An ITK Implementation of Physics-based Non-rigid Registration Method

Release 0.00

Yixun Liu[1,2], Andriy Kot[1], Fotis Drakopoulos[1], Andriy Fedorov[1,3] Andinet Enquobahrie[4], Olivier Clatz[5] and Nikos Chrisochoides[1]

July 13, 2012

[1]CRTC Lab and Computer Science, Old Dominion University

[2]Radiology and Imaging Science, National Institutes of Health

[3]Brigham And Women's Hospital Harvard Medical School

[4]Kitware Inc

[5]Asclepios Research Laboratory at INRIA Sophia Antipolis

Abstract

As part of the ITK v4 project efforts, we have developed ITK filters for physics-based non-rigid registration (PBNRR), which satisfies the following requirements: account for tissue properties in the registration, improve accuracy compared to rigid registration, and reduce execution time using GPU and multi-core accelerators. The implementation has three main components: (1) Feature Point Selection, (2) Block Matching (mapped to both multi-core and GPU processors), and (3) a Robust Finite Element Solver. The use of multi-core and GPU accelerators in ITK v4 provides substantial performance improvements. For example, in average for the non-rigid registration of brain MRIs, the performance of the Block Matching filter is about 12 times faster when 12 hyperthreaded multi-cores are used and about 540 times faster when the Quadro 6000 with 448 threads is used in Dell Workstation.

Contents

1 Introduction

A clinically practical non-rigid registration method should take the following factors into account: speed, robustness and accuracy. The registration should be done within a short time to provide timely responses to the surgeons. The registration results should not be susceptible to image intensity inhomogeneity and artifacts. Moreover, the registration results should realistically reflect the physical deformation of the tissue.

In this paper, we present an ITK implementation of a physics-based non-rigid registration method [1]. This method relies on:

1. a sparse displacement field and a Finite Element biomechanical model to estimate the entire deformation field. The unknown deformation field is represented by a parameterized piece-wise linear polynomials, whose parameters are estimated by approximating the scattered displacements, essentially a scattered data approximation approach.
2. The sparse registration points are obtained by locating the centers of small image blocks with rich structural information, a typical feature point selection approach.
3. The displacement associated with the registration point is obtained by comparing the block surrounding the registration point with the blocks located in a predefined window, i.e., block matching.

The Physics Based Non-Rigid Registration (PBNRR) filter in ITK, itk::fem::PhysicsBasedNonRigidRegistrationMethod, is implemented by connecting the following three ITK filters into one pipeline: itk::fem::FEMScatteredDataPointSetToImageFilter, itk::MaskFeaturePointSelection, and itk::BlockMatchingImageFilter.

The sparse displacement field is characterized by sparsity and outliers, which compromises the accuracy of the estimation of the entire deformation field. To deal with sparsity of the deformation filed, the parameter estimation is regularized by a biomechanical model, which is capable of describing the physical deformation based on quite few data, i.e., the boundary condition. To make the estimation robust again outliers, the parameter are estimated as a Least Trimmed Squares (LTS) regression [5]. More specifically, at each iteration estimate parameters first without any outliers, then identify the points with larger error as outliers, finally remove outliers from the data and re-estimate the parameters. To reduce the approximation error, the entire deformation field is estimated by an iterative method that gradually shifts from an approximation problem (minimizing the sum of a regularization term and a data error term) towards an interpolation problem (least square minimization of the data error term).

Among the three filters or steps in the PBNRR filter, block matching is the most computationally intensive, about 30% of the total execution time for an average case of non-rigid registration of pre-operative and intra-operative brain MRIs (see Table 2). The block matching algorithm [1], for each block, loops over its search window to calculate the cross-correlation. In ITK implementation, we parallelized the Block Matching based on our previous work [2, 3] within ITK multithreading/GPU framework to make full use of multi-core and GPU processors available to even average computing platforms like desktops and laptops.

In this paper, we first describe the principle of the physics-based non-rigid registration method. Users are referred to [1, 2, 3] for more details about the sequential and parallel algorithms. Then, we present its ITK implementation covering three independent filters and one main filter which puts together all three filters.

Finally, we evaluate our implementation using 5 public datasets [12] on the registration of the pre-operative MRI and the intra-operative MRI.

2 Physics-Based Non-Rigid Registration Method

The physics-based non-rigid registration method is based on the following three steps: *feature point selection, block matching, and robust finite element solver*. Feature point selection identifies the small image blocks with rich structure information (see [1]), block matching estimates the displacements of these block to produce a sparse displacement filed, and the robust solver iteratively approximates the scattered data while rejecting outliers to produce an entire deformation field.

2.1 Feature point selection

The relevance of a displacement estimated with a block matching algorithm depends on the existence of highly discriminative structures within a block. The block variance is used to measure its relevance and only select a fraction of all potential blocks based on a predefined fraction. To avoid redundancy by the overlapping of blocks (i.e., eliminate blocks which are too “close to each other), a parameter of prohibited connectivity is used. Three connectivity patterns are supported in the ITK implementation: 6-connectivity, 18-connectivity, and 26-connectivity (see Section 3.1).

To address the aperture problem [6, 7], the structural tensor of the block is calculated. The structural tensor reflects the distribution of the edge detections within the block, which will be incorporated into the Finite Element solver to make the estimated node displacement to favor the reduction of the deviation along the direction orthogonal to the edge direction. To avoid finding false correspondence (e.g., the tumor resection cavity), the block selection utilizes a mask image when necessary to exclude certain portions of the image while searching for the feature points (e.g., in the case of tumor resection).

2.2 Block matching

Block Matching is a well-known technique widely used in motion coding, image processing and compression [8, 9, 10]. Block Matching is based on the assumption that a complex non-rigid transformation can be approximated by point-wise translations of small image regions. Considering a block $B(O_k)$ in a floating image centered in O_k and a predefined search window W_k in a reference image, the Block Matching algorithm consists in finding the position O_m in W_k that maximizes a similarity measure M, which can be: mean square difference of intensity (MSD), mutual information (MI), and normalized cross correlation (NCC).

$$O_m = \underset{O_i \in W_k}{\arg\max}[M(B(O_k), B(O_i))] \tag{1}$$

We use NCC as the similarity measurement; however future implementation can include MSD or other similarity measures pertinent to the application of the PBNRR filter. Figure 1 illustrates the procedures of block matching. Note that the block can be specified in both floating and reference images depending on the application. For each block detected by feature point selection, we use Block Matching to calculate the displacement of the block. The location of the block that maximizes the similarity is obtained by exhaustive search. By assembling the individual displacement vectors one can create a sparse displacement field D, which will be used by the solver to estimate the unknown displacement vector associated with the mesh nodes.

