Affine Transformation for Landmark Based Registration Initializer in ITK

Release 0.00

E.Y. Regina Kim[1,2], Hans J. Johnson[1,2] and Norman K. Williams[1]

July 11, 2011

[1]Biomedical Engineering Dept., University of Iowa, Iowa city
[2]Psychiatry, University of Iowa General Hospital, Iowa city

Abstract

This document describes an affine transformation algorithm as an additional feature for landmark based registration in ITK www.itk.org. The algorithm is based on the paper by Späth, H [1]. The author derives a set of linear equations from paired landmarks and generates an affine transform from them. The method implemented here gives more freedom in the choice of registration and/or initialization method in ITK. The submission describes ITK implementation of the algorithm.

Contents

1 Introduction

The work in [1] describes mathematics for calculating affine transformation from the number of paired landmarks. For any number of dimension images, the algorithm will calculate a best affine based in the least squares sense.

For the two sets of points P and Q given by

$$p_i = (p_{1i}, ..., p_{ni})^T, q_i = (q_{1i}, ..., q_{ni})^T \qquad (i = 1, ..., m)$$

The author derives equation to get two matrixs, A and t, to satisfy

$$p_i \approx Aq_i + t \qquad (i = 1, ..., m)$$

After some calculation and rearrangement, we get the following form of equation:

$$\tilde{Q}\tilde{a}_j = \tilde{c}_j \qquad (j = 1, ..., n)$$

where

$$\tilde{q}_i = (q_{1i}, ..., q_{ni}, q_{n+1,i})^T, \qquad where \quad q_{n+1,i} = 1$$

$$\tilde{Q} = \sum_{i=1}^{m} (\tilde{q}_i \cdot \tilde{q}_i^T)$$

$$\tilde{a}_j = (a_{j1}, ..., a_{jn}, t_j)^T$$

$$\tilde{c}_j = (\tilde{c}_{j1}, ..., \tilde{c}_{j,n+1})^T \qquad with \qquad \tilde{c}_{jk} = \sum_{i=1}^{m} (q_{ki} p_{ji}) \qquad (k = 1, ..., n+1).$$

For more mathematical details, please see [1].

2 Implementation

This submission adds a functionality to the `itk::LandmarkBasedTransformInitializer`. Usage is straightforward and is demonstrated with the included CMake testing routine `itk::LandmarkBasedTransformInitializerTest.cxx`

We could start by including following ITK class.

```
#include <itkLandmarkBasedTransformInitializer.h>
```

The user instantiates the filter and sets the input landmarks, which is ordered pair of fixed and moving landmarks. For the affine transformation, the user starts with instantiating of `itk::AffineTransform`, and set the transformation for the initializer.

```
typedef itk::Image< PixelType, 3>  ImageType;

typedef itk::AffineTransform<double,3> TransformType;

TransformType::Pointer transform = TransformType::New();

typedef itk::LandmarkBasedTransformInitializer< TransformType,
                                                ImageType, ImageType > TransformInitializerType;

TransformInitializerType::Pointer initializer = TransformInitializerType::New();

TransformInitializerType::LandmarkPointContainer fixedLandmarks;
TransformInitializerType::LandmarkPointContainer movingLandmarks;
```

Set the paired landmarks:

```
double fixedLandMarkInit[6][3] =
  {
    { -1.33671 , -279.739 , 176.001 },
    { 28.0989  , -346.692 , 183.367 },
    { -1.36713 , -257.43  , 155.36 },
    { -33.0851 , -347.026 , 180.865 },
    { -0.16083 , -268.529 , 148.96 },
    { -0.103873, -251.31  , 122.973 }
  };
double movingLandmarkInit[6][3] =
  {
    { -1.66705 , -30.0661 , 20.1656},
    { 28.1409  , -93.1322 , -5.34366},
    {-1.55885  , -0.495696, 12.7584},
    {-33.01651 , -92.0973 , -8.66965},
    {-0.189769 , -7.3485  , 1.75063},
    {0.1021    , 20.2155  , -12.8466}
  };
for(unsigned i = 0; i < 6; i++)
  {
  TransformInitializerType::LandmarkPointType fixedPoint, movingPoint;
  for(unsigned j = 0; j < 3; j++)
    {
    fixedPoint[j] = fixedLandMarkInit[i][j];
    movingPoint[j] = movingLandmarkInit[i][j];
    }
  fixedLandmarks.push_back(fixedPoint);
  movingLandmarks.push_back(movingPoint);
  }

  initializer->SetFixedLandmarks(fixedLandmarks);
  initializer->SetMovingLandmarks(movingLandmarks);

  initializer->SetTransform(transform);
```

The computation of transformation could be start as following:

```
initializer->InitializeTransform();
```

3 Sample Results

The test result of brain landmarks is shown here. The test data set is the same data set which included in the testing routine `itk::LandmarkBasedTransformInitializerTest.cxx`. Our fixed and moving landmarks constitutes 6 pairs. Figure 1 2 shows our landmarks in 3D view with 3D Slicer `www.slicer.org`. After initialization, moving landmarks are well overlaied with transformed landmarks. Please see figure 3.

