Theses and Dissertations

Summer 2011

Software architecture of the non-rigid image registration evaluation project

Jeffrey Allan Hawley
University of Iowa

Recommended Citation

Hawley, Jeffrey Allan. "Software architecture of the non-rigid image registration evaluation project." MS (Master of Science) thesis, University of Iowa, 2011.
http://ir.uiowa.edu/etd/1229.

SOFTWARE ARCHITECTURE OF THE NON-RIGID IMAGE REGISTRATION

EVALUATION PROJECT

by

Jeffrey Allan Hawley

A thesis submitted in partial fulfillment of the
requirements for the Master of Science
degree in Electrical and Computer Engineering
in the Graduate College of
The University of Iowa

July 2011

Thesis Supervisor: Professor Gary E. Christensen

Graduate College
The University of Iowa
Iowa City, Iowa

CERTIFICATE OF APPROVAL

_____

MASTER'S THESIS

_____

This is to certify that the Master's thesis of

Jeffrey Allan Hawley

has been approved by the Examining Committee for the
thesis requirement for the Master of Science degree in
Electrical and Computer Engineering at the July 2011
graduation.

Thesis Committee: _____
                   Gary E. Christensen, Thesis Supervisor

                   _____
                   Hans Johnson

                   _____
                   Jon Kuhl

ACKNOWLEDGEMENTS

TABLE OF CONTENTS

LIST OF FIGURES

Figure

CHAPTER 1
INTRODUCTION

In medical image registration the goal is to find point by point correspondences between a source image and a target image such that the two images are aligned. There are rigid and non-rigid registration algorithms. Rigid registration uses rigid transformation methods which preserve distances between every pair of points. Non-rigid registration uses transformation methods that do not have to preserve the distances. Image registration has many medical applications -tracking tumors, anatomical changes over time, differences between characteristics like age and gender, etc. A gold standard transformation to compare and evaluate the registration algorithms would be ideal to use to verify if the two images are perfectly aligned. However, there is hardly if ever a gold standard transformation for non-rigid registration algorithms. The reason why there is no gold standard transformation for non-rigid registration algorithms is that pointwise correspondence between two registered points is not unique. In the absence of a gold standard various evaluation methods are used to gauge registration performance. However, each evaluation method only evalutes the error in the transformation from a limited perspective and therefore has its advantages and drawbacks. The Non-Rigid Image Registration Evaluation Project (NIREP) was was created to provide one central tool that has a collection of evaluation methods to perform the evaluations on non-rigid image registration algorithms and rank the registration algorithms based on the outputs of the evaluation methods

in the absence of without having to use a gold standard. NIREP utilizes the following examples of evaluations of both rigid and non-rigid image registrations done over the years.

In the "Evaluation of 14 nonlinear deformation algorithms applied to human brain MRI registration"[11] study the author, Arno Klein, evaluated 14 nonlinear deformation algorithms applied to T1-weighted MRI brain image's using 8 different error measures. The authors started the study by performing the 14 algorithms over the brain images. Then they applied the 8 different error measures on the outputs of the deformation algorithms. Finally, they ranked the results of the evaluations based on permutation tests, one-way ANOVA tests and indifference-zone ranking.

The study only used one modality, T1-weighted MRI, brain data but there are other modalities such as PET, CT, T2-weighted MRI, PD-weighted MRI, etc. NIREP will provide a framework to support multiple modalities. They also did comparison of labels, which assumed that the "manual labels sets are correct, or 'silver standards'." Manual labels have an inherent human error which if they are used as a standard, relegates them to a silver standard and not a "gold standard". NIREP will support labels and the corresponding evaluation methods. As well, they left out some algorithms partly because using them requires considerable knowledge of the software, partly because some were semi-automated approaches that require even minimal intervention to reduce bias. NIREP will have users perform their own registration algorithms outside of NIREP and then use NIREP to do the evaluation. Finally, they showed three ways to rank registration algorithms based on the evalu-

ation methods. NIREP will rank registration algorithms and can use the study as a reference. The study show's how an evaluation can be done for non-rigid image algorithms and that there is a need for an automated program to evaluate and rank non-rigid image registration algorithms.

In "Comparison and evaluation of retrospective intermodality brain image registration techniques." study [14] the author, Jay West under the supervision of Michael Fitzpatrick, tried a blind test of rigid deformation algorithms to see the error of the algorithms against a "gold standard" based on fiducial markers. They came up with an evaluation dataset that consisted of CT, MR and PET image volumes. Each of the image's were taken of patients with bone-implanted fiducial markers. During the processing of the images, the fiducial markers were air brushed out and then given to collaborates outside of Vanderbilt to perform retrospective registrations. The collaborates would run their algorithms to transform the CT to MR and/or from PET to MR images. The investigators gave the transformations back to Vanderbilt which then measured them for accuracy. The accuracy was measured at multiple "volumes of interest" i.e., areas in the brain that would commonly be areas of neurological interest. The volumes of interest would then be used with the Target Registration Error (TRE) evaluation algorithm to find out how well the transformations worked.

The Vanderbilt study only evaluated rigid registrations. NIREP will evaluate non-rigid registrations. The study used one evaluation algorithm. NIREP will use as many evaluation algorithms as possible. The NIREP project discussed in this work is designed to extend the Vanderbilt study to evaluate non-rigid registration

algorithms. The study had people run their own algorithms on a data set and then submit the transformations. NIREP will do the same thing of having users run their own algorithms on a data set and then submit the transformations. The study shows how evaluating rigid registration can be performed which is a starting point for NIREP.

Since there is no gold standard, it has been proposed to use a less vigorous standard called the bronze-standard [9]. The bronze-standard uses n images and m methods to register the images to try to converge towards the "perfect" registration. To get better results requires m and/or n to increase. Also, the methods are required to be independent so as to reduce the bias. To try and combat the massive amount of data and computation, the author, Tristan Glatard, suggested using a grid infrastructure of computers to do the work. The experiments used both the EGEE production grid which has 18,000 CPU's and 5 PB storage capacity and the Grid5000 experimental infrastructure which has 2,000 CPU's and hundreds of GB storage capacity. The experiment also used 110 patients, who got imaged numerous times, and 4 different registration algorithms.

The study addressed the problem of storage of large data. Medical images are large and the bronze standard requires a large amount of images. NIREP will have to take care of inputting images into the program but the storage of the data is up to the user. The study took care of an issue of a very computationally intensive problem of finding the bronze-standard. NIREP can reference the study when the computation of evaluations reaches a very computationally intensive problem. There is going to

be lots of data and algorithms to use for NIREP and this study shows a solution to combat the large data and algorithms and very computationally intensive problem.

The Analyze software is the "Mayo Clinic's comprehensive biomedical imaging software suite." The software displays both 2-D and 3-D views. One of the views is a 3x3 grid that fuses two images, from column one and three, into one image, in column 2. There is an additional package that computes some Evaluation Methods between two images and even between object maps. The software is a closed source exclusively developed by Mayo Clinic. In having a closes source, people can not see the method algorithms used for evaluation.

"Slicer, or 3D Slicer, is a free, open source software package for visualization and image analysis." The program has lots of features in part because of the plug-in structure. Slicer is highly customizable to different user choices with plug-in's. The program displays both 2-D and 3-D views. As well, users can perform some registration's (or transformations).

Vv is a viewer of 4D images. The software uses 4 panels to display slices of an image, all of the panels are linked. Since the program is only a viewer, only qualitative evaluations can be performed, such as fusing images, using landmarks and superimposing vector fields. Finally, the program is open source, based on VTK, ITK and QT.

