Improvements to the itk::KernelTransform and subclasses

Release 1.10

Rupert Brooks and Tal Arbel

August 15, 2007

McGill Centre for Intelligent Machines
McGill University, Montreal, Canada
rupert.brooks@mcgill.ca, arbel@cim.mcgill.ca

Abstract

Kernel-based transforms such as the thin plate spline are frequently used to model deformations in medical imaging. The existing implementation in ITK is capable of being used to warp images, but does not work in the registration framework. The existing implementation is inefficient, requiring recomputation of all cached values at every parameter change, and the Jacobian calculation is not implemented. By reversing the roles of the fixed and moving parameters, the transform can be adapted for registration use. We present modified classes which are more efficient, and calculate the Jacobian correctly.

Contents

1 Introduction

Deformable transformations based on a small set of matched points are extremely useful in medical imaging. While the deformation at each point is explicitly defined by the correspondence, the deformation between points must be interpolated. There are a family of spline based approaches which interpolate a smooth deformation field, of which the most widely known is the thin plate spline [3, 1]. In the ITK, this family of transformations is referred to as *kernel* transformations, because they can be expressed as linear combinations of radially symmetric kernel functions centered on each point.

2 Kernel Transforms

We begin with two sets of n corresponding landmark points, in a space with dim dimensions. We will refer to these as the source landmarks, $P_S = \{\mathbf{p}_{S_i}\}$, and the target landmarks, $P_T = \{\mathbf{p}_{T_i}\}$. A displacement vector, $\mathbf{d}_i = \mathbf{p}_{T_i} - \mathbf{p}_{S_i}$, can be defined at each point, and these vectors can be grouped into one long vector, $\mathbf{D}$, as follows:

$$\mathbf{D} = \left[\begin{array}{cccc} \mathbf{d}_1^T & \mathbf{d}_2^T & \dots & \mathbf{d}_n^T \end{array} \right]. \tag{1}$$

The kernel function, $g(\mathbf{x})$ maps dim-dimensional vectors onto $dim \times dim$ symmetric matrices.

In this model, a deformation field, $\mathbf{F}(\mathbf{x})$, is viewed as a set of dim independent functions of spatial position,$\mathbf{x}$, $\mathbf{F}(\mathbf{x}) = [f_1(\mathbf{x}), ..., f_{dim}(\mathbf{x})]^T$. Thus there is a separate, independent spline for the displacement in each coordinate. Each such spline is considered to be a combination of a linear (affine) transformation, plus a weighted combination of kernel functions centered at each point.

$$\mathbf{F}(\mathbf{x}) = C^T \cdot G(\mathbf{x}) + A \cdot \mathbf{x} + \mathbf{b} \tag{2}$$

where C^T is a $dim \times n$ vector of weights, G is the dim $\times dim \cdot n$ matrix made by stacking the kernel functions centered at each landmark point, evaluated at the point of interest, and A and $\mathbf{b}$ are a linear transformation in the coordinates. Thus the complete spline is defined by $n \cdot dim + dim^2 + dim$ parameters.

For n landmark points the value of the displacement field at each point provides $dim \cdot n$ constraints. There are fewer equations than unknowns so this leaves the system underdetermined. A further constraint is applied by requiring that the deformation should flatten out to an affine transformation far from all the landmarks. This flatness constraint can be developed into the set of linear equations [3]

$$P_S^T \cdot C = 0, \tag{3}$$

where P_S is the source landmark coordinates arranged in a matrix, as follows:

$$P_S = \left[\begin{array}{cccc} \mathbf{p}_{S_{1_1}} I & \cdots & \mathbf{p}_{S_{1_{dim}}} I & I \\ \vdots & \dots & \vdots & \vdots \\ \mathbf{p}_{S_{n_1}} I & \cdots & \mathbf{p}_{S_{n_{dim}}} I & I \end{array} \right] \tag{4}$$

This provides enough additional equations to make the system have full rank.