References

[1] H Späth. Fitting affine and orthogonal transformations between two sets of points. *Mathematical Communications*, 2004. (document), 1

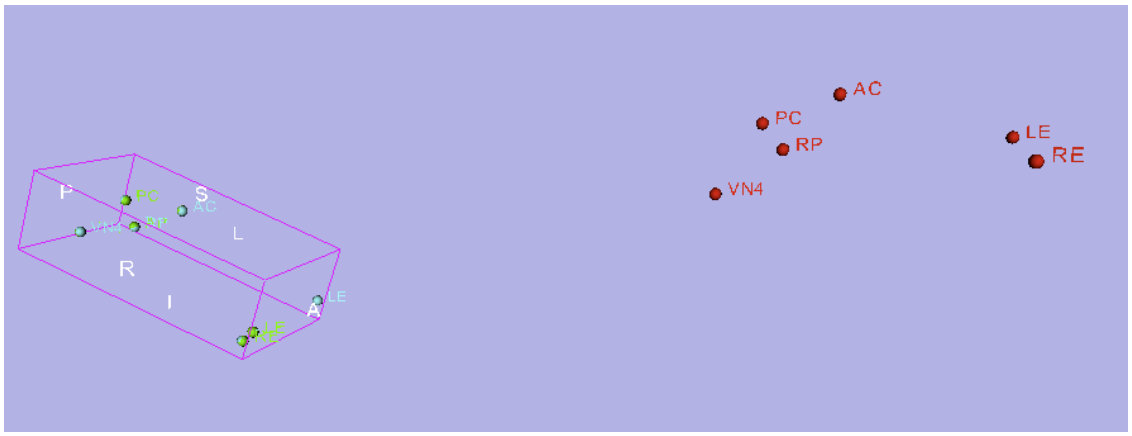


Figure 1: Fixed and moving landmarks for 3D Affine transformation with 6 paired set points.