The VALMET software is used for measuring and visualizing the differences between corresponding object segmentations. However the program has received only one minor update since 2001. The different methods used for evaluation are: relative

overlap, Hausdorff distance, surface distance and probabilistic overlap.

The Amide software does multimodality image analysis. The analysis is computed using statistics from volumetric regions of interest (ROIs). As well, the program allows users to perform rigid body registration. Finally the program is open source and cross platform.

The OsiriX software display's multidimensional images for interpretation without the need for high-end expensive hardware or software. Some data today are very large, something like 800 to 1,000 slices, that need to be interpreted. With large data, high-end 3D workstations were required to view the images. The program is designed to take care of the large data by a "streaming" technique. Users are allowed to customize the program with different functions and tools. The only inputs allowed are DICOM. Finally the program is open source but designed only for Macintosh.

However, all of these studies and tools are not a full evaluation of non-rigid image registration algorithms. A full evaluation of the registration algorithms will need to set up an evaluation database of differing image types. Then different registration algorithms will use the evaluation database to create transformations. From there, as many evaluation methods as possible will be performed on the transformations and evaluation database. Once that happens, then the results of the evaluations need to be saved and a ranking system created. When the ranking system is created, the results need to be shared with the community.

The NIREP is both an evaluation database and a software tool for performing a full evaluation of non-rigid image registration algorithms. The software tool has

been developed to: read in the evaluation database and transformations, evaluate registration algorithms by bringing together a collection of evaluation methods, letting the user decide which evaluations are useful, displaying both the visualization and numerical results and allow the user to save the evaluation results and upload them to a website to be shared.

CHAPTER 2
NIREP DESIGN

The NIREP will help evaluate Non-rigid Image Registration Algorithms. NIREP is a downloadable program and a downloadable evaluation database. Users will be responsible for generating their own transformations, using the evaluation database, using as many algorithms as they want to compare. The NIREP software will read in these transformations and an evaluation database, compute the evaluations and visually and numerically compare the algorithm performance. The evaluation database has to be common so that fair comparisons are made between algorithms.

2.1   Requirements

2.1.1   Program Input

2.1.1.1   Evaluation Database

Currently there is no common database used for evaluations. The common reference evaluation database will be used as a consistent measuring ruler for all Non-rigid Image Registration algorithms. In having a consistent measuring ruler, the amount of variables that are different between evaluations are kept to a minimum. The evaluation database can consist of images, segmentations and contours.

2.1.1.2   Transformations

Users will use the evaluation database to create transformation data, images or coefficients, that will be stored on disk. The transformation data is a mapping

of one coordinate system to another coordinate system. The program has to read in the transformations and use the data to transform the evaluation data into deformed data. The evaluation algorithms will then use the deformed data to perform the evaluations. The program should have a framework to be able to add in new modules for different transformation readers and generators.

2.1.1.3   Organizing Data on Disk

Both the evaluation database and transformation data will be stored on disk and required to be organized. The organization will require a file to store the location of the data, the type of the data, and if the user wants, comments about the data. All of the data described in the file will be considered to be part of a unique dataset. This means that the evaluation database will have one unique file and the transformations will have another file.

2.1.1.4   Organizing Data in Memory

The data will be read in from disk and stored in memory. The storage of the data will need to be organized for a fast look up. The reason for the fast look up is that different evaluation algorithms and visualization and numerical views will require the data.

2.1.2   Evaluation Methods

There is no one evaluation algorithm that is a "gold standard". The program needs to bring together as many evaluation algorithms as possible. The following evaluation algorithms (2.1.6.1-2.1.6.10) will be in the first version of NIREP and used

to create a frame work for future evaluation algorithms.

2.1.2.1 Difference

The Difference shows where the intensities in two images are not the same. In equation form the difference is defined as

$$I_D(x) = |I_1(x) - I_2(x)| \qquad (2.1)$$

The difference computed for each ROI is defined as

$$I_{Di} = \sum_{x \in ROI_i} |I_{1i}(x) - I_{2i}(x)| \qquad (2.2)$$

The maximum value for each ROI is defined as

$$Max_{Di} = \max_{x \in ROI_i} |I_{1i}(x) - I_{2i}(x)| \qquad (2.3)$$

When computing the values for each ROI of other evaluation algorithms, 2.2 will be slightly modified in that $|I_1 i(x) - I_2 i(x)|$ will be replaced with whatever algorithm is used. When computing the maximum value for each ROI of other evaluation algorithms, 2.3 will be slightly modified in that $|I_1 i(x) - I_2 i(x)|$ will be replaced with whatever algorithm is used.

2.1.2.2   Jacobian

The Jacobian determinant of a transformation measures local expansion/contraction

and detects local singularity [4, 2]. In equation form the Jacobian determinant of

transformation $h(x) = x + u(x)$ in 2-D is defined as

$$J2(h(x)) = \begin{vmatrix} \frac{\partial u_1(x)}{\partial x_1} & \frac{\partial u_2(x)}{\partial x_1} \\[2mm] \frac{\partial u_1(x)}{\partial x_2} & \frac{\partial u_2(x)}{\partial x_2} \end{vmatrix}, \tag{2.4}$$

where u_1 and u_2 are two components of the displacement vector at location $x =$

$[x_1, x_2]$; and in 3-D is defined as

$$J3(h(x)) = \begin{vmatrix} \frac{\partial u_1(x)}{\partial x_1} & \frac{\partial u_2(x)}{\partial x_1} & \frac{\partial u_3(x)}{\partial x_1} \\[2mm] \frac{\partial u_1(x)}{\partial x_2} & \frac{\partial u_2(x)}{\partial x_2} & \frac{\partial u_3(x)}{\partial x_2} \\[2mm] \frac{\partial u_1(x)}{\partial x_3} & \frac{\partial u_2(x)}{\partial x_3} & \frac{\partial u_3(x)}{\partial x_3} \end{vmatrix}, \tag{2.5}$$

where u_1, u_2 and u_3 are three components of the displacement vector at location

$x = [x_1, x_2, x_3]$. The scale of the Jacobian determinant image intensity can be adjusted

to a log-scale.

The output will be both an image and table of numbers. The table of numbers

will hold the maximum value found, the minimum value found and the average values.

The table may be split up into region of interests, which means that the maximum,

minimum and average will be for each of the region of interests instead of over the

whole image.

2.1.2.3   Inverse Consistency Error

The Inverse Consistency Error[2, 3, 1] measures the consistency of the correspondence defined by forward and reverse transformations between two coordinate systems. In an ideal case, the forward transformation should equal the inverse of the reverse transformation. There are two type of Inverse Consistency Errors (ICE) computed.

- First method

$$ICE1_{ij}(x) = \|h_{ji}(h_{ij}(x)) - x\|^2,\tag{2.6}$$

- Second method

$$ICE2_{ij}(x) = \left\|h_{ij}(x) - h_{ji}^{-1}(x)\right\|^2.\tag{2.7}$$

The output will be both an image and table of numbers. The table of numbers will hold the maximum value found, the minimum value found and the average values. The table may be split up into region of interests, which means that the maximum, minimum and average will be for each of the region of interests instead of over the whole image.