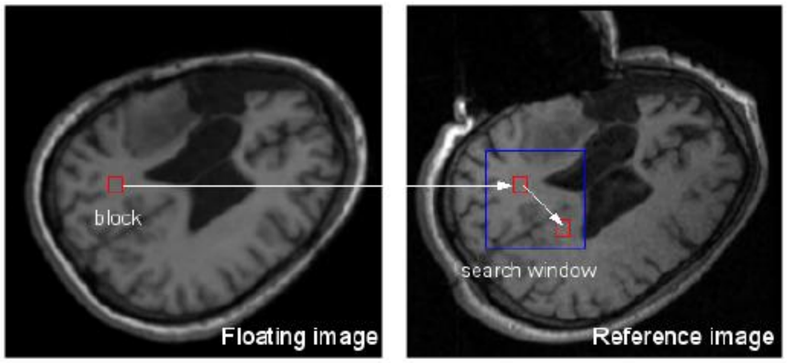


Figure 1. Block matching. For a small block in the floating image, find its corresponding block in a predefined search window in the reference or fixed image, then the displacement associated with the block can be calculated. The block can be specified in both floating and reference images depending on the application.

2.3 Finite element solver

Finite element solver is used to find the unknown displacement vector associated with the mesh nodes. The energy function is defined as,

$$W = U^T KU + (HU - D)^T S(HU - D) \qquad (1)$$

The first regularization term describes the stain energy of a linear elastic biomechanical model, and the second term describes the error between the simulated displacements and the real displacements, i.e., D. λ controls the balance of these two terms. U is the unknown node displacement vector with a size of $3n$. n is the number of nodes of the mesh. K is the mesh stiffness matrix of size $3n \times 3n$. The building of K has been well documented in [4]. H is the linear interpolation matrix of size $3p \times 3n$, where p is the number of the registration points. S is the matching stiffness matrix of size $3p \times 3p$. S is an extension to the classical diagonal stiffness matrix, taking into account the matching confidence and the local structure distribution [1], whose 3 × 3 sub-matrix S_k corresponding to registration point k is defined as:

$$S_k = \lambda \frac{n}{p} c_k T_k \qquad (2)$$

where T_k is tensor structure of the block surrounding the registration point, which allows us to only consider the matching direction collinear to the orientation of the intensity gradient in the block.

The equation (1) can be solved by

$$\frac{\partial W}{\partial U} = [K + H^T SH]U - H^T SD = 0 \qquad (3)$$

leading to the linear system

$$[K + H^T SH]U = H^T SD \quad (4)$$

The above approximation formulation performs well in the presence of outliers but suffers from a systematic error. Alternatively, solving the exact interpolation problem based on noisy data is not adequate. The robust solver can take advantage of both approximation and interpolation to iteratively estimate the deformation from the approximation to the interpolation while rejecting outliers. The gradual convergence to the interpolation solution is achieved through the use of an external force F added to the approximation formulation of Equation (4), which balances the internal mesh stress:

$$[K + H^T SH]U = H^T SD + F \quad (5)$$

This force F is computed at each iteration i to balance the mesh internal force KU_i, which leads to the iterative scheme:

$$\begin{aligned} F_i &\Leftarrow KU_i \\ U_{i+1} &\Leftarrow [K + H^T SH]^{-1}[H^T SD + F_i] \end{aligned} \quad (6)$$

3 ITK Implementation

The ITK implementation of the physics-based non-rigid registration method contains three filters: MaskFeaturePointSelection, BlockMatchingImageFilter, and FEMScatteredDataPointSetToImageFilter, which correspond to the above mentioned three components: feature point selection, block matching and robust finite element solver, respectively. These three filters can be used independently or connected together to perform non-rigid registration. Block Matching is parallelized using ITK v4 multithreading/GPU framework, for both multi-core and GPU, to accelerate the computation and gain some speedup. The robust solver is enhanced to allow the accommodation of different geometry elements in dealing with linear elastic problems by simply providing appropriate mesh. To implement non-rigid registration and achieve ease-of-use the three filters are combined into a single registration filter, PhysicsBasedNonRigidRegistrationMethod.

3.1 Feature point selection

MaskFeaturePointSelectionFilter (see Figure 2 for chart flow and Figure 3 for inheritance diagram) generates a list of feature points selected from a masked input image. It takes an Image and a mask Image as inputs and generates a PointSet of feature points as output. The feature points are physical centers of a small image blocks with higher variance. Optionally, a structure tensor is computed and stored as a pixel value for each feature point. The following optional parameters can be set:

- NonConnectivity: defines connectivity pattern (VERTEX_CONNECTIVITY, EDGE_CONNECTIVITY or FACE_CONNECTIVITY) to a feature point. The default is VERTEX_CONNECTIVITY;
- BlockRadius: radius measured in voxels over which the variance is computed, its default value is 1;

- SelectFraction – fraction of points to select out of total eligible points, default is 0.05.

