Adapting the ITK Registration Framework to Fit Parametric Image Models

Release 1.00

Cory Quammen[1] and Russell M. Taylor II[1]

April 30, 2010

[1]Center for Computer Integrated Systems for Microscopy and Manipulation
University of North Carolina at Chapel Hill, NC, USA

Abstract

The image registration framework in the Insight Tookit offers a powerful body of code for finding the optimal spatial transform that registers one image with another. However, ITK currently lacks a way to fit parametric models of image pixel values to an input image. Such a capability is necessary in certain applications such as sub-resolution localization of molecules in fluorescence microscopy. This document describes new classes that enable the use of the registration framework to provide this capability. We describe a new base class, `itk::ParametricImageSource`, that defines an interface for parametric image sources. An adapter class `itk::ImageToParametricImageSourceMetric` that enables `itk::ParametricImageSources` to be hooked into the registration framework is also described. An example adapter class that enables the existing `itk::GaussianImageSource` to be used for image fitting is presented, and we demonstrate use of the classes by fitting a 2D Gaussian function to an image generated by the `itk::GaussianImageSource` class.

Contents

1 Parametric Models of Image Pixel Values **2**

2 Connecting Parametric Image Sources to the Registration Framework **3**

3 Writing Adapter Classes for Existing Image Sources **5**

4 Fitting a 2D Gaussian Pixel Intensity Model to an Image **6**

5 Acknowledgements **7**

INTRODUCTION

Fitting models of image pixel values to images is a task central to certain image analysis applications. For example, in fluorescence microscopy, light from an in-focus point source diffracts to an intensity pattern that approximately matches a 2D Gaussian function. Using this observation, methods for determining the positions of individual fluorescent molecules by fitting a 2D Gaussian function to the image have been developed [2, 3, 1]. The mean (center) of the fitted Gaussian is taken to be the location of the molecule. Localization accuracy of as low as 1.5 nanometers has been reported using this method [3].

Despite having mature implementations of the necessary algorithmic foundations, the Insight Toolkit (ITK) does not currently have a mechanism to enable the fitting of image pixel value models to images. This document describes a set of classes that enable such fitting.

1 Parametric Models of Image Pixel Values

An image pixel value[1] pattern can be modeled as a function $F(\mathbf{x};\mathbf{u})$ where $\mathbf{x}$ is a spatial position and $\mathbf{u}$ is a set of function parameters that change the pattern. The goal in fitting the function $F(\mathbf{x};\mathbf{u})$ to an input image $I(\mathbf{x})$ is to find the parameter vector $\mathbf{u}$ that minimizes a cost function defining how well the pixel values from the function match the pixel values of the input image. This is largely the same goal as in image registration. However, in registration, the parameter vector $\mathbf{u}$ is an argument to a spatial transformation $T(\mathbf{x};\mathbf{u})$ such that the transformed input image $I_1(T(\mathbf{x};\mathbf{u}))$ closely matches another input image $I_2(\mathbf{x})$.

An example of a pixel value profile often used in image analysis is the Gaussian function. For the case of a normalized k-dimensional Gaussian function whose major axes are aligned with the Cartesian axes, $F(\mathbf{x};\mathbf{u})$ has the form

$$F(\mathbf{x};\mathbf{u}) = \frac{s}{(2\pi)^{k/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)\right)$$

where s is a scaling factor, μ is the mean (center) vector, and Σ is a diagonal matrix whose ith diagonal element is the standard deviation σ_i of the Gaussian function along the i-th dimension. For this function, the parameter vector $\mathbf{u}$ is equal to $\{s,\sigma_1,\ldots,\sigma_k,\mu_1,\ldots,\mu_k\}$. In ITK, the class `itk::GaussianImageSource` generates images whose intensities are defined by this function when its `Normalized` flag is set to 1.

Certain ITK image filters may also be considered parametric models of image pixel values. For these filters, the function is of the form $F(\mathbf{x};\mathbf{u},I(\mathbf{x}))$ where $\mathbf{u}$ is a parameter vector as before and $I(\mathbf{x})$ is the image input for the filter. In some applications it may be desirable to optimize the filter parameters $\mathbf{u}$. For example, the `itk::MultiplyByConstantImageFilter` might be used to find an optimal intensity scaling factor for an image with respect to a particular objective function.