Combining these components in the following way

$$Y = \begin{bmatrix} D \\ 0 \end{bmatrix} \tag{5}$$

$$L = \begin{bmatrix} K & P_S \\ P_S^T & 0 \end{bmatrix} \tag{6}$$

$$W = \begin{bmatrix} C \\ \mathbf{a}_1^T \\ \vdots \\ \mathbf{a}_{dim}^T \\ \mathbf{b}^T \end{bmatrix} \tag{7}$$

$$\tag{8}$$

where the $\mathbf{a}_i$ are the rows of A, and K is the matrix of kernel functions at each point combination,

$$K = \begin{bmatrix} g(\mathbf{p}_{S_1} - \mathbf{p}_{S_1}) & \dots & g(\mathbf{p}_{S_1} - \mathbf{p}_{S_n}) \\ \vdots & \dots & \vdots \\ g(\mathbf{p}_{S_n} - \mathbf{p}_{S_1}) & \dots & g(\mathbf{p}_{S_n} - \mathbf{p}_{S_n}) \end{bmatrix} \tag{9}$$

we can write a set of linear equations for the parameters of each spline, $L \cdot W = Y$, and solve for the weights [2, 1, 3] , W, as

$$W = L^{-1} \cdot Y \tag{10}$$

The classic spline interpolation can be relaxed to allow for some misfit at the landmark points, by adding a multiple of the identity to the matrix K [4]. That is, L becomes

$$L = \begin{bmatrix} K + \lambda I & P_S \\ P_S^T & 0 \end{bmatrix} \tag{11}$$

3 Problems with the existing implementation

The ITK system contained a kernel transform implementation which consisted of the `itk::KernelTransform`, and its subclasses (`itk::ElasticBodySplineKernelTransform`, `itk::ElasticBodyReciprocalSplineKernelTransform`, `itk::ThinPlateR2LogRSplineKernelTransform`, `itk::ThinPlateSplineKernelTransform` and `itk::VolumeSplineKernelTransform`). Most of the implementation was in `itk::KernelTransform` with each subclass simply reimplementing the specific kernel function, G and possibly the computation of the deformation contribution where efficient, specialized methods exist.

This implementation was suitable for warping images, but did not integrate well into the ITK registration framework for two reasons. Primarily, the Jacobian of the transformation was not implemented. Indeed, the code only contained this comment in the Jacobian method:

```
// TODO
// The Jacobian should be computable in terms of the matrices
// used to Transform points...
```

Without a Jacobian, gradient based optimizers cannot be used, and approaches such as Powell's method or Amoeba (Downhill Simplex) are not well suited for transformations with this many parameters.

A more serious problem was that the implementation required that the weights, W needed to be recomputed each time the parameters were set. This is extremely computationally inefficient when the class is used in an optimization framework. This requirement arose because the source landmark positions were considered to be the moving parameters. From the development in section 2 it is clear that changing the locations of P_S requires recomputing K and therefore L and W. Furthermore, there is not a clear way to take the derivative of $\mathbf{F}(\mathbf{x})$ with respect to the source parameter positions, which is most likely why the Jacobian was not completed.

4 Proposed changes

Here we propose to reverse the role of the fixed and moving parameters, so that the `SetParameters()` method updates the *target* landmarks rather than the source landmarks. In this case, L^{-1} does not change when the moving parameters are updated, so it can be precomputed and cached. On each update of the parameters, Y needs to be updated, and the weights W can be found through a matrix multiplication via Equation 10. This way, the parameters can be changed repeatedly in the optimization framework at much less computational cost.

Furthermore, with this formulation, the Jacobian of the transformation becomes easy to calculate.

