration for T2 MR images with varying number of input images.

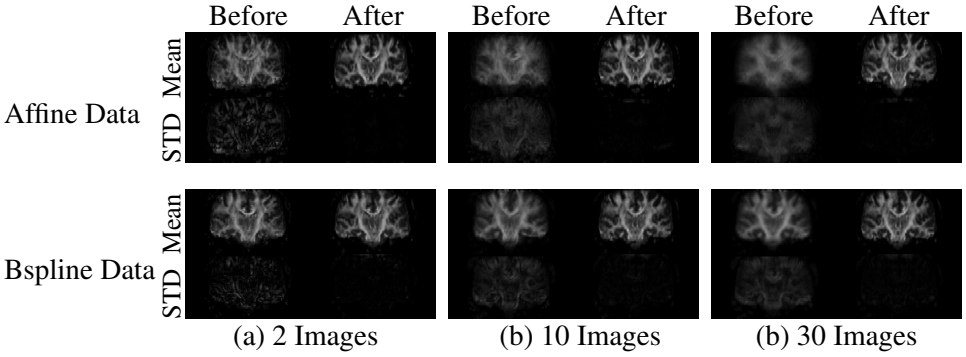

Figure 7: Central slices of 3D volumes before and after registration for FA images with varying number of input images.

B Registration Example

The source code for this example can be found in `Source/GroupwiseRegistrationExample.cxx`

In this example we show how to use the groupwise registration framework. The example code performs groupwise registration on an input data and outputs the resulting transform parameters. The resulting transform parameters can be used to transform input images, calculate mean and standard deviation images. As the size of the example code is relatively large to explain all in detail here, we will briefly mention main parts and show how to set up main components.

We first include project specific headers and define the pixel types using `#define` statements.

```
#include "MultiResolutionMultiImageRegistrationMethod.h"
#include "VarianceMultiImageMetric.h"
#include "UnivariateEntropyMultiImageMetric.h"
#include "BSplineDeformableTransformOpt.h"
#include "GradientDescentLineSearchOptimizer.h"
```

```
#define Dimension         3
#define InternalPixelType float
```

As itk::ImageFileReader casts input images to specified internal type we only define an internal image type. For metric calculations pixel type should have real value (float or double). To decrease the memory requirements we use float type.

```
typedef itk::Image< InternalPixelType, Dimension >      InternalImageType;
```

We declare the registration method type by using the internal image type as a template argument. This class handles the interconnection between registration components and uses a multi-resolution optimization.

```
typedef itk::MultiResolutionMultiImageRegistrationMethod< InternalImageType >
                                                          RegistrationType;
RegistrationType::Pointer   registration  = RegistrationType::New();
```

First we set the number of images to be used in the registration and the number of multi-resolution levels. The number N is parsed from the input parameters.

```
registration->SetNumberOfImages(N)
registration->SetNumberOfLevels( multiLevelAffine );
```

To optimize the objective function we use gradient descent algorithm combined with line search. We declare the optimizer type and connect to the registration using SetOptimizer() method.

```
typedef itk::GradientDescentLineSearchOptimizer  LineSearchOptimizerType;
LineSearchOptimizerType::Pointer lineSearchOptimizer;
lineSearchOptimizer     = LineSearchOptimizerType::New();

registration->SetOptimizer(     lineSearchOptimizer     );
```

Next, we define the metric type by passing internal image type as a template argument. To avoid if statements later in the code we assign metric pointers to the base class typedef itk::MultiImageMetric. The metric pointer is connected to the registration using SetMetric() method. The type of the metric to be used can be set through parameters in the input files.

```
typedef itk::MultiImageMetric< InternalImageType>                   MetricType;
typedef itk::VarianceMultiImageMetric< InternalImageType>           VarianceMetricType;
typedef itk::UnivariateEntropyMultiImageMetric< InternalImageType>  EntropyMetricType;
MetricType::Pointer                 metric;
if(metricType == "variance")
{
  metric        = VarianceMetricType::New();
}
else
{
  EntropyMetricType::Pointer entropyMetric        = EntropyMetricType::New();
  entropyMetric->SetImageStandardDeviation(parzenWindowStandardDeviation);
  metric = entropyMetric;
}

registration->SetMetric( metric  );
```