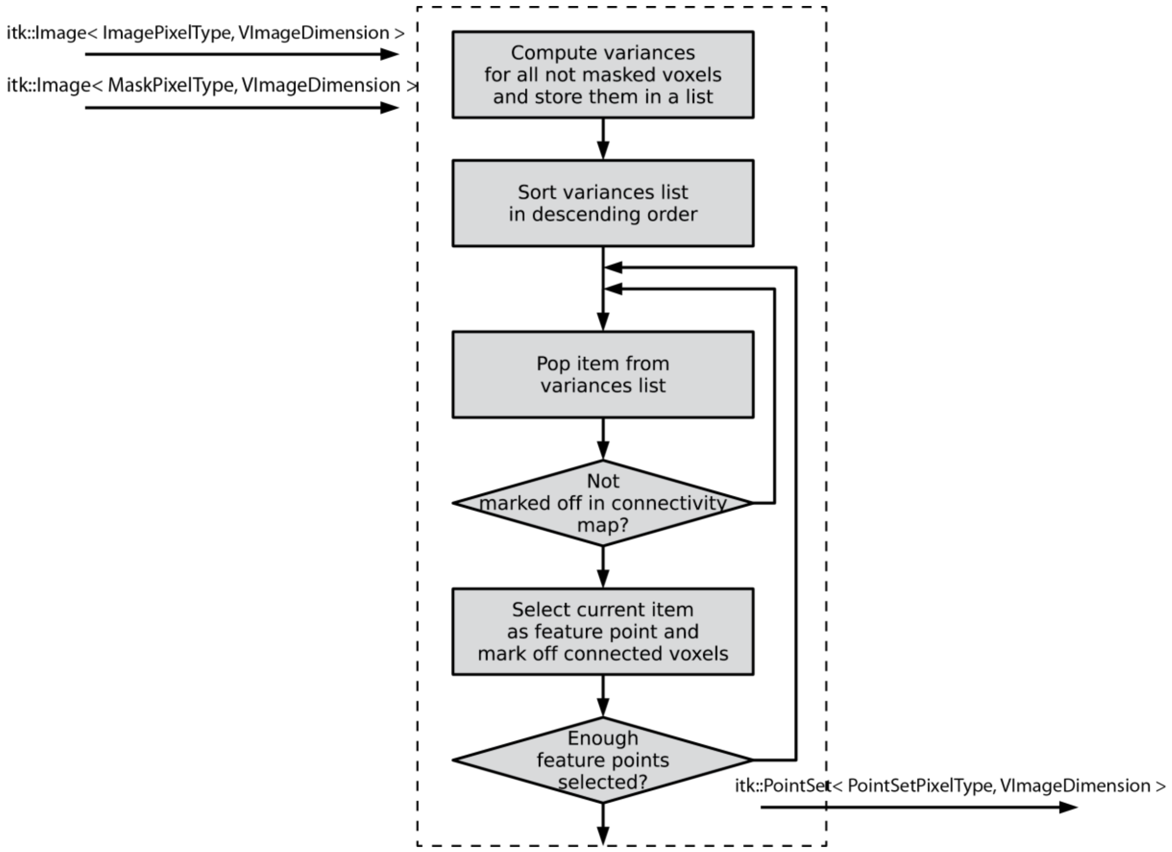

itk::Image< ImagePixelType, VImageDimension >
itk::Image< MaskPixelType, VImageDimension >
Compute variances for all not masked voxels and store them in a list
Sort variances list in descending order
Pop item from variances list
Not marked off in connectivity map?
Select current item as feature point and mark off connected voxels
Enough feature points selected?
itk::PointSet< PointSetPixelType, VImageDimension >

Figure 2. Flow chart of feature point selection

After the filter is created and inputs are set using SetInput and SetMaskImage, the calculation is triggered by calling Update method. After the Update, the method GetOutput returns a PointSet that contains coordinates of feature points as Point values and (optionally) structure tensors as Pixel values.

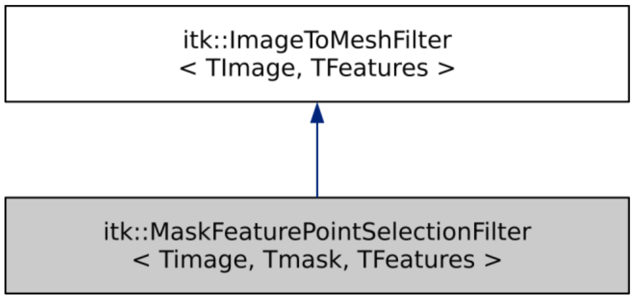

itk::ImageToMeshFilter
< TImage, TFeatures >
itk::MaskFeaturePointSelectionFilter
< Timage, Tmask, TFeatures >

Figure 3. Inheritance diagram of feature point selection

The following is the normal usage of the filter:

```
const unsigned int ImageDimension = 3;

typedef unsigned char                                                  InputPixelType;
typedef itk::Matrix< double, 3, 3 >                                    PointSetPixelType;
typedef itk::Image< InputPixelType,  ImageDimension >                  InputImageType;
typedef itk::PointSet< PointSetPixelType, 3 >                          PointSetType;
typedef itk::MaskFeaturePointSelectionFilter< InputImageType,
InputImageType, PointSetType >                                         FilterType;
typedef FilterType::PointType                                          PointType;

FilterType::Pointer filter = FilterType::New();
filter->SetInput( imageReader->GetOutput() );

filete->SetMaskImage( maskReader->GetOutput() );
filter->SetSelectFraction( 0.01 );

itk::Size< ImageDimension > BlockRadious;

BlockRadious.Fill(1);

filter->SetBlockRadius(BlockRadious);

filter->ComputeStructureTensorsOn();

try
  {
  filter->Update();
  }
catch ( itk::ExceptionObject &err )
  {
  std::cerr << err << std::endl;
  return EXIT_FAILURE;
  }

PointSetType::Pointer output = filter->GetOutput();
```

3.2 Block matching

BlockMatchingImageFilter (see Figure 4 and Figure 5 (left) for chart flow and Figure 5 (right) for inheritance diagram) computes displacements of given points from one image to another. It takes fixed and moving Images as well as a PointSet of feature points as inputs. Pixel values of input point set, i.e., the structural tensors are not used in this filter. The feature points are expected to lie at least SearchRadius + BlockRadius voxels from the image boundary. This is usually achieved by using an appropriate mask during selection of feature points. The default output (0) is a PointSet with displacement vector stored as the pixel value. Additional output (1) is a PointSet containing similarities, i.e., the NCC value. The number of points in the output PointSet is equal to the number of points in the input PointSet.

The following optional parameters can be set:

- BlockRadius: radius over which variance is computed, default is 1.
- SearchRadius: radius of the search window, default is 3.

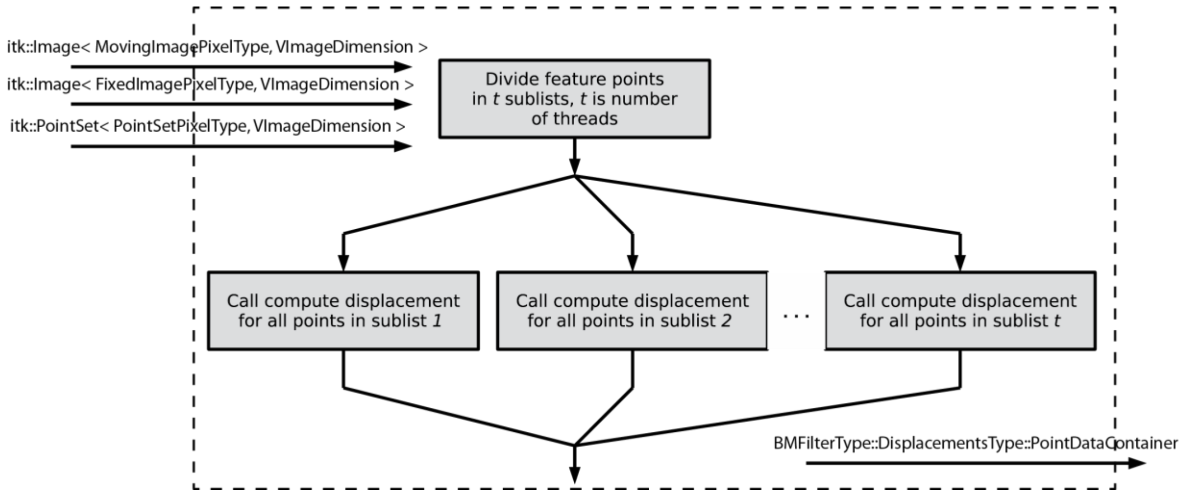
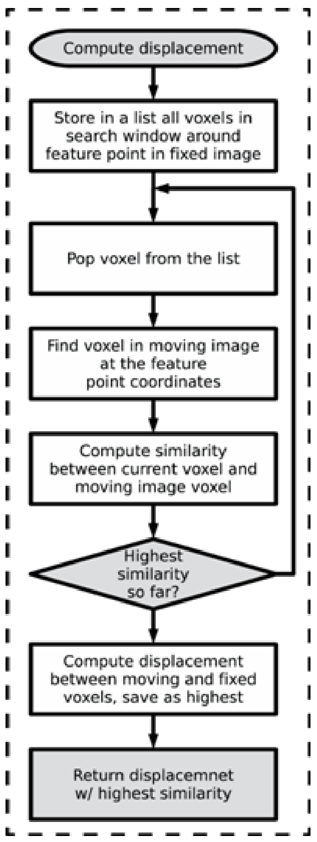

itk::Image< MovingImagePixelType, VImageDimension >
itk::Image< FixedImagePixelType, VImageDimension >
itk::PointSet< PointSetPixelType, VImageDimension >
Divide feature points in t sublists, t is number of threads
Call compute displacement for all points in sublist 1
Call compute displacement for all points in sublist 2
. . .
Call compute displacement for all points in sublist t
BMFilterType::DisplacementsType::PointDataContainer

Figure 4. Flow chart of block matching.