2.1.2.4   Transitivity Error

The Transitivity Error (TE)[3, 10, 7] measures the consistency of the correspondence defined by compositions of transformations. In an ideal case, the compositions of the transformations should result in an identity transformation. The Transitivity Error is computed using the following equation.

$$CTE_k(x) = \frac{1}{(M-1)(M-2)} \sum_{\substack{i=1 \\ i \neq k}}^{M} \sum_{\substack{j=1 \\ j \neq i \\ j \neq k}}^{M} \|h_{ki}(h_{ij}(h_{jk}(x))) - x\|^2 \qquad (2.8)$$

The output will be both an image and table of numbers. The table of numbers will hold the maximum value found, the minimum value found and the average values. The table may be split up into region of interests, which means that the maximum, minimum and average will be for each of the region of interests instead of over the whole image.

2.1.2.5 Intensity Variance

The Intensity Variance measures the similarity between a population of images based on voxel intensity difference. In image registration applications driven by voxel intensity features, the ideal registration should result in zero voxel intensity difference between the registered images. The variance takes in a list of images that are used to compute the variance.

The equation of the Intensity Variance

Mean:

$$u = E(x) = 1/N * \sum_{i=1}^{N} x_i \qquad (2.9)$$

where N is the number of images.

Variance:

$$Var = 1/(N-1) \sum_{i=1}^{N} (x_i - u)^2 \qquad (2.10)$$

where N is the number of images.

The output will be both an image and table of numbers. The table of numbers will hold the maximum value found, the minimum value found and the average values. The table may be split up into region of interests, which means that the maximum, minimum and average will be for each of the region of interests instead of over the whole image.

2.1.2.6   Deformed Grids

The deformed grid shows how the transformation performs on straight lines, see Figure 2.1 for grid before transformation. A certain number of rows and columns of lines are created and then the transformation is applied to the grid, see Figure 2.2 for grid after transformation. Changes to the grid will be the width of the grid, the number of columns or rows and the ability to change the color of the grid. The output will only be an image.
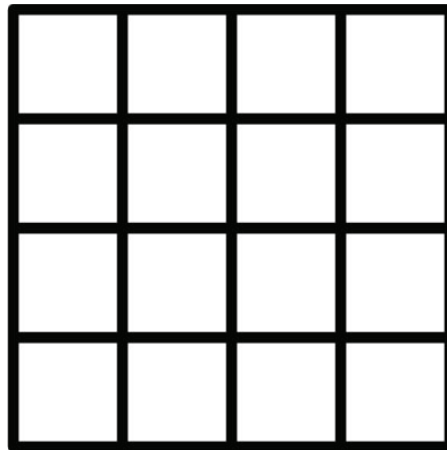
Figure 2.1: Regular 4 rows and 4 columns grid before being transformed.
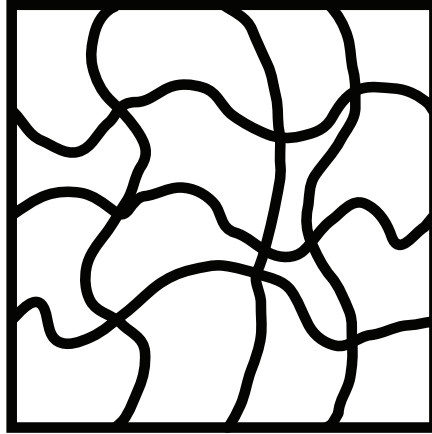
Figure 2.2: Grid from figure 2.1 after it has been transformed. The figure shows how the transformation, transforms straight lines and to a degree how the transformation looks like.

2.1.2.7 Vector Plots

The vector plot shows how the transformation moves points. The vector works by going along a defined grid to get the starting points, see Figure 2.3. Then figure out from the starting point, where the end point is by looking at the transformation for the given starting point. Then draw an arrow from the starting point to the end point, see Figure fig:vectorAfter. The vector plot will be able to change the number of starting points, the arrow width and the color of the arrows. The output will only be an image.
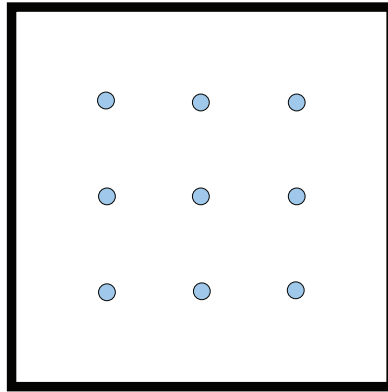
Figure 2.3: Vector plot before being transformed. As can be seen, there are only points in the vector plot but no vectors. If the transformation is the identity then this image is what should be shown.
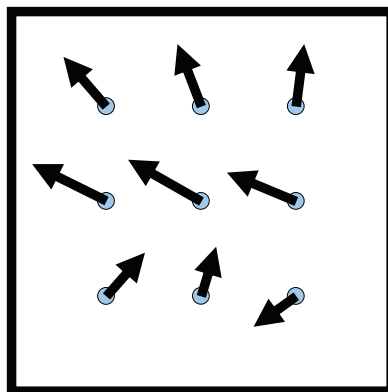
Figure 2.4: Vector plot is shown after being transformed. As can be seen, there are vectors pointing away from the points in the image. The vector plot shows the transformation.

2.1.2.8   Dice Coefficient

The alignment of objects, structures, organs, regions of interest (ROIs), etc., are a good indicator of how well two images are registered. These subvolumes are defined by partitioning or segmenting an image into objects or ROIs. The Dice Coefficient[11, 15, 5, 6] figures out the region of overlap by taking twice the intersection over the union of the segmented region of the deformed and target volumes.

Assume S_i and T_i are defined as the ith segmented region in the deformed source and target volumes.

The Dice Coefficient in equation form is:

$$MO_i(S_i, T_i) = 2\frac{|S_i \cap T_i|}{|S_i| + |T_i|} \tag{2.11}$$

The output will be both an image and table of numbers. The table of numbers will hold the maximum value found, the minimum value found and the average values. The table may be split up into region of interests, which means that the maximum, minimum and average will be for each of the region of interests instead of over the whole image.

2.1.2.9   Relative Overlap

The alignment of objects, structures, organs, regions of interest (ROIs), etc., are a good indicator of how well two images are registered. These subvolumes are defined by partitioning or segmenting an image into objects or ROIs. The Relative Overlap (RO)[12, 13, 8, 5, 6, 11] assesses how well two equally likely segmentations

of the same region of interest (ROI) agree or disagree with each other. Ideally, RO of all regions should be 1.0.

Assume S_i and T_i are defined as the ith segmented region in the deformed source and target volumes.

The relative overlap in equation form:

$$UO_i(S_i, T_i) = \frac{|S_i \cap T_i|}{|S_i \cup T_i|} \tag{2.12}$$

The output will be both an image and table of numbers. The table of numbers will hold the maximum value found, the minimum value found and the average values. The table may be split up into region of interests, which means that the maximum, minimum and average will be for each of the region of interests instead of over the whole image.

2.1.2.10 Transform Images and Segmentations

The transformations read in from disk are a mapping of one coordinate system into another coordinate system. The program will use the transformations to transform images and segmentations.

2.1.3 Visualization

The results of the evaluations and the images on disk will be displayed in a viewer. The 2DViewer is just a viewer of data whereas the difference, checkerboard, wipe, superimpose segmentations over images are viewers used for evaluation. In all, the visualization views will be qualitative evaluations of data.

2.1.3.1 2DViewer

The 2DViewer will visualize the data as a single slice of the data. The location of the slice will be able to be changed. As well, the 2DViewer will follow the properties outlined in the section Visualization View properties.