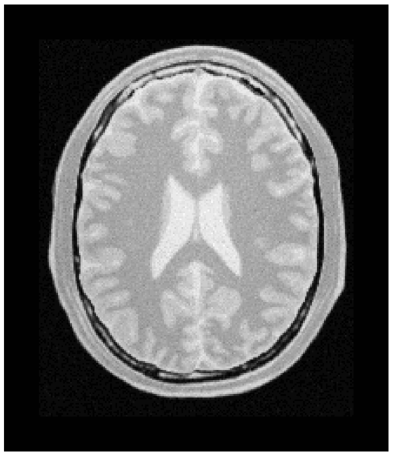
$$\begin{aligned}
\mathbf{F}(\mathbf{x}) &= \begin{bmatrix} G(\mathbf{x}) & \mathbf{x}_1 I & \dots & \mathbf{x}_{dim} I & I \end{bmatrix} \cdot W && (12)\\
\mathbf{F}(\mathbf{x}) &= \begin{bmatrix} G(\mathbf{x}) & \mathbf{x}_1 I & \dots & \mathbf{x}_{dim} I & I \end{bmatrix} \cdot L^{-1} \cdot Y && (13)\\
\frac{\partial \mathbf{F}(\mathbf{x})}{\partial \phi} &= \begin{bmatrix} G(\mathbf{x}) & \mathbf{x}_1 I & \dots & \mathbf{x}_{dim} I & I \end{bmatrix} \cdot L^{-1} \cdot \frac{\partial Y(\phi)}{\partial \phi} && (14)\\
\frac{\partial \mathbf{F}(\mathbf{x})}{\partial \phi} &= \begin{bmatrix} G(\mathbf{x}) & \mathbf{x}_1 I & \dots & \mathbf{x}_{dim} I & I \end{bmatrix} \cdot L^{-1} \cdot \begin{bmatrix} Z \\ 0 \end{bmatrix} && (15)
\end{aligned}$$

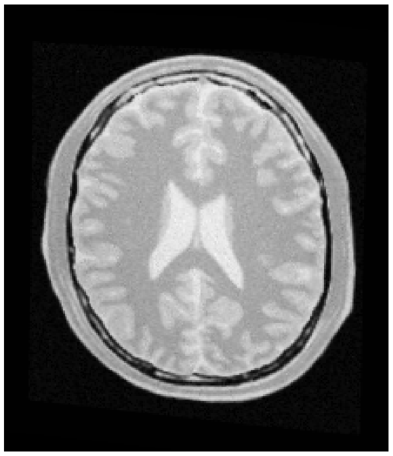
where Z is a $n \cdot d$ vector with all entries -1.

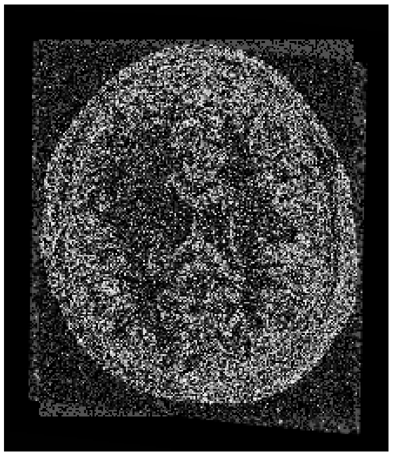
Conceptually, this implementation supports the idea that the positions of the source landmarks (or fixed parameters) are the positions of landmark points in the coordinate system of the fixed image. The position of the target landmarks, (or moving parameters) will then be the corresponding positions of these points in the moving images. (The previous approach corresponded to the idea that the fixed parameters were the positions of landmarks in the target image, and the moving parameters were the locations of these landmarks in the coordinate system of the fixed image.)

However, there are subtle differences in the meaning of the transformation when expressed in this way. Each kernel spline transformation is capable of expressing a certain subset of all possible transformations. This subset of transformations is completely determined by the number and positions of the source landmarks, P_S. Therefore, the set of transformations that can be expressed is not exactly the same, if the roles of the source and target landmarks are interchanged. However, under normal circumstances, we know of no compelling argument why the set of transformation defined by the source landmarks is more useful than the set of transformations defined by the target landmarks.