After the filter is created and inputs are set using SetFixedImage, SetMovingImage and SetFeaturePoints, the calculation is triggered by calling Update method. After update the method GetDisplacements returns a PointSet that contains coordinates of feature points as Point values and displacement vectors as Pixel values, GetSimilarities returns a PointSet that contains coordinates of feature points as Point values and similarity values as Pixel values.


Compute displacement
Store in a list all voxels in search window around feature point in fixed image
Pop voxel from the list
Find voxel in moving image at the feature point coordinates
Compute similarity between current voxel and moving image voxel
Highest similarity so far?
Compute displacement between moving and fixed voxels, save as highest
Return displacemnet w/ highest similarity

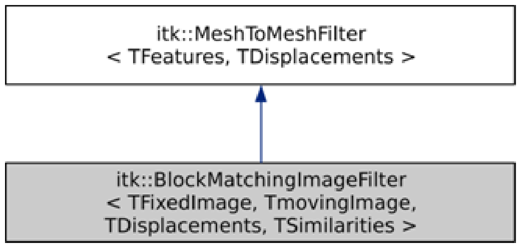

itk::MeshToMeshFilter
< TFeatures, TDisplacements >
itk::BlockMatchingImageFilter
< TFixedImage, TmovingImage,
TDisplacements, TSimilarities >

Figure 5. Left: flow chart of computing displacement component in Block Matching. Right: Inheritance diagram of Block Matching.

The following is the normal usage of the filter:

```
typedef unsigned  char                                        InputPixelType;
typedef itk::Image< InputPixelType,  3 >                      InputImageType;
typedef itk::BlockMatchingImageFilter< InputImageType >       BMFilterType;

BMFilterType::Pointer BMFilter = BMFilterType::New();
BMFilter->SetFixedImage( fixedImageReader->GetOutput() );
BMFilter->SetMovingImage( movingImageReader->GetOutput() );

BMFilter->SetFeaturePoints( featureSelectionFilter->GetOutput() );
BMFilter->SetBlockRadius( blockRadius );
BMFilter->SetSearchRadius( searchRadius );

try
  {
  BMFilter->Update();
  }
catch ( itk::ExceptionObject &err )
  {
  std::cerr << err << std::endl;
  return EXIT_FAILURE;

  }


BMFilterType::DisplacementsType::Pointer displacements = BMFilter->GetDisplacements();
BMFilterType::SimilaritiesType::Pointer similarities = BMFilter->GetSimilarities();
```

After Feature point selection and Block Matching, three point sets are available: feature point set with the structure tensor as the pixel value, block matching point set with the displacement as the pixel value, and the confidence point set with the similarity value as the pixel value. These three point sets will be used by the FEMScatteredDataPointSetToImageFilter to perform approximation and compute the displacement field everywhere.

3.3 Scattered data approximation

The class itk::fem::RobustSolver implements the solver presented in [1]. FEMScatteredDataPointSetToImageFilter is a wrapper of RobustSolver. FEMScatteredDataPointSetToImageFilter is used to facilitate the use of RobustSolver by converting natural inputs such as mesh and feature points into specific FEMObject, providing built-in 2D and 3D rectilinear meshes, invoking RobustSolver to resolve the solution to produce a deformed FEMObject, and converting the deformed FEMObject into a deformation field. RobustSolver takes a FEMObject as input, then iteratively approximates the data (displacement) associated with the feature points while rejecting outliers, and finally outputs a deformed FEMObject.

3.3.1 FEMScatteredDataPointSetToImageFilter

Figure 6 shows the flow chart and the inheritance diagram of this filter, respectively. FEMScatteredDataPointSetToImageFilter provides a built-in 2D quadrilateral and 3D hexahedron mesh if the input mesh is not available. Otherwise, just simply passes the input mesh to the converter. The natural inputs of the RobustSolver are mesh and point sets including mandatory feature points and optional confidence and tensor. itk FEM library requires a FEMObject as input. FEMScatteredDataPointSetToImageFilter converts the mesh and point sets into a FEMObject, which is undertaken by a member function:

```
InitializeFEMObject(FEMObjectType * femObject)

{
  this->InitializeMaterials(femObject);
  this->InitializeNodes(femObject);
  this->InitializeElements(femObject);
  this->InitializeLoads(femObject);

  // produce DOF
  femObject->FinalizeMesh();

}
```

The material properties of the biomechanical model such as Young modulus and Poisson's ratio are specified in InitializeMaterials. InitializeNodes and InitializeElements are used to store the nodes and the elements of the mesh into containers of the FEMObject. The displacement associated with the feature points are stored as loads in the FEMObject by InitializeLoads, in which the correspondence and tensor will be stored, too, if they are provided by users. After initialization, FinalizeMesh should be invoked to produce Degree of Freedoms (DoF) for the building of stiffness matrix K. After converting to FEMObject, RobustSolver is invoked to construct linear system of equations described by equation (6), resolve U of the linear system, and output a deformed FEMObject, which is used by DeformationFieldGenerator to produce a deformation field. The following codes show its typical usage.

```
const unsigned int ParametricDimension = 2;
const unsigned int DataDimension = 2;
typedef      short                                                  PixelType;
typedef      double                                                 RealType;
typedef itk::Image<PixelType, ParametricDimension>                  ImageType;
typedef itk::Vector<RealType, DataDimension>                        VectorType;
typedef itk::Matrix<RealType, DataDimension, DataDimension>         MatrixType;
typedef itk::Image<VectorType, ParametricDimension>                 DeformationFieldType;
typedef itk::PointSet <VectorType, ParametricDimension>             PointSetType;
typedef itk::PointSet <MatrixType, ParametricDimension>             TensorPointSetType;
typedef itk::PointSet <RealType, ParametricDimension>               ConfidencePointSetType;
typedef itk::Mesh< VectorType, ParametricDimension>                 MeshType;
typedef itk::FEMScatteredDataPointSetToImageFilter
<PointSetType, MeshType, DeformationFieldType,
ConfidencePointSetType, TensorPointSetType>                         FilterType;

FilterType::Pointer filter = FilterType::New();
PointSetType::Pointer featurePoints = PointSetType::New(); // feature points
associated with displacement

MeshType::Pointer mesh = MeshType::New(); // 2D triangle/rectilinear or 3D
tetrahedral/hexahedral mesh

ConfidencePointSetType::Pointer confidence = ConfidencePointSetType::New();
TensorPointSetType::Pointer tensor = TensorPointSetType::New();

filter->SetInput(featurePoints);
filter->SetConfidencePointSet(confidence); //optional
filter->SetTensorPointSet(tensor); //optional
filter->SetMesh(mesh); // optional

filter->Updata();

DeformationFieldType::Pointer field = filter->GetOutput();
```