2.1.3.2 Difference

The difference view is needed to show where two images do not align correctly by taking the difference of the images. Wherever the two images do not align properly, there will be a visual intensity as computed by the difference. If the images align perfectly, a blank screen will be displayed. The view will be like the 2DViewer, in that slices of the images will be shown.

2.1.3.3 Checkerboard

The checkerboard is like the difference, in that the view shows where two images do not align correctly. However, this view shows the images in a checkerboard format, like Figure 2.5 shows. The two images are placed on top of each other but only parts of each of the images will be shown. The white square's correspond to the parts of the first image being shown and the black squares correspond to the parts of the second image being shown. The number of squares within the checkerboard can be changed by the user. The reason for this view is to show where the two images do not align along the borders of the squares.
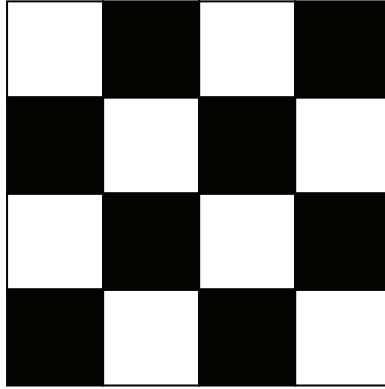
Figure 2.5: Rough sketch of a Checkerboard view. The view takes in two images, which are overlaid on top of each other. The white squares correspond to a portion of image 1 being visible but not image 2 and the black squares correspond to a portion of image 2 being visible and not image 1.

2.1.3.4   Wipe

The Wipe view is like the checkerboard, in that the view shows where two images do not align correctly. However, this view is a 2x2 grid that the user is allowed to grab the borders and wipe across the checkerboard, see Figure 2.6. Another way to think of this, is that the size of the squares changes but the overall size of the checkerboard is kept the same. The reason for this view is to show where the two images do not align along the borders of the squares.

2.1.3.5   Superimpose Segmentations Over Images

Segmentations are a way to split up and label an image with anatomical locations. In having segmentations, the errors are localized to the different regions. The program needs to be able to superimpose the segmentations over images to see where the regions are in conjunction with the image. The program should be able to change
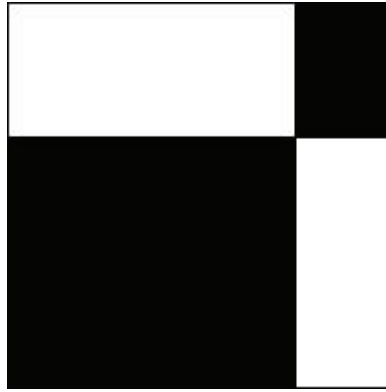
Figure 2.6: Rough sketch of Wipe view. The view takes in two images, which are overlaid on top of each other. The white squares correspond to a portion of image 1 being visible but not image 2 and the black squares correspond to a portion of image 2 being visible and not image 1. The user can grab the boundaries and wipe across the two images.

the opacity of the segmentations so that either more of the image can be seen or more of the segmentation can be seen.

2.1.3.6 Color Map

Users will want to view images in different color schemes to bring out different aspects of the images. All of the visualization views will have color maps. Color maps are a mapping of an intensity value to a color. The default color map is black and white but can be changed to user specified color maps.

2.1.3.7 Color Bar

Users will want to have a quick method to tell what color maps with what intensity. The color bar will tell the mapping of intensity's to color's in a bar chart.

2.1.4   Visualization Interactions

All of the visualization views will have the following properties.

2.1.4.1   On the Fly Change of Inputs

Different inputs to the evaluation algorithms may change the outputs of the algorithms. When the user is evaluating registration algorithms, they will want to see if the outputs of the evaluation algorithms do change with different inputs. This means that while the program is running, the inputs to the visualization views will be able to be changed.

2.1.4.2   Specify Orientation

All of the visualization views are slices of an image of a certain orientation. The user will want to see different orientations to help evaluate the registration algorithms. The views will allow the user to specify which orientation to view.

2.1.4.3   See Cursors

When a user clicks on an image, they will want a way to remember the location of the click. The program will use a cross-hair, a vertical and horizontal line, to remember the location of the click. For different orientations the colors of the cross hairs will change to reflect what the horizontal line represents and what the vertical line represents.

2.1.4.4   See Value at Clicked Location

When a user clicks on an image they will want to see the intensity value. The intensity value can correspond to a numerical analysis of an algorithm. The program will display the intensity value within the visualization view.

2.1.4.5   Cursor Locks

With the user being able to click on an image and seeing cross hairs and intensity values, the different views will be able to be locked together. This means that if the user clicks in one view, all other views that are linked to that one view are updated with the new click information. The user is able to lock every view, just rows of view, just columns of view, or individually choose which views to lock.

2.1.5   Quantitative Analysis

Some of the evaluations produce numerical values which require a viewer. The following viewers will display the numerical values from the evaluations. In all the numerical views will be the display of quantitative data.

2.1.5.1   Tables

Tables will be a raw dump of the numerical values produced by an evaluation. There will be headings for the columns and row to help understand what each cell in the table means. This helps a user to quantitatively say which algorithm is better.

2.1.5.2 Graphs

Graphs will be a visual representation of the numerical values produced by an evaluation. Graphs encompasses all the types of graphs, this includes pie charts, bar graph, scatter plot and whisker plots. For those graphs that have horizontal and vertical axis a title will be displayed.

2.1.5.3 Save Results to Disk

The program will output the quantitative reports out to disk. The user can then submit the reports to the NIREP webpage to share with everyone. The webpage will be able to display the evaluations so that people around the world can see how different Non-Rigid Image Registration algorithms perform for different evaluation algorithms.

2.1.6 Display Layout

All of the view's, both visualization and numerical must have the following properties.

2.1.6.1 Grid Layout

All of the views will be laid out in a grid format. In having a grid format, the program saves valuable screen space. In saving screen space, there is greater detail that a user can use to evaluate the different algorithms. Each of the positions of the views is user specified. Figure 2.7 shows a grid layout of 3 columns and 3 row of two images being evaluated using difference. The grid has titles for both the row and columns, explaining what is happening in the grid. Each of the rows are depicting

one of the three orientations that are possible within NIREP. Each of the columns are depicting the first image, the difference of image 1 and image 2 and image 2. Within each of the views are different objects that are within the images. For the difference column, for the first row, it can be seen that the rectangle is shown but not the circle because the circle's completely overlap in image 1 and image 2. For the difference column, for the second row, it can be seen that the rectangle in image 1 and rectangle in image 2 do not align at all and so both rectangles are shown in the difference. For the difference column, for the third row, it can be seen that the circle of image 1 and rectangle of image 2 overlap each other however the rectangle of image 2 overlaps more then just the circle, so some of the rectangle are shown.

2.1.6.2 Titles of Each Column and Row

The rows and columns of the grid layout, must have a title, as seen in Figure 2.7. This allows a user to easily understand what is being viewed and how the evaluations are being done. When the inputs of the views are changed, the titles will change as well. The text within the titles will be able to be changed. The title properties, the font, the font size, the color, italicized, bold and underline, will be able to be changed.