After instantiating the metric and the optimizer, we connect images to the registration method. Each image is associated with an interpolator and a transform. Therefore, we start by declaring interpolator and transform arrays.

```
typedef itk::AffineTransform< ScalarType, Dimension >   TransformType;
typedef std::vector<TransformType::Pointer>             TransformArrayType;
TransformArrayType                          affineTransformArray(N);

typedef itk::LinearInterpolateImageFunction<InternalImageType,ScalarType>  InterpolatorType;
typedef vector<InterpolatorType::Pointer>                              InterpolatorArrayType;
InterpolatorArrayType      interpolatorArray(N);
```

We instantiate interpolator and transforms in a for loop and connect to the registration method

```
affineTransformArray[i]     = TransformType::New();
registration->SetTransformArray( i, affineTransformArray[i]);

interpolatorArray[i]  = InterpolatorType::New();
registration->SetInterpolatorArray( i,interpolatorArray[i]);
```

To perform a multi-resolution registration we build image pyramids and connect to the registration method. We start by declaring image pyramid filters.

```
typedef itk::RecursiveMultiResolutionPyramidImageFilter<InternalImageType,InternalImageType>
                                                          ImagePyramidType;
typedef  vector<ImagePyramidType::Pointer>                ImagePyramidArray;
ImagePyramidArray imagePyramidArray(N);
```

In a for loop we construct image pyramid and connect them to registration using `SetImagePyramidArray` function.

```
imagePyramidArray[i] = ImagePyramidType::New();
imagePyramidArray[i]->SetNumberOfLevels( multiLevelAffine );
imagePyramidArray[i]->SetInput( imageReader->GetOutput() );
imagePyramidArray[i]->Update();

registration->SetImagePyramidArray(i, imagePyramidArray[i]);
```

We now set up the initial parameters of the registration. Registration method uses a parameters array which is formed by the concatenation of individual transform parameters.

```
typedef RegistrationType::ParametersType ParametersType;
ParametersType initialAffineParameters( affineTransformArray[0]->GetNumberOfParameters()*N );
initialAffineParameters.Fill(0.0);

registration->SetInitialTransformParameters( initialAffineParameters );
```

Next, we specify the region from which the samples are taken. We specify the sample region to be the whole image region.

```
InternalImageType::RegionType fixedImageRegion;
imagePyramidArray[0]->GetOutput(imagePyramidArray[0]->GetNumberOfLevels()-1)->GetBufferedRegion();
registration->SetFixedImageRegion( fixedImageRegion );
```

In the following line we set the number samples to be used in the metric. For choosing the parameters of the algorithm see the section C.1 on registration parameters.

```
metric->SetNumberOfSpatialSamples( numberOfSamples );
```

Now, we all components are instantiated and the registration is ready to be started

```
registration->StartRegistration();
```

We use the results of global affine registration above to initialize B-spline transforms. We start by declaring the B-spline transform type.

```
const unsigned int SplineOrder = 3;
typedef double CoordinateRepType;
typedef itk::BSplineDeformableTransformOpt<  CoordinateRepType,Dimension,SplineOrder >
                                              BSplineTransformType;
typedef vector<BSplineTransformType::Pointer> BSplineTransformArrayType;
BSplineTransformArrayType      bsplineTransformArrayLow(N);
```

Next, we set the total parameters length of B-splines. This is necessary as ITK's B-spline implementation only holds pointers to actual transform parameters. We calculate length of the B-spline parameters explicitly from the input parameters.

```
registration->SetTransformParametersLength(
             static_cast<int>( pow( static_cast<double>(bsplineInitialGridSize+SplineOrder),
                                    static_cast<int>(Dimension))*Dimension*N ) );
```