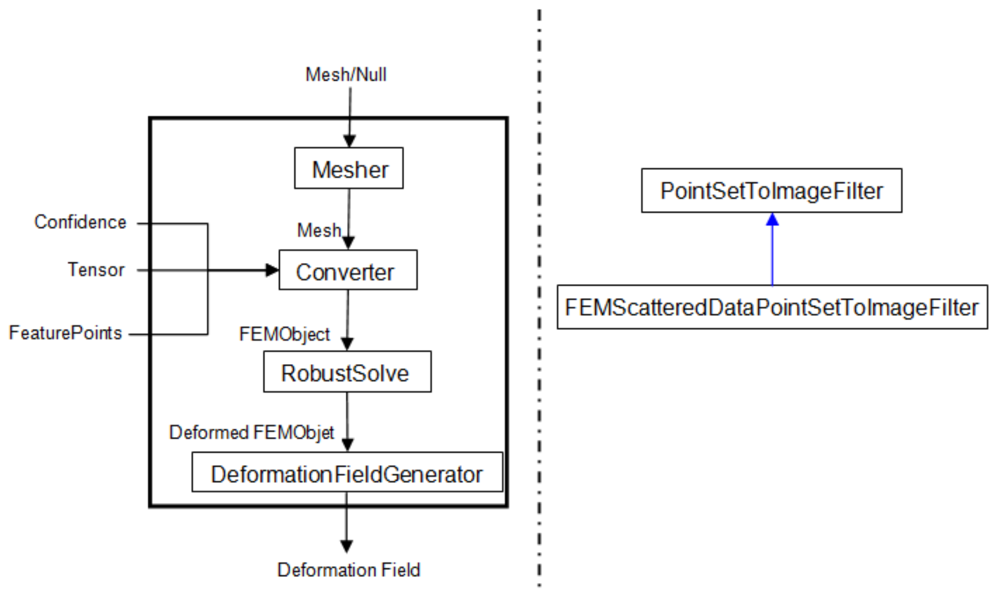


Figure 6. The flow chart (left) and the inheritance diagram (right) of FEMScatteredDataPointSetToImageFilter. FEMScatteredDataPointSetToImageFilter takes mesh, feature points, confidence and structural tensor as inputs. Converter first converts these inputs into a FEMObject, and then invokes RobustSolver to produce a deformed FEMObject. This deformed Object is converted into the deformation filed by DeformationFildGenerator.

3.3.2 RobustSolver

Given a 2- or 3-D scattered and noisy point set, in which each point is associated with a 2-D or 3-D displacement, RobustSolver is able to approximate the data while rejecting outliers, advance toward interpolation, and finally output a deformed FEMObject. The flow chart and inheritance diagram are described in Figure 7 and Figure 8, respectively.

RobustSolver also takes into account two optional point sets: the confidence and structural tensor. Confidence point set describes our confidence for each feature point using a value between 0 and 1 (0: not trustful, 1: completely trustful), which will make the solver behavior like a weighted Least Square. Tensor point set describes the distribution of the edge direction within a small block surrounding the feature point, which is used to avoid the aperture problem [6, 7]. The following codes show the typical usage of the RobustSolver.

```
typedef itk::fem::FEMObject<2>     FEMObjectType;

FEMObjectType::Pointer underformedFEMObject = FEMObjectType::New();

// initialize underformedFEMObject  here or use FEMScatteredDataPointSetToImageFilter,
which will undertake the initialization.

typedef itk::fem::RobustSolver<2>    FEMSolverType;

FEMSolverType::Pointer solver = FEMSolverType::New();
```

```
solver->SetInput(underformedFEMObject);

solver->Update( );

FEMObjectType::Pointer deformedFEMObject = solver->GetOutput( );
```

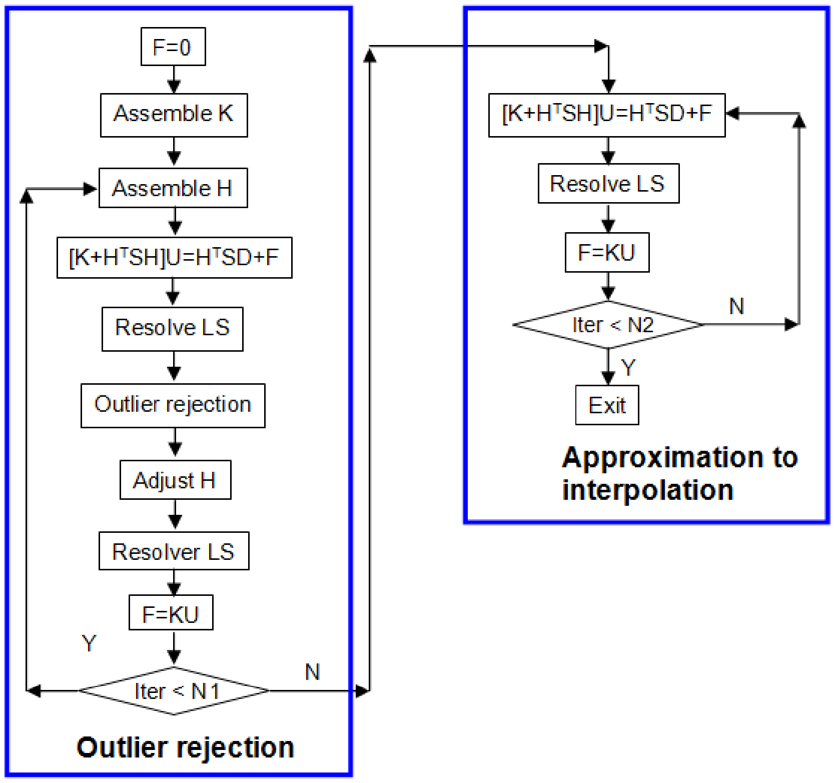


Figure 7. The flow chart of RobustSolver. RobustSolver includes two parts: outlier rejection and approximation to interpolation. Outlier rejection proceeds as a LTS regression [5]: resolve U first, then detect outliers, remove outliers and resolve U again. The F is used to reset the strain energy to enable the mesh to be deformed further. The difference between the two parts is there is no outlier rejection in the approximation to interpolation part.