2.1.6.3 Change View

While the program is running the user will be able to change each individual view. When the view is changed, default values will be used unless otherwise specified by the user. The user may want to try a different view for evaluations and may also
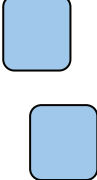
	Image 1	Difference	Image 2
SAGITTAL			
CORONAL			
TRANSVERSE			

Figure 2.7: Grid layout showing a 3 column and 3 row of two images being evaluated using difference. The grid has titles for both the row and columns, explaining what is happening in the grid. Each of the rows are depicting one of the three orientations that are possible within NIREP. Each of the columns are depicting the first image, the difference of image 1 and image 2 and image 2. Within each of the views are different objects that are be within the images. For the difference column, for the first row, it can be seen that the rectangle is shown but not the circle because the circle's completely overlap in image 1 and image 2. For the difference column, for the second row, it can be seen that the rectangle in image 1 and rectangle in image 2 do not align at all and so both rectangles are shown in the difference. For the difference column, for the third row, it can be seen that the circle of image 1 and rectangle of image 2 overlap each other however the rectangle of image 2 overlaps more then just the circle, so some of the rectangle are shown.

change the inputs for a completely different evaluation.

2.1.6.4 Save State

Users will want to re-use the same evaluation. This means that the current state of the view will have to be saved. The slice number, the zoom factor, the orientation and the locking of views will have to be saved.

2.1.7 Additional Requirements

2.1.7.1 Open Source

The program must not be a black box but a program that people can see the evaluation algorithms being used. If users want, they can modify their Non-Rigid Image Registration Algorithms to better perform for certain evaluation algorithms. As well, people should be able to help improve NIREP.

2.1.7.2 Distribution

NIREP will be given out to people to independently run and perform evaluations of their non-rigid image registration algorithms. However, the program will not run the Non-Rigid Image Registration Algorithms. When NIREP is given out, the platforms that will be supported are Windows and Linux.

2.2 NIREP Software Design

This section talks about the NIREP software design. Looking over the specifications, four main classes were revealed: Data Manager, Evaluator, Display Manager and Interface. The Data Manager takes care of reading in the data and controlling the

data in memory (see Section 2.2.2). The Display Manger displays a grid of images and statistics (see Section 2.2.4). The Evaluator is the heart of NIREP and takes care of bringing all the evaluation algorithms together evaluating the data (see Section 2.2.3). The Interface deals with the start up of NIREP and the first interaction with the user. Some things that the Interface does is create the Data Manger and Evaluator, spawn off a new Display Manager whenever the user wants to display data or evaluations (see Section 2.2.1).

2.2.1 Interface

The Interface is the first Graphical User Interface (GUI) shown to users and can be thought of as the anchor or base for NIREP. When the Interface is closed NIREP quits, whereas if users close a Display Manager, NIREP will still keep on running. Users will have the ability to create a Display Manager by reading in a display description or creating a display description - both of which describes how a Display Manager looks. See Display Manager for further information about the Dispaly Manager.

The Use Case for the Interface is shown in Figure 2.8. The Use Case is initiated by a User and the entry condition is that the user opens NIREP. When the user opens NIREP, then the Interface will be created. After the Interface is created, the Interface will create an instance of the DataManager. After the DataManager is created, the Interface will create an instance of the Evaluator and pass a reference of the DataManager to the Evaluator. The Use Case is illustrating that there will be one instance of the DataManager and one instance of the Evaluator and references

Use case name	User opens NIREP for first time
Participating actors	Initiated by User Interface DataManager Evaluator
Flow of events	1. Interface creates an instance of DataManager 2. Interface creates an instance of the Evaluator and passes a reference of the DataManager
Entry condition	⋏ User opens NIREP
Exit condition	⋏ Evaluator is created
Quality requirements	

Figure 2.8: Use case for the Interface. The use case goes over a user opening NIREP for the first time.

Interface class diagram
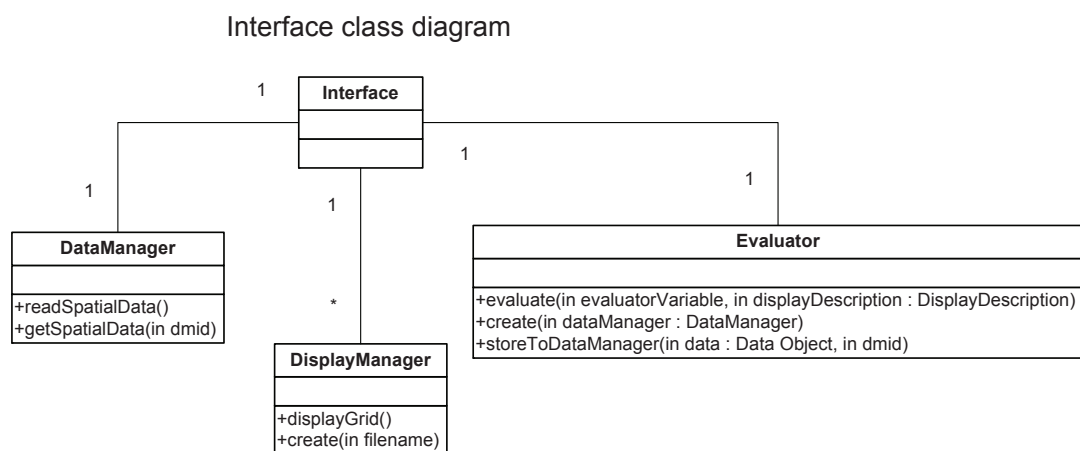
Figure 2.9: Class diagram for the Interface. The Interface has a reference to the DataManger, the Evaluator and to the DisplayManager's.

passed around.

The Class diagram for the Interface is shown in Figure 2.9. As can be seen in the Class diagram there will be one DataManager, one Evaluator and 0 to as many as the user wants of DisplayManagers. Also, there have been some functions found that would be useful in the different classes.

The sequence diagram for the Interface is shown in Figure 2.10. As can be

User Opens NIREP for first time
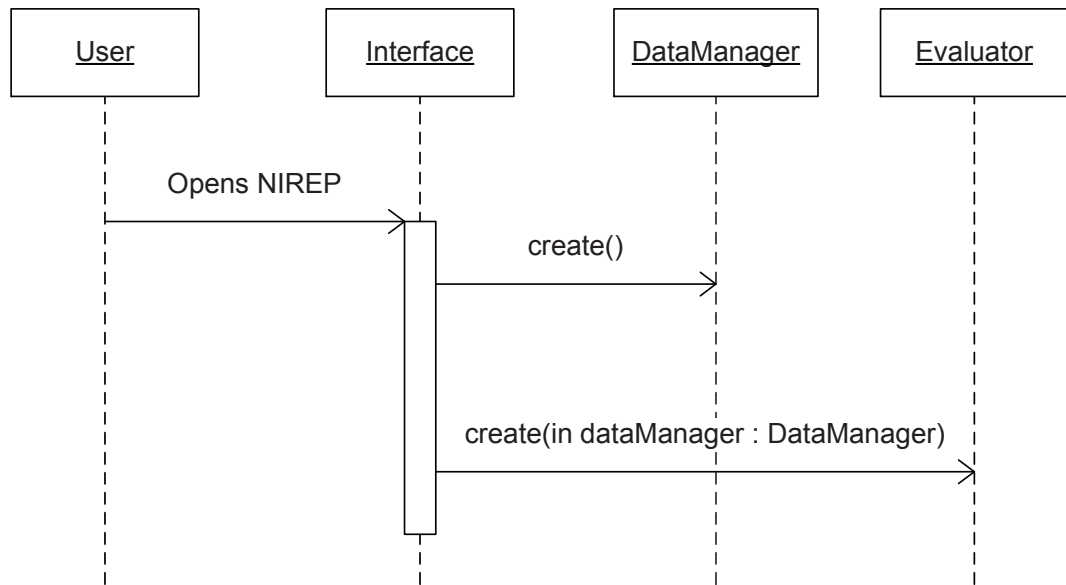
Figure 2.10: Sequence diagram for the Interface. The sequence diagram follows the use case of a user opening NIREP for the first time.

seen, the sequence diagram follows the use case. Time runs from top to bottom. The User opens NIREP, which creates the Interface. From there, the Interface calls the create function of the DataManger, to create an instance of the DataManager. From there the Interface calls create of the Evaluator and at the same time passes a reference of the DataManager.