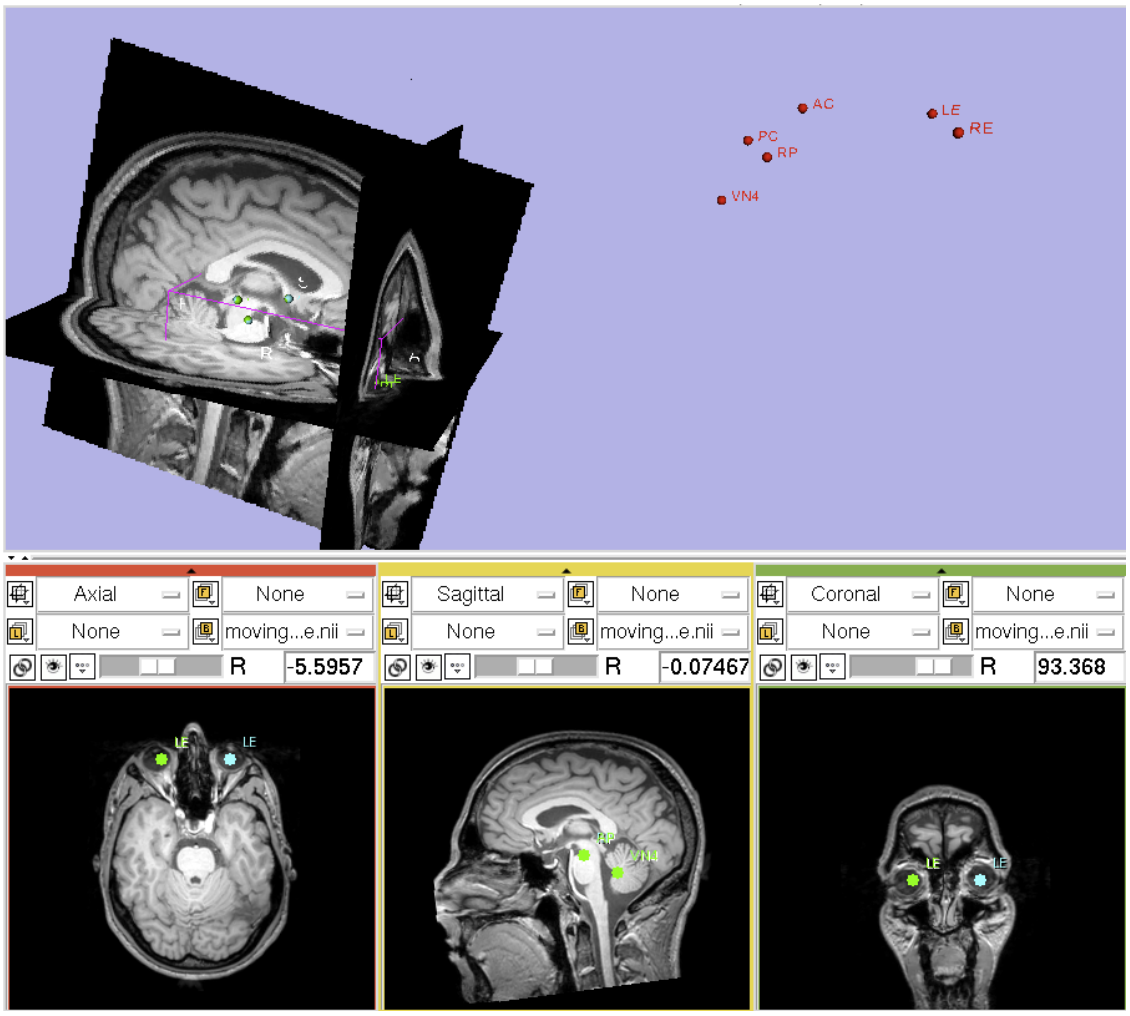
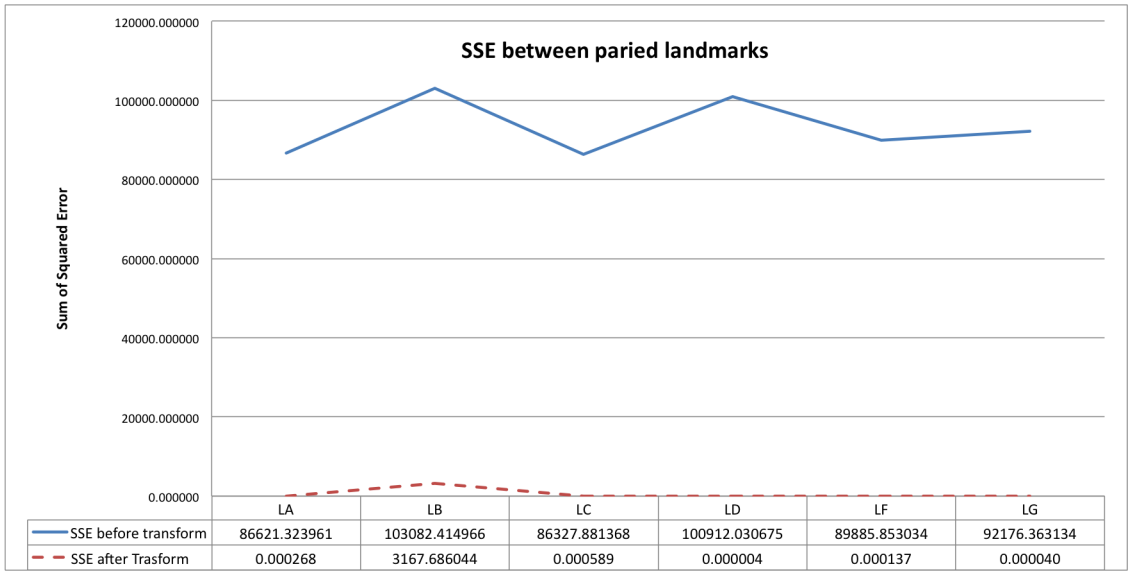


Figure 2: Fixed and moving landmarks for 3D Affine transformation with 6 paired set points where brain MR image overlaid.



	LA	LB	LC	LD	LF	LG
SSE before transform	86621.323961	103082.414966	86327.881368	100912.030675	89885.853034	92176.363134
SSE after Trasform	0.000268	3167.686044	0.000589	0.000004	0.000137	0.000040

Figure 3: Fixed and moving landmarks for 3D Affine transformation with 6 paired set points. Sum of squared error(SSE) were calculated to see how well the algorithm performes for the case. Graph shows that the SSE is very small after transformation (red dotted line) than before the transformation (solid blud line), and actual numbers for each plot are shown right below the graph