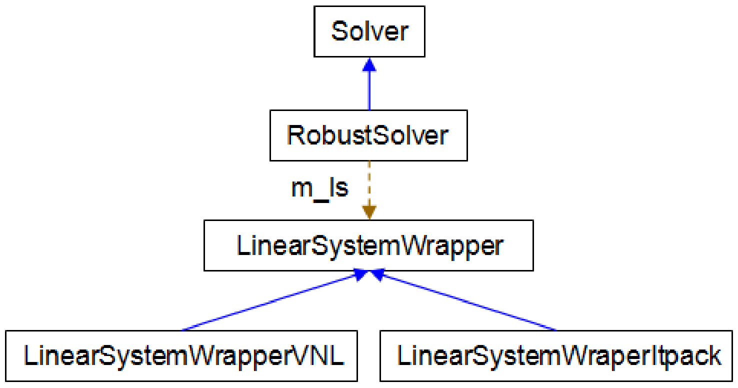


Figure 8. The inheritance diagram of RobustSolver. RobustSolver supports both VNL solver and Itpack solver to resolve the linear system of equations. Compared to VNL solver, Itpacks runs faster, which is the default LS solver in RobustSolver.

3.4 Main registration filter

PhysicsBasedNonRigidRegistrationMethod is the filter, which connects the MaskFeaturePointSelection, BlockMatchingImageFilter, and FEMScatteredDataPointSetToImageFilter into one pipeline to perform non-rigid registration as shown in the following codes:

```
template <class TFixedImage, class TMovingImage, class TMaskImage, class TMesh, class
TDeformationField>
void
PhysicsBasedNonRigidRegistrationMethod<TFixedImage, TMovingImage, TMaskImage, TMesh,
TDeformationField>
::GenerateData()
{
  // feature selection
  this->m_FeatureSelectionFilter->SetInput( this->GetMovingImage() );
  this->m_FeatureSelectionFilter->SetMaskImage( this->GetMaskImage() );
  this->m_FeatureSelectionFilter->SetSelectFraction( this->m_SelectFraction );
  this->m_FeatureSelectionFilter->SetNonConnectivity( this->m_NonConnectivity );
  this->m_FeatureSelectionFilter->SetBlockRadius( this->m_BlockRadius );

  // block matching
  this->m_BlockMatchingFilter->SetFixedImage( this->GetFixedImage() );
  this->m_BlockMatchingFilter->SetMovingImage( this->GetMovingImage() );
  this->m_BlockMatchingFilter->SetBlockRadius( this->m_BlockRadius );
  this->m_BlockMatchingFilter->SetSearchRadius( this->m_SearchRadius );

  // assembly and solver
  typename BlockMatchingFilterType::DisplacementsType * displacements =
  this->m_BlockMatchingFilter->GetDisplacements();

  this->m_FEMFilter->SetInput( displacements );
  this->m_FEMFilter->SetMesh( const_cast< MeshType * >( this->GetMesh() ) );

  const FixedImageType * fixedImage = this->GetFixedImage();
  this->m_FEMFilter->SetSpacing( fixedImage->GetSpacing() );
  this->m_FEMFilter->SetOrigin( fixedImage->GetOrigin() );
  this->m_FEMFilter->SetSize( fixedImage->GetLargestPossibleRegion().GetSize() );

  typename FEMFilterType::FEMSolverType * femSolver =
  this->m_FEMFilter->GetFEMSolver();

  femSolver->SetApproximationSteps( this->m_ApproximationSteps );
  femSolver->SetOutlierRejectionSteps( this->m_OutlierRejectionSteps );

  this->m_FEMFilter->Update();
}
```

The output of the PhysicsBasedNonRigidRegistrationMethod is a deformation filed.

The following codes shows how to use the PhysicsBasedNonRigidRegistrationMethod:

Command : ./PBNRR FixedImage MovingImage MaskImage Mesh WarpedImage

Test Case Code:

```
int itkPhysicsBasedNonRigidRegistrationMethodTest(int argc, char *argv[] )
{
  if ( argc < 6)
    {
    std::cerr << "Five arguments are required :"<< std::endl;
    std::cerr <<" FixedImage, MovingImage, MaskImage, Mesh, WarpedImage" << std::endl;
    return EXIT_FAILURE;
    }

  enum { FIXED_IMG = 1, MOVING_IMG, MASK_IMG, MESH, WARPED_IMG ,CANNY_VALID };

  const unsigned int ImageDimension = 3;
  typedef float                                           InputPixelType;
  typedef itk::Image< InputPixelType, ImageDimension >  InputImageType;
  typedef itk::Mesh< float, ImageDimension >             MeshType;

  // read fixed image
  typedef itk::ImageFileReader< InputImageType >         ImageReaderType;

  ImageReaderType::Pointer readerFixed = ImageReaderType::New();

  readerFixed->SetFileName( argv[FIXED_IMG] );

  // read moving image
  ImageReaderType::Pointer readerMoving = ImageReaderType::New();
  readerMoving->SetFileName( argv[MOVING_IMG] );

  // read mask image
  ImageReaderType::Pointer readerMask = ImageReaderType::New();
  readerMask->SetFileName( argv[MASK_IMG] );

  // read mesh
  typedef itk::VTKTetrahedralMeshReader< MeshType >   MeshReaderType;
  MeshReaderType::Pointer readerMesh = MeshReaderType::New();
  readerMesh->SetFileName( argv[MESH] );

  // update the readers
  try
    {
    readerFixed->Update();
    readerMoving->Update();
    readerMask->Update();
    readerMesh->Update();
    }
  catch( itk::ExceptionObject & e )
    {
    std::cerr << "Error while reading inputs: " << std::endl;
    std::cerr << e << std::endl;
    return EXIT_FAILURE;
    }

  // Create the NRR filter and set the input
  PBNRRFilterType::Pointer filter = PBNRRFilterType::New();

  filter->SetFixedImage( readerFixed->GetOutput() );
  filter->SetMovingImage( readerMoving->GetOutput() );
  filter->SetMaskImage( readerMask->GetOutput() );
  filter->SetMesh( readerMesh->GetOutput() );

  itk::Size< ImageDimension > BlockRadious;
  BlockRadious.Fill(1);
  filter->SetBlockRadius(BlockRadious);

  itk::Size< ImageDimension > SearchRadious;
  SearchRadious.Fill(5);
  filter->SetSearchRadius(SearchRadious);
  filter->SetApproximationSteps(10);
  filter->SetOutlierRejectionSteps(10);
  filter->SetSelectFraction( 0.05 );
  std::cout << "Filter: " << filter << std::endl;

  // Update the PBNRR filter
  try
    {
    filter->Update();
    }
  catch( itk::ExceptionObject & e )
    {
    std::cerr << "Error during filter->Update(): " << e << std::endl;
    return EXIT_FAILURE;
    }

   // Create - Write ITK deformed image
   InputImageType::Pointer deformedImage;
  filter->CreateDeformedImage(deformedImage);

  std::cout << "Save Deformed Image at  : " << argv[WARPED_IMG] << std::endl;
  typedef itk::ImageFileWriter<InputImageType> WriterType;
  typename WriterType::Pointer deformedImageWriter = WriterType::New();
  deformedImageWriter->SetFileName(argv[WARPED_IMG]);
  deformedImageWriter->SetInput(deformedImage);
  deformedImageWriter->Update();

  return EXIT_SUCCESS;
}
```

4 Experiments and Results

We conducted experiments on the registration between preoperative MRI and the intra-operative MRI (iMRI). The five datasets come from public cases from SPL of Harvard medical school [12]. Table I lists the patient information including the gender, tumor location, and histopathology.