To initialize B-splines using the results of the global affine registration we get the latest parameters from the registration.

```
ParametersType affineParameters = registration->GetLastTransformParameters();
ParametersType affineCurrentParameters(affineTransformArray[0]->GetNumberOfParameters());

for( int i=0; i<N; i++)
{
  for(unsigned int j=0; j<affineTransformArray[0]->GetNumberOfParameters(); j++)
  {
    affineCurrentParameters[j]=
                affineParameters[i*affineTransformArray[0]->GetNumberOfParameters()+j];
  }
  affineTransformArray[i]->SetParametersByValue(affineCurrentParameters);
}
```

In a for loop, we initialize B-splines and connect them to registration method.

```
bsplineTransformArrayLow[i] = BSplineTransformType::New();
bsplineTransformArrayLow[i]->SetBulkTransform(affineTransformArray[i]);
bsplineTransformArrayLow[i]->SetParameters( bsplineParametersArrayLow[i] );

registration->SetInitialTransformParameters
                         (i, bsplineTransformArrayLow[i]->GetParameters());
registration->SetTransformArray(i, bsplineTransformArrayLow[i]);
```

To make use of the B-spline optimization, we explicitly connect B-spline transforms to the metric.

```
metric->SetBSplineTransformArray(i, bsplineTransformArrayLow[i]);
```

Now, B-spline registration can be started

```
registration->StartRegistration();
```

After each stage of the algorithm, e.g. affine and B-spline registration, transform parameters are written to files using `itk::TransformFileReader`.

To obtain a dense deformation field capturing variations at different scales, we gradually increase the complexity of the deformation field by refining the grid of B-spline control points. See the example code on how to initialize B-spline control points at finer grids using results at coarser grids.

C Using Command Line Modules

We provide the modules `CreateImageSetAffine` and `CreateImageSetBspline` to generate synthetic datasets from a sample data. These modules apply random transforms to the input image, generate a dataset with the specified number of images and write the results to an output folder. The usage of the command line modules is as follows

```
CreateImageSetAffine inputImage outputFolder numberOfImages
```

`GroupwiseRegistration` is the main command line tool to perform groupwise registration. The parameters of the registration algorithm are passed to the binary using two text files. Folders under `Testing/Temporary` contain the text files that we used for each experiment. The following line shows the usage of the binary

```
GroupwiseRegistration filenames.txt parameters.txt
```

The first text file contains the paths of the input folder, output folder and the file names. The second text file supplies the parameters of the registration algorithm and is explained more in detail in the next section. A sample `filenames.txt` for setting up a registration with two input images is as follows

```
-i inputFolder/
-o outputFolder/

-f filename1
-f filename2
```

The registration code only outputs transform parameters. These transform parameters can be used to visualize registration results. We provide `ComputeOutputs` command line module to visualize registration results in 3D. This module outputs transformed images, mean and standard deviation images along with central slices. `ComputeOutputs` takes the same input arguments as `GroupwiseRegistration` and can be used as follows

```
ComputeOutputs filenames.txt parameters.txt
```

C.1 Registration Parameters

Registration parameters for each experiment can be found under corresponding folders in `Testing/Temporary`. Here we briefly go over the registration parameters used in a non-rigid regis-

tration setting.

The metric type can be specified using the option `-metricType`. `entropy` option computes sum of univariate entropies and `variance` option computes sum of variances along pixel stacks.

```
-metricType entropy
```

By default, global affine registration is performed. B-spline registration can be turned on by using the `-useBspline on` option. `-useBsplineHigh` option specifies whether to use mesh refinement in the registration.

```
-useBspline on
-useBsplineHigh off
```

The initial size of the B-spline deformation field can be set using the following option. To capture anatomical variations at different scales we start with a coarse size of 8 points and gradually increase the number of control points.

```
-bsplineInitialGridSize  8
```

If `-useBsplineHigh` is turned on, the number of mesh refinements can be specified using the `-numberOfBsplineLevel` option. After each level of B-spline registration the number of B-spline control points are doubled, e.g. $8 \rightarrow 16 \rightarrow 32$ for a two level mesh refinement starting from an initial size of 8.

```
-numberOfBsplineLevel 2
```