2.2.2 Data Manager

The Data Manager is a stand alone manager of the data on disk and in memory. The Resource Description List (RDL) within the Data Manager is used to find the data on disk and create unique Data Manager Identification's (DMID) for the data

in memory. The DMID will be used as a means to look up the data in memory, this way duplicate data is not stored in memory. If a piece of data is not defined in the RDL, then there will need to be a prompt to the user to fill in a description of the data on disk or a creation by an evaluation task. See RDL for further information about the RDL.

The Class diagram for the Data Manager is shown in Figure 2.11. As can be seen in the Class diagram there will be one Resource Description List and 0 to as many as read in Data Objects. The Data Object will serve as a superclass for all data. In having a superclass the program can use polymorphism and an easy way to pass around data within the program. A polymorphic superclass helps to bring together classes that share common traits and function calls. The inheriting classes can override the functions but to the program, the classes look the same and only the common function calls are used except for specific times. For future data classes, they can inherit from Data Object and very little changes will need to be made to the program. Also, there have been some functions found that would be useful in the different classes.

2.2.3 Evaluator

The Evaluator is a manager of evaluator function names with their corresponding evaluation object classes. The evaluation object classes will all inherit from a super class called Statistic Tasks. In doing the inheritance, the usage of polymorphism can be done and the addition of more statistics is easily done. NIREP will start off with the following statistics: Dice Coefficient, Intensity Variance and Inverse
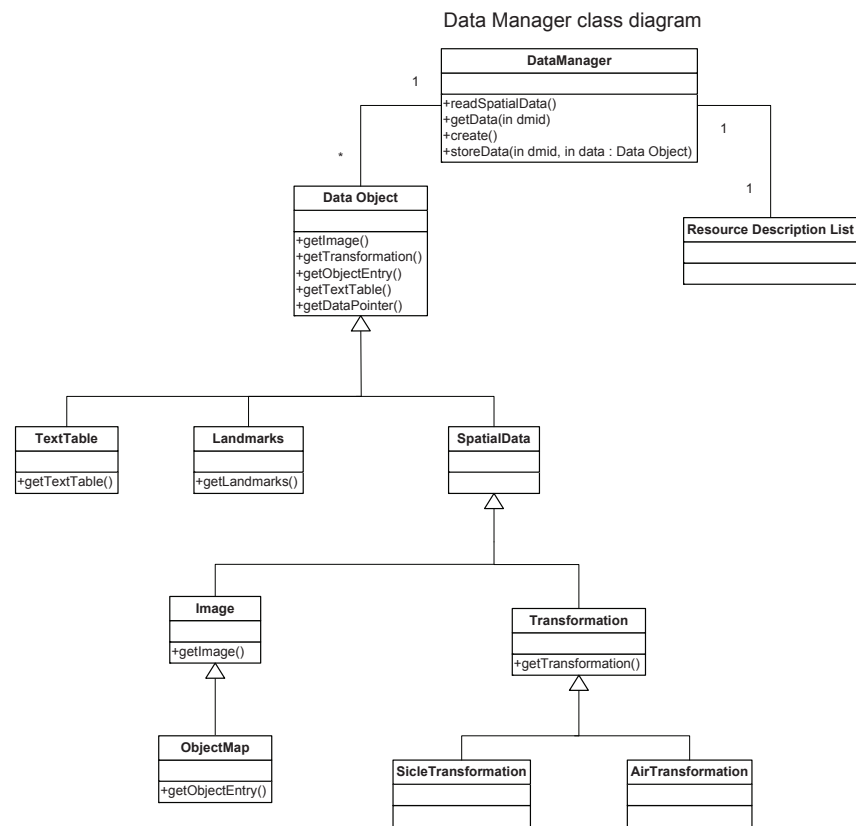
Figure 2.11: Class diagram for the Data Manager. The Data Manager keeps track of all the data that NIREP needs to use. To keep track of the data, the Data Manager needs a reference to 0 to as many specified DataObjects.

Consistency Error.

The Use Case for the Evaluator is shown in Figure 2.12. This specific use case will describe how a Difference Evaluator Task will compute the difference. This specific Use Case will be a framework for how other Evaluator Task's will compute other evaluation algorithms. The Use Case is initiated by the Panel and the entry condition is that the Panel calls Evaluate of the Evaluator. From there, the Evaluator will figure out the evaluator command, this will be a fully qualified name that if the data is in memory, will be quickly found. From there, the Evaluator will ask the Data Manager for data using the evaluator command as the key to look up the data. The Data Manager will see if the data is in memory. In this case the data is not in memory, so NULL is returned. From there the Evaluator will look up the evaluator command's function name, in doing so the different Evaluator Task's can be easily looked up for computation or a base case will be hit. In this case the function name is difference, which corresponds to the Difference Evaluator Task. The Evaluator will then call the Evaluate function of the Evaluator Task. From here the Evaluator Task will search for it's inputs by asking the Evaluator for it's inputs. The evaluator will go through the same process as looking up `difference(image1,image2)`. In this respect, the Evaluate function of the Evaluator is acting like a recursion function until a base case is hit. In the example, the base case is hit when an evaluator command's function name is `readSpatialData`. When the base case is hit, the Evaluator will ask the Data Manager to read in the data. The Data Manager will read in the data and return a pointer to the data. From there the Evaluator will return a pointer to

the data to the Evaluator Task. When the Evaluator Task eventually gets a pointer to both image1 and image2, then the Evaluator Task can compute the difference in this case. When the Evaluator Task is done computing the evaluations, the Evaluator Task tells the Evaluator to store the newly created data. The Evaluator will then tell the Data Manager to store the data, since the Data Manager is the holder of all data. Finally the Evaluator Task will call Evaluate of the Evaluator one more time to get the created data. The reason for this, is that the Evaluator Task can create more then one new data and doesn't know which one to return. So the Evaluator will go through the process of asking the Data Manger for the data, which should now be in memory. The Data Manager will return a pointer to the data, the Evaluator will return a pointer to the data, the Evaluator Task will return a pointer to the data and finally the Evaluator will return a pointer to the data to the Panel. The exit condition is when the Panel receives a pointer to the newly created data.

The Class diagram for the Evaluator is shown in Figure 2.13. As can be seen in the Class diagram there will be one DataManager and 0 to as many as created of Evaluator Tasks. The Evaluator Task will be a super class that all evaluation algorithms inherit from. In having a super class, the program can use polymorphism to use the evaluation algorithms. By using polymorphism the addition of new evaluation algorithms will be easy and very minimal changes to the program will need to be done. Also, there have been some functions found that would be useful in the different classes.