Table 1. Patient information of five cases from SPL of Harvard medical school.

Case#	Gender	Tumor location	Histopathology
1	F	R occipital	Anaplastic Oligodendroglioma WHO III/IV
2	F	L posterior temporal	Glioblastoma WHO IV
3		R frontal	Oligodendroglioma WHO II/IV
4		R occipital	
5	F	R frontal	Oligoastrocytoma WHO II/IV

The MRI of the 5 public cases were acquired with a protocol: whole brain sagittal 3D-SPGR (slice thickness 1.3 mm, TE/TR=6/35 ms, FA=75^o, FOV=24 cm, matrix=256 x 256) [11].

In Table 2 we show the qualitative results of the PBNRR filter for the 5 cases which run in a workstation equipped with an Intel® Core™ i7 CPU 260 @ 2.80 GHz with 8 GiB of RAM.

As a measure of the registration accuracy we use the one directional Hausdorff Distance (HD) as it is implemented in the vtkHausdorffDistancePointSetFilter. The HD(1→2) before PBNRR corresponds to the error between canny points in pre-operative MRI and intra-operative MRI while the HD(1→2) after PBNRR corresponds to the error between canny points in warped pre-operative MRI and intra-operative MRI. The running time includes the time for the PBNRR filter and the time for creating and writing the warped pre-operative MRI, not including the time for generating the canny points and the calculation of the HD.

Table 2. The quantitative results for the 5 cases are obtained by running PBNRR filter using 8 threads . The parameters for all cases are: Block radius : [1,1,1] , Window radius : [5,5,5], Selection fraction : 0.05, Rejection fraction : 0.25, Num of outlier rejection steps : 10 , Num of approximation steps : 10

Case	HD (1→2) before PBNRR (mm)	HD (1→2) after PBNRR (mm)	HD (1→2) improvement	Num Registration Points	Num rejected Outliers	Num nodes	Num elements	Running time (sec)
1	25.980	20.099	0.226	69244	17310	8109	41646	57.64
2	9.110	4.690	0.485	76821	19200	8843	45719	65.88
3	9.433	5.385	0.429	68745	17180	7984	41081	54.26
4	9.695	7.000	0.278	84445	21110	9512	49106	67.94
5	6.708	4.123	0.385	68225	17050	7935	40789	54.92

In table 3 we show the running time for each component of the PBNRR filter for the case 4. The experiment has run for different number of threads (1, 12, 24 and GPU Quadro 6000 with 448 cores).

Table 3: Running time (sec) for PBNRR filter for case 4. The parameters are: Block radius : [1,1,1], Window radious : [5,5,5], Selection fraction : 0.05, Rejection fraction : 0.25, Num of outlier rejection steps : 10 , Num of approximation steps : 10 , Young modulus = 694 Pa, Poisson's ratio = 0.45.

Component		Num of Threads			
		1	12	24	GPU (Quadro 6000)
Feature Selection		5.32	5.34	5.32	5.32
Block Matching		**20.42**	**2.63**	**1.71**	**0.38**
Solver	Initialize FEM Object	0.39	0.43	0.44	0.44
	Initialize Matrices	1.54	1.54	1.54	1.54
	System Solution	13.00	13.15	13.08	13.09
	Copy Input to Output	15.78	15.82	15.47	15.47
	Produce Deformation Field	2.12	2.29	2.27	2.26
	Total Time for Solver	32.83	33.23	32.80	32.80
Total Time for ITK v4 PBNRR filter		**58.57**	**41.20**	**39.83**	**38.50**
Additional operation to Create and Write Deformed Image		11.01	11.39	11.36	11.37
Total time for PBNRR and Deforming Pre-op MRI		**69.58**	**52.59**	**51.19**	**49.87**

In Figure 9 we present the quantitative results of the PBNRR filter for the same 5 cases we used throught this evaluation. To reproduce the results, users need to checkout the master branch and the patch with topic "A2D2PBNRR" if they use the version less than ITK4.3.

5 Conclusion

We present an ITK implementation of a physics-based non-rigid registration method. The three filters: MaskFeaturePointSelection, BlockMatchingImageFilter, and FEMScatteredDataPointSetToImageFilter can be used separately or combined together to do registration. MaskFeaturePointSelection identifies small blocks with rich structure information. BlockMatchingImageFilter is used to find the displacement associated with these blocks in order to produce a sparse deformation field, which is used by FEMScatteredDataPointSetToImageFilter to interpolate the entire deformation field. For each block MaskFeaturePointSelection stores the structure tensor, and BlockMatchingImageFilter stores the confidence, i.e., the cross-correlation value. Both structure tensor and confidence are incorporated into the FEMScatteredDataPointSetToImageFilter to deal with aperture problem and weight the least square formula. To reduce the computational time of block matching, it is parallelized on multi-core and GPU, and the execution time is reduced from about 70 secs to 50 secs i.e., only 30% improvement since there are many other sequential parts in the PBNRR filter. The experiments on five brain MRI data demonstrate the effectiveness of the non-rigid registration method.