While ITK has parametric image sources, no unified interface for setting their parameters exists. This kind of interface, analogous to the interface currently found in subclasses of `itk::Transform`, would enable application of the optimization routines in ITK to fitting image intensity models. We propose a new subclass of `itk::ImageSource`, called `itk::ParametricImageSource`, to provide a suitable interface. This class defines three methods:

[1]We use the general term "pixel value" because the classes proposed here impose no restriction on the type of data associated with pixels, e.g., scalar intensity values or vector values.

```
virtual void           SetParameters(const ParametersType& parameters) {};

virtual ParametersType GetParameters() const
  {
  return ParametersType(0);
  }

virtual unsigned int   GetNumberOfParameters() const
  {
  return 0;
  }
```

where the `ParametersType` is defined by

```
typedef double                             ParametersValueType;
typedef Array< ParametersValueType >   ParametersType;
```

Subclasses with one or more parameters must override these three methods. For example, the `itk::GaussianImageSource` could be modified to be a subclass of `itk::ParametricImageSource`. In this case, the parameters for the Gaussian are the intensity scale, the mean, and the standard deviation. Subclasses of `itk::ImageToImageFilter` could also be modified to be a subclass of `itk::ParametricImageSource` with appropriate overriding of these three methods.

2 Connecting Parametric Image Sources to the Registration Framework

The image registration framework in ITK consists of several types of components: cost functions (e.g., `itk::MeanSquareImageToImageMetric`, `itk::NormalizedCorrelationImageToImageMetric`), spatial transforms (e.g., `itk::AffineTransform`, `itk::BSplineDeformableTransform`), interpolators (e.g., `itk::LinearInterpolateImageFunction`, `itk::WindowsSincInterpolateImageFunction`), and optimizers (e.g., `itk::GradientDescentOptimizer`, `itk::PowellOptimizer`). Briefly, an optimizer is a search routine that iteratively evaluates a cost function at different values of the transform parameters until the cost function is minimized (or possibly maximized). A cost function assigns a value to two images, a fixed image and a moving image, according to how well they match. The fixed image remains unmodified during registration while the moving image is warped according to the spatial transform. If the optimizer converges to a local minimum, the transform and parameters found by the optimizer define a locally optimal warping from the moving image to the fixed image.

We would like to be able to reuse the existing subclasses of `itk::ImageToImageMetric` as cost functions in ITK to fit parametric image models to images. These cost functions have the methods

```
MeasureType GetValue(const ParametersType &parameters) const;

void GetDerivative(const ParametersType &parameters, DerivativeType &derivative) const;

void  GetValueAndDerivative(const ParametersType &parameters, MeasureType &Value,
                            DerivativeType &Derivative) const;
```

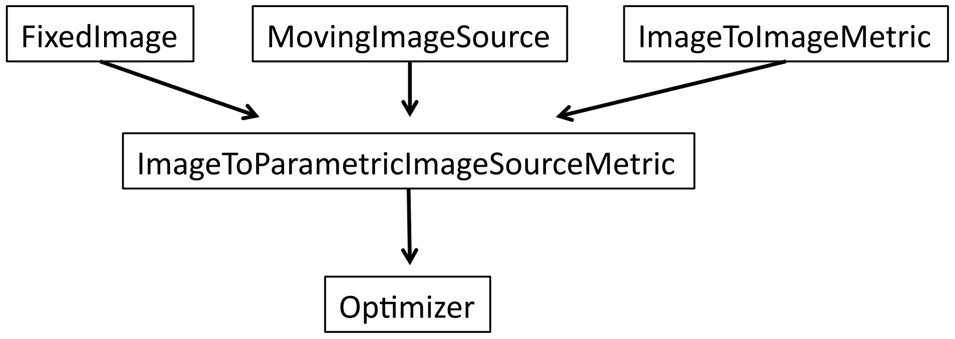

Figure 1: The itk::ImageToParametricImageSourceMetric adapts itk::ImageToImageMetrics for use in fitting image pixel value models. It sits between an existing itk::ImageToImageMetric and an itk::Optimizer.

that are called by optimizers. These methods pass their parameters argument to a itk::Transform object assigned to the image-to-image metric. This behavior makes it impossible to set parameters of a itk::ParametricImageSource through the existing registration framework. To overcome this limitation, we have defined a new subclass of itk::ImageToImageMetric, called itk::ImageToParametricImageSourceMetric, that lets us control where the parameters are passed.