The sequence diagram for the Evaluator is shown in Figure 2.14. As can be

Use case name	Difference Evaluator Task
Participating actors	Initiated by Panel Evaluator Difference Data Manager EvaluatorList
Flow of events	1. Evaluator asks the EvaluatorList for the evaluator command of difference 2. EvaluatorList returns the evaluator command (difference(image1,image2)) 3. Evaluator asks Data Manager for the differenceData based on the evaluator command 4. Data Manager responds back with NULL 5. Evaluator then looks up the Difference based on the evaluator command's function name (i.e. difference) 6. Once Difference is found, call the Evaluate function of the difference 7. The Difference calls the Evaluate function of the Evaluator for image1. 8. Evaluator asks the EvaluatorList for the evaluator command for image1. 9. EvaluatorList returns the evaluator command (readSpatialData(na0|01,MRI)) 10. The Evaluator asks the Data Manager for the image1Data based on the evaluator command 11. Data Manager responds back with NULL 12. The Evaluator asks the Data Manager to read in image1Data 13. Data Manager reads in image1Data and returns a pointer to image1Data 14. The Evaluator returns a pointer to image1Data to the Difference 15. The Difference calls the Evaluate function of the Evaluator for image2. 16. Evaluator asks the EvaluatorList for the evaluator command of imag2 17. EvaluatorList returns the evaluator command (readSpatialData(na0|02,MRI)) 18. Evaluator asks Data Manager for the image2Data based on the evaluator command 19. Data Manager responds back with NULL 20. The Evaluator asks the Data Manager to read in image2Data 21. Data Manager reads in image2Data and returns a pointer to image2Data 22. The Evaluator returns a pointer to image2Data to the Difference 23. The Difference computes the difference of the two images 24. The Evaluator Task calls the StoreToDataManager function of the Evaluator, to keep track of the newly created data 25. The Evaluator asks the Data Manager to keep track of the new data 26. The Data Manager puts the new data in a map of in memory data objects 27. The Evaluator Task calls the Evaluate function of the Evaluator to get the new data 28. Evaluator asks the EvaluatorList for the evaluator command of difference 29. EvaluatorList returns the evaluator command (difference(image1,image2)) 30. The Evaluator asks the Data Manager for the differenceData 31. The Data Manager returns a pointer to the differenceData to the Evaluator 32. The Evaluator returns a pointer to the differenceData to the Difference 33. The Difference returns a pointer to the differenceData to the Evaluator 34. The Evaluator returns a pointer to the differenceData to the Panel
Entry condition	⋏ Panel calls Evaluate of the Evaluator
Exit condition	⋏ Panel receives a pointer to the newly created data
Quality requirements	

Figure 2.12: Use case for the Difference Evaluator Task. The use case goes over a typical request for an evaluation task, such as the Difference.
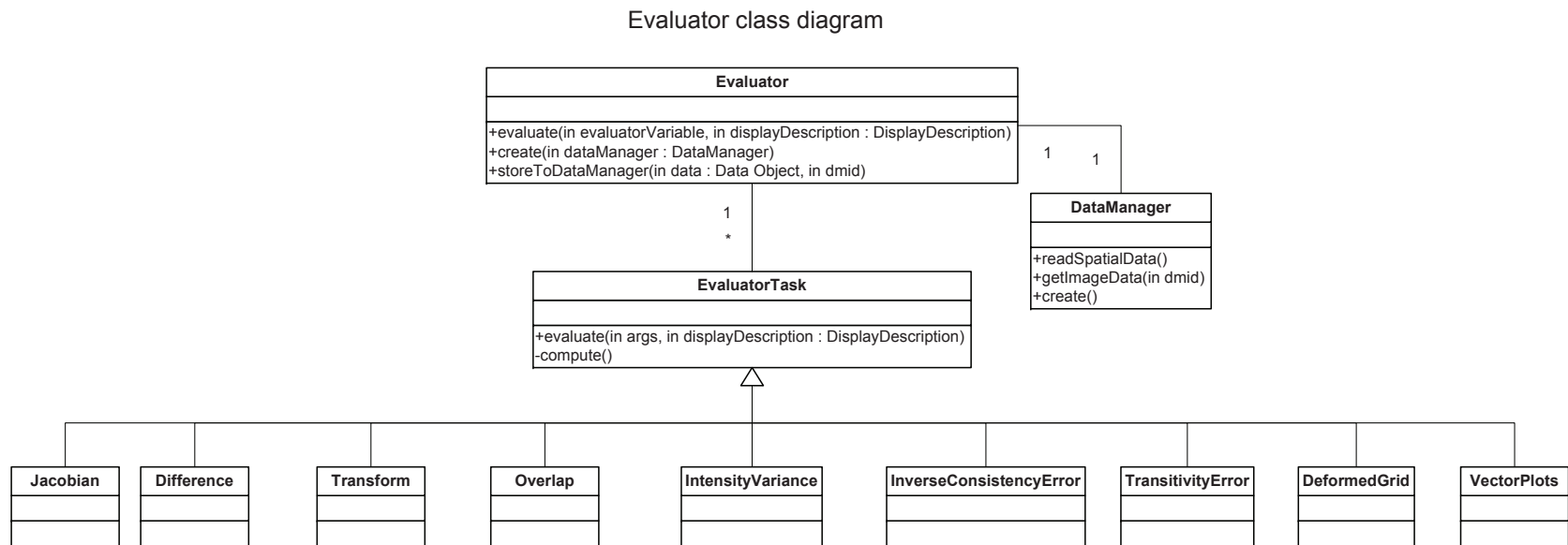
Figure 2.13: Class diagram for the Evaluator. The Evaluator has references to the DataManager and EvaluatorTasks. All evaluator tasks, such as Difference, RelativeOverlap, Transitivity Error, will inherit from EvaluatorTask so that the program can use polymorphism.

seen, the sequence diagram follows the use case. Time runs from top to bottom. The Panel calls Evaluate of the Evaluator with some parameters. From there the Evaluator asks the EvaluatorList for the evaluator command and the EvaluatorList returns the differenceCommand. The Evaluator will ask the DataManager for the data that corresponds with the differenceCommand. The DataManager will return NULL signifying that the data is not in memory. The Evaluator will then figure out the evaluator command function name, which will be difference. From there the Evaluator will look up the Difference EvaluatorTask and call the function evaluate with some parameters. From there the Evaluator Task will ask the Evaluator for its inputs, image1 and image2. Eventually the EvaluatorTask will get it's inputs and then compute the difference. Once the EvaluatorTask has computed the evaluations, it will call storeToDataManager of the Evaluator. The Evaluator will then call storeData of the DataManager to store the newly created data. Then the EvaluatorTask will ask the Evaluator for the newly created data because the EvaluatorTask can create more then just one data and doesn't know which to return. The Evaluator will ask the DataManger for the data and return a pointer to the data. The Evaluator will return a pointer to the data to the EvaluatorTask. The EvaluatorTask will return a pointer to the data to the Evaluator. Finally the Evaluator will return a pointer to the data to the Panel. The ending part where the pointer is passed along is wrapping up the recursion that was performed.