For stochastic optimization we randomly subsample the image domain. Increasing the percentage of samples increases the registration accuracy; however, it also increases the registration time. Empirically, we found the following parameters as a good trade-off between registration accuracy and run-time.

```
-numberOfSpatialSamplesAffinePercentage      0.0025
-numberOfSpatialSamplesBsplinePercentage     0.0050
-numberOfSpatialSamplesBsplineHighPercentage 0.0200
```

For a successful registration, multi-resolution optimization plays a key role. For all experiments we used a three level multi-resolution optimization. If the resolution of the input images are smaller than the data we used ($\approx 200 \times 200 \times 200$) the number of multi-resolution levels can be decreased to two.

```
-multiLevelAffine 3
-multiLevelBspline 3
-multiLevelBsplineHigh 3
```

For optimization we used fixed number of iterations as the stopping criteria. The command line module `GroupwiseRegistration` outputs metric values every ten iterations. These outputs can be used to check the convergence of the algorithm. The number of iterations can be increased for higher registration accuracy. During the tests we observed that the number of iterations should be increased with an increase in the number of input images.

```
-optAffineNumberOfIterations 50
-optBsplineNumberOfIterations 75
-optBsplineHighNumberOfIterations 50
```

The initial step size of the gradient descent algorithm can be set using the following options. As we use line search for the actual step size, the registration accuracy is relatively robust to step size.

```
-optAffineLearningRate  1e-4
-optBsplineLearningRate 1e4
-optBsplineHighLearningrate 1e4
```

An important parameter for `itk::UnivariateEntropyMultiImageMetric` is the width of the Gaussian kernel to compute the entropy. We observed that approximately five percent of the range of the intensity values work in practice. For intensity values ranging between 0-255 we used a value of 10.

```
-parzenWindowStandardDeviation 10
```

References

[1] K K Bhatia, J V Hajnal, B K Puri, A D Edwards, and D Rueckert. Consistent groupwise non-rigid registration for atlas construction. In *IEEE ISBI*, 2004. 2.2, 2.2

[2] W R Crum, T Hartkens, and D L G Hill. Non-rigid image registration: Theory and practice. *The British Journal of Radiology*, 77(140–153), 2004. 1

[3] R.O. Duda and P.E. Hart. *Pattern Classification and Scene Analysis*. John Wiley and Sons, 1973. 2.1

[4] L. Ibanez, W. Schroeder, L. Ng, and J. Cates. *The ITK Software Guide*. Kitware, Inc. ISBN 1-930934-10-6, http://www.itk.org/ItkSoftwareGuide.pdf, first edition, 2003. 3

[5] S Joshi, Brad Davis, Matthieu Jomier, and Guido Gerig. Unbiased diffeomorphic atlas construction for computational anatomy. *NeuroImage*, 23:151–160, 2004. 3.1

[6] Stefan Klein, Marius Staring, and Josien P.W. Pluim. A comparison of acceleration techniques for nonrigid medical image registration. In *WBIR*, pages 151–159, 2006. 3.1

[7] E. Miller, N. Matsakis, and P. Viola. Learning from one example through shared densities on transforms. In *IEEE CVPR*, pages 464–471, 2000. 1, 2

[8] D. Rueckert and et al. Nonrigid registration using free-form deformations: Application to breast mr images. *IEEE TMI*, 22:120–128, 2003. 2.2

[9] A Toga and P Thompson. The role of image registration in brain mapping. *Image and Vision Computing*, 19(3–24), 2001. 1

[10] Barbara Zitova and Jan Flusser. Image registration methods: A survey. *Image and Vision Computing*, 21(977–1000), 2003. 1

[11] Lilla Zollei, E. Learned-Miller, E. Grimson, and W. Wells. Efficient population registration of 3d data. In *Computer Vision for Biomedical Image Applications, ICCV*, pages 291–301, 2005. 1, 2