Like existing itk::ImageToImageMetric subclasses, the itk::ImageToParametricImageSourceMetric is parameterized over fixed and moving image types. The moving image type, however, is expected to be a subclass of itk::ParametricImageSource, so it is actually an image source rather than static image data. The meaning of "moving" in this context means that the image intensities are changing values, rather than that the moving image is undergoing a spatial deformation.

The itk::ImageToParametricImageSourceMetric adapts a delegate itk::ImageToImageMetric for the purposes of computing the image match value. Parameters passed to the GetValue(...) method from the optimizer are forwarded to the itk::ParametricImageSource, which is updated before the delegate computes the cost function value. Figure 1 shows how the itk::ImageToParametricImageSourceMetric integrates into the registration framework.

We assume that the parametric image source has parameters that can accommodate any necessary spatial transformations. Therefore, the itk::ImageToParametricImageSourceMetric forces the delegate metric to use an itk::IdentityTransform. We do not make any assumptions about the size, spacing, or origin of the fixed or moving images, so an interpolator for the fixed image must be set using the method SetInterpolator(...).

The interface for the itk::ParametricImageSource does not currently provide a mechanism to retrieve partial derivative images that can be used for computing the gradient of the itk::ImageToImageMetric in the method GetDerivative(...). Instead, we estimate the derivative via forward differences.

Because it may be desirable to optimize over a subset of the parameters in a itk::ParametricImageSource, the itk::ImageToParametricImageSourceMetric class also holds a mask vector that indicates which parameters should be optimized. We call these *enabled parameters*. The method GetNumberOfParameters() reports the number of enabled parameters. This method tells the itk::Optimizer the dimensionality of the parameter space that it will explore, so the itk::Optimizer will pass arrays containing values for only the enabled parameters to the GetValue(...) method of itk::ImageToParametricImageSourceMetric. However, the SetParameters(...) method in `itk::ParametricImageSource` expects an array containing values for *all* the parameters. The simple solution is to retrieve the current parameters from the `itk::ParametricImageSource`, set the active parameters passed in by the `itk::Optimizer`, and then pass this modified parameter array back to the `itk::ParametricImageSource` using that class's `SetParameters(...)` method. The class `itk::ImageToParametricImageSourceMetric` has its own `SetParameters(...)` method that takes care of this step.

3 Writing Adapter Classes for Existing Image Sources

A pragmatic way to implement image model fitting is to write classes that adapt existing image source to the `itk::ParametricImageSource` interface. As an example, the implementations of the key methods in the adapter class for the `itk::GaussianImageSource` are given below.

```
template<typename TOutputImage>
void
ParametricGaussianImageSource<TOutputImage>
::SetParameters(const ParametersType& parameters)
  {
  unsigned int i;
  const unsigned int dimensions = itkGetStaticConstMacro(OutputImageDimension);
  ArrayType sigma, mean;
  for (i=0; i<dimensions; i++)
    {
    sigma[i] = parameters[i];
    mean[i]  = parameters[dimensions + i];
    }
  this->SetSigma(sigma);
  this->SetMean(mean);
  this->SetScale(parameters[2*dimensions]);
  }


template<typename TOutputImage>
typename ParametricGaussianImageSource<TOutputImage>::ParametersType
ParametricGaussianImageSource<TOutputImage>
::GetParameters() const
  {
  ParametersType parameters(GetNumberOfParameters());
  ArrayType sigma = this->GetSigma();
  ArrayType mean  = this->GetMean();

  unsigned int i;
  const unsigned int dimensions = itkGetStaticConstMacro(OutputImageDimension);
  for (i=0; i<dimensions; i++)
    {
    parameters[i] = sigma[i];
    parameters[dimensions + i] = mean[i];
    }
  parameters[2*dimensions] = this->GetScale();

  return parameters;
  }

template<typename TOutputImage>
unsigned int
ParametricGaussianImageSource<TOutputImage>
::GetNumberOfParameters() const
  {
  // Standard deviation vector, mean vector, and scale.
  return 2*itkGetStaticConstMacro(OutputImageDimension) + 1;
  }
```

4 Fitting a 2D Gaussian Pixel Intensity Model to an Image

Several nanometer-resolution fluorescence microscopy imaging methods rely on fitting a 2D Gaussian to the pattern of image intensities produced by a single fluorescent molecule. Here, we show how this can be accomplished using ITK's registration framework and the adapter classes presented in this document.