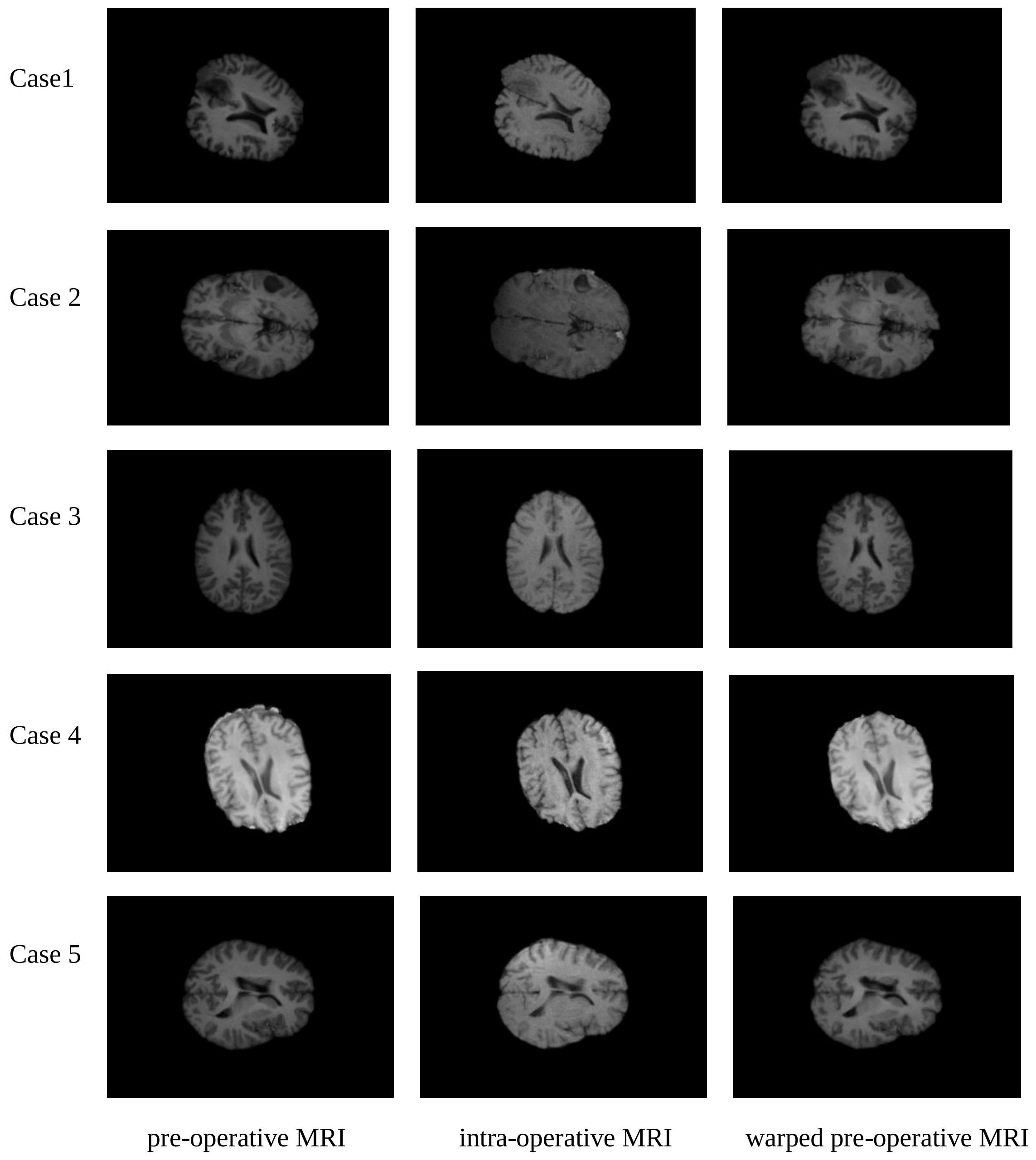


Figure 9. The Qualitative results for the 5 cases of the PBNRR filter. Each row corresponds to a different case, and each column from left to right: the pre-operative MRI , the intra-operative MRI and the warped pre-operative MRI.

6. Future work

In the future we plan to provide a web-service for image-to-mesh conversion to generate over the WEB the mesh of the images. This service can maintain new functionality as we better understand the needs of the ITK community. However, for the near future we can help as many ITK users as we can with the finite element mesh generation of the images that might want to use in conjunction of this filter, since we strongly suggest users to provide an anatomically adapted mesh as input for the physics-based non-rigid registration although a default rectilinear mesh is provided inside.

7. Acknowledgements

This work is funded mainly by the ARRA funds for the ITK-v4 implementation with grant number: NLM A2D2 201000586P. In addition, this work is supported in part by NSF grants: CCF-1139864, CCF-1136538, and CSI-1136536 and by the John Simon Guggenheim Foundation and the Richard T. Cheng Endowment. We would like to thank Luis Ibanez, Matt McCormick, and Arnaud Gelas for their help on ITK coding style. Also, we would like to thank the image-to-mesh conversion team at CRTC, especially Panos Foteinos for providing the Finite Element Meshes for the evaluation of the software and Dr. Michele Audette at ODU for bringing this opportunity to our attention while he was at Kitware. The staff of W&M at Grants Office have been extremely helpful with the transition of this grant from W&M to ODU. At last but not least Dr. Terry Yoo and his staff at NLM, for his constant support and understanding to accommodate us with all shorts of issues created due to our transition to ODU.

References

[1] O. Clatz, H. Delingette, I.-F. Talos, A. Golby, R. Kikinis, F. Jolesz, N. Ayache, and S. Warfield, "Robust non-rigid registration to capture brain shift from intra-operative MRI", IEEE Trans. Med. Imag., 24(11);1417-27, 2005.

[2] Nikos Chrisochoides, Andriy Fedorov, Andriy Kot, Neculai Archip, Peter Black, Olivier Clatz, Alexandra Golby, Ron Kikinis and Simon K. Warfield, “Toward Real-Time, Image Guided Neurosurgery Using Distributed and Grid Computing”, IEEE/ACM International Conference for High Performance Computing, Networking, Storage and Analysis (Supercomputing), 2006.

[3] Yixun Liu, Andriy Fedorov, Ron Kikinis and Nikos Chrisochoides, “Real-time Non-rigid Registration of Medical Images on a Cooperative Parallel Architecture”, IEEE International Conference on Bioinformatics & Biomedicine, pages 401 -- 404, November, 2009.

[4] K. Bathe, “Finite Element Procedure”, Prentice-Hall, 1996.

[5] P. J. Rousseeuw and A. M. Leroy, “Robust regression and outlier detection”, John Wiley & Sons, Inc., New York, NY, USA, 1987.

[6] S. Shimojo, G. H. Silverman, and K. Nakayama, “Occlusion and the solution to the aperture problem for motion”, Vision Research, 29, 619-626, 1989

[7] T. Poggio, V. Torre, and C. Koch, “Computational vision and regularization theory,” Nature, vol. 317, pp. 314–319, Oct. 1985.

[8] M. Bierling, Displacement estimation by hierarchical block matching, Proc. SPIE Vis. Comm. and Image Proc. 1001 (1988) 942951.

[9] X. Yuan, X. Shen, Block matching algorithm based on particle swarm optimization for motion estimation, in: ICESS '08: Proceedings of the 2008 International Conference on Embedded Software and Systems, IEEE Computer Society, Washington, DC, USA, 2008, pp. 191-195.

[10] M. Stefano, T. Federico, S. L. Di, P. Marco, Efficient and optimal block matching for motion estimation, in: ICIAP '07: Proceedings of the 14th International Conference on Image Analysis and Processing, IEEE Computer Society, Washington, DC, USA, 2007, pp. 705-710.

[11] N. Archip, O. Clatz, A. Fedorov, A. Kot, S. Whalen, D. Kacher, N. Chrisochoides, F. Jolesz, A. Golby, P. Black, and S. K. Warfield, “Non-rigid alignment of preoperative MRI, fMRI, DT-MRI, with intraoperative MRI for enchanced visualization and navigation in image guided neurosurgery,” Neuroimage, vol. 35(2), pp. 609–624, 2007.

[12] I.-F. Talos and N. Archip, “Volumetric non-rigid registration for MRI guided brain tumor surgery,” 08 2007.