This paper presents a new implementation of the `itk::KernelTransform` and its subclasses that reflects these changes. So that these classes may be used and tested without interfering with the existing implementation, all the names have been suffixed with a "2". Ultimately we propose that these changes


(a) Original Brain Image


(b) Warped Brain Image


(c) Difference before registration

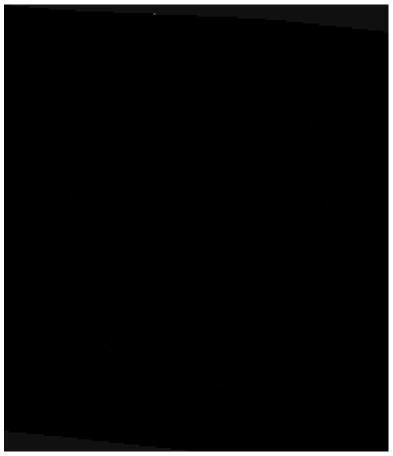

(d) Difference after registration

Figure 1: Example registration of a warped 2D image.

should be integrated into the main implementation. Besides this minor name change, only the base class (`itk::KernelTransform`) required changes to the code.

Specifically, the roles of the source and target landmarks have been interchanged in all the parameter setting code. The displacements, D, are now computed in the `ComputeY()` rather than in `ComputeK()`. To support the more efficient implementation, two new methods have been added to the class `itk::KernelTransform`. The method `ComputeLInverse()` computes and stores the inverse of the L matrix. A convenience method `SetIdentity()` was also provided which sets the target landmarks to be the same as the source landmarks. These methods are supported by new variables `m_LMatrixInverse` which stores the inverse of L and the booleans `m_LMatrixComputed` and `m_LInverseComputed` which serve as status indicators for L and L^{-1}. Finally, the `GetJacobian()` method now computes the Jacobian, using Equation 15.

5 Testing

Testing of the original implementation of the kernel splines was performed by a single test program (`itkSplineKernelTransformTest.cxx`) for all the subclasses of `itk::KernelTransform`. These tests passed on the new version of the spline classes without changes. However, as written, these tests defined sets of landmark points that were related by an affine transform. To better test the deformation part of the transform, the tests have been altered slightly so that the landmarks generated are not related so simply. Additional tests have been added to test the computation of the Jacobian. The Jacobian is computed for a known point and transformation, and the result is compared to a result numerically calculated independently in Matlab.

An example program (`TPSDeformableRegistration.cxx`) similar to the example program `DeformableRegistration6.cxx` is provided to show how the new spline classes can be used in a registration problem. The brain slice from the ITK examples, and a warped version of it are used as input. Because the fixed point positions are exactly those used to warp the image in the first place a near perfect fit is obtained.

6 Conclusion

This paper presents modifications to the `itk::KernelTransform` and its subclasses that enable them to be used efficiently in the ITK registration framework. The role of the source and target landmarks has been interchanged so that the parameters can be updated without requiring a matrix inversion, and so that the expression for the transform Jacobian is simplified. The changed class passes all tests that were used for the old class, and an example showing the use of the class in the registration framework is provided. We propose that this formulation of the `itk::KernelTransform` should replace the existing one.

References

[1] Fred L. Bookstein. Principal warps: Thin-plate splines and the decomposition of deformations. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 11(6):567–585, 1989. 1, 2

[2] Malcolm H. Davis, Alireza Khotanzad, Duane P. Flamig, and Steven E. Harms. A physics-based coordinate transformation for 3-d image matching. *IEEE Trans. Medical Imaging*, 16(3):317–328, 1997. 2

[3] R. L. Harder and R. N. Desmarais. Interpolation using surface splines. *Journal of Aircraft*, 9(2):189–191, 1972. 1, 2, 2

[4] Rainer Sprengel, Karl Rohr, and H. Sigfried Steihl. Thin-plate spline approximation for image registration. In *18th Internat. Conf. of the IEEE Engineering in Medicine and Biology Society*, 1996. 2