First, we include the necessary software components from the registration framework:

```
#include <itkAmoebaOptimizer.h>
#include <itkLinearInterpolateImageFunction.h>
#include <itkMeanSquaresImageToImageMetric.h>
```

Next, we include the existing `itk::GaussianImageSource` along with the image-to-image metric adapter and the adapter for the `itk::GaussianImageSource`.

```
#include <itkImageToParametricImageSourceMetric.h>
#include <itkGaussianImageSource.h>
#include <itkParametricGaussianImageSource.h>
```

We then define some useful `typedefs`:

```
typedef double                                        PixelType;
typedef itk::Image<PixelType, 2>                      GaussianImageType;
typedef itk::GaussianImageSource<GaussianImageType> GaussianImageSourceType;
typedef itk::ParametricGaussianImageSource<GaussianImageType>
  ParametricGaussianImageSourceType;
typedef ParametricGaussianImageSourceType::ParametersType ParametersType;
typedef itk::LinearInterpolateImageFunction<GaussianImageType, double> InterpolatorType;
typedef itk::ImageToParametricImageSourceMetric<GaussianImageType,
  ParametricGaussianImageSourceType> MetricType;
typedef itk::MeanSquaresImageToImageMetric<GaussianImageType, GaussianImageType>
  DelegateMetricType;
typedef itk::AmoebaOptimizer AmoebaOptimizerType;
```

Assuming we have a 2D itk::Image named input that has a 2D Gaussian intensity profile, we next want to set up a itk::ParametricGaussianImageSource that will be used in the fitting.

```
ParametricGaussianImageSourceType::Pointer fittingImageSource =
    ParametricGaussianImageSourceType::New();
fittingImageSource->SetSize(input->GetSize());
fittingImageSource->SetSpacing(input->GetSpacing());
fittingImageSource->SetOrigin(input->GetOrigin());
```

We set up the metric with an interpolator, delegate metric, fixed image, and moving image source.

```
InterpolatorType::Pointer interpolator = InterpolatorType::New();
DelegateMetricType::Pointer delegate = DelegateMetricType::New();

MetricType::Pointer metric = MetricType::New();
metric->SetInterpolator(interpolator);
metric->SetDelegateMetric(delegate);
metric->SetFixedImage(originalImage->GetOutput());
metric->SetFixedImageRegion(originalImage->GetOutput()->GetLargestPossibleRegion());
metric->SetMovingImageSource(fittingImage);
```

Finally, we set up the optimizer and start optimization.

```
AmoebaOptimizerType::Pointer amoebaOptimizer = AmoebaOptimizerType::New();
amoebaOptimizer->SetCostFunction(metric);
amoebaOptimizer->SetInitialPosition(startingParameters);
amoebaOptimizer->StartOptimization();
```

The companion source code file GaussianFitting2DTest.cxx contains a test of the classes for image model fitting presented here. It generates an image of a Gaussian intensity profile using the itk::GaussianImageSource with known mean, standard deviation, and scale, then fits a itk::ParametricGaussianImageSource to this image. The test shows that the known Gaussian parameters can be successfully recovered using both the itk::AmoebaOptimizer and itk::ConjugateGradientOptimizer.

5 Acknowledgements

This work was funded by NIH grant P41-EB002025-25A1.

References

[1] Eric Betzig, George H. Patterson, Rachid Sougrat, O. Wolf Lindwasser, Scott Olenych, Juan S. Bonifacino, Michael W. Davidson, Jennifer Lippincott-Schwartz, and Harald F. Hess. Imaging Intracellular Fluorescent Proteins at Nanometer Resolution. *Science*, 313(5793):1642–1645, 2006. (document)

[2] Russell E. Thompson, Daniel R. Larson, and Watt W. Webb. Precise nanometer localization analysis for individual fluorescent probes. *Biophysical Journal*, 82(5):2775–2783, 2002. (document)

[3] Ahmet Yildiz, Joseph N Forkey, Sean A McKinney, Taekjip Ha, Yale E Goldman, and Paul R Selvin. Myosin v walks hand-over-hand: single fluorophore imaging with 1.5-nm localization. *Science*, 300(5628):2061–2065, 2003. (document)