Difference Evaluator Task
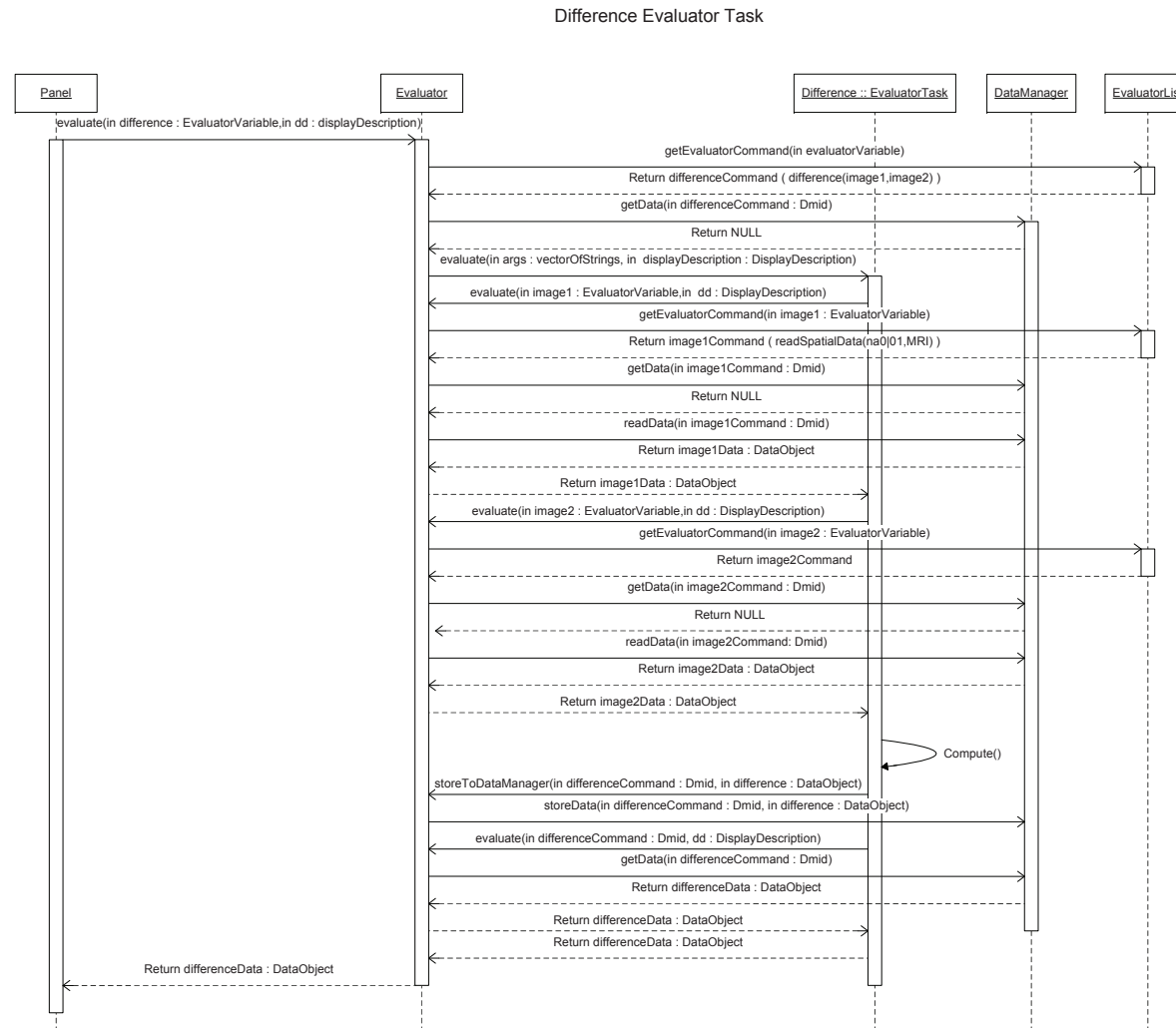
Figure 2.14: Sequence diagram for the Evaluator. The sequence diagram follows the use case of a typical request for an evaluation task, such as the Difference.

2.2.4   Display Manager

The Display Manager is the manager of displaying the qualitative and quantitative evaluations. To be able to display the evaluations a Display Description is needed that describes the visualization, the evaluations and the data. The Display Description can be used in multiple Display Managers. The Display Manager will display the evaluations in a grid layout, where each cell of the grid contains a panel. The Display Manager will be responsible for locking the panels. Each of the Panels consists of one Widget and a title and can be thought of as a manager of that Widget and title. The panels will take care of changing the widgets, and changing the title of the panel. A widget is a super class that other views can inherit from, so that polymorphism can be used and the addition of future widgets is easy. In addition, some of the widgets will inherit from the 2D View Widget, for specialized cases. The different widgets are: 3D View Widget, 2D View Widget, Checkerboard Widget, Difference Widget and Wipe Widget.

The Use Case for the DisplayManager is shown in Figure 2.15. This example will use a display description to illustrate a user displaying a 2DViewer from the interface. The Use Case is initiated by the Interface and the entry condition is that the Interface creates a DisplayManager. When the Interface creates a DisplayManager, its going to pass along the Display Description file name. This is the location on disk where the Display Description file is located. From there the DisplayManager will create a DisplayDescription. The DisplayDescription will represent the Display Description file on disk as an object in memory. The DisplayDescription

will create a VariableList, an EvaluatorList, a WidgetList, a DisplayAttributes and a ResourceDescriptionList. When the DisplayDescription creates the 5 classes, it will pass the file name to each of them and in turn each of the classes will parse the file and pull out the relevant data. Once the DisplayDescription is created, the DisplayManager will create one panel for this example. A panel can be thought of as a manager of a cell in the grid within the DisplayManager. The DisplayManager will ask the WidgetList for the inputs for the specific panel and then adds the inputs to the Panel. Then the DisplayManager will ask the WidgetList for the widget type for the specific panel. A widget type is a key into which type of view to use, for this example the widget type is 2DViewer. The DisplayManager will then call useWidget of the Panel while passing in the widget type. From there the Panel will create a 2DViewer. Then the Panel will ask the Evaluator for data based on the inputs the DisplayManager passed in. The Evaluator will get the data and return a pointer to the data. The Panel will then call setInput of the 2DViewer. Next the DisplayManager will ask the DisplayAttributes for the attributes and then distribute those attributes to the respective panels. The exit condition is when the DisplayManager is displayed.

The Class diagram for the DisplayManager is shown in Figure 2.16. As can be seen in the Class diagram there will be one DisplayDescription and composed of 0 to as many as Panels as the user specifies. The black diamond means that the DisplayManager is composed of Panel. As well, the Widget class will be a super class for all views. In having a super class, the program can use polymorphism for future views. Also, there have been some functions found that would be useful in the

Use case name	User Displays a 2D viewer from the Interface, using a display description
Participating actors	Initiated by Interface Evaluator DisplayDescription Panel DisplayManager DataManager EvaluatorList WidgetList DisplayAttributes ResourceDescriptionList 2DViewer
Flow of events	1. Interface passes Display Description file name to DisplayManager 2. DisplayManager creates a DisplayDescription, while passing in the Display Description file name 3. DisplayDescription creates VariableList while passing in the Display Description file name 4. DisplayDescription creates EvaluatorList while passing in the Display Description file name 5. DisplayDescription creates WidgetList while passing in the Display Description file name 6. DisplayDescription creates DisplayAttributes while passing in the Display Description file name 7. DisplayDescription creates ResourceDescriptionList while passing in the Display Description file name 8. DisplayManager creates one Panel 9. DisplayManager asks WidgetList for the inputs for the specific panel 10. DisplayManager adds the inputs to the Panel 11. DisplayManager asks WidgetList for the widget type for the specific panel 12. DisplayManager calls useWidget of the Panel 13. Panel creates a 2DViewer 14. Panel asks Evaluator for data 15. Evaluator asks DataManager for data 16. DataManager reads data from disk 17. DataManager returns data 18. Evaluator returns data 19. Panel calls setInput of the 2DViewer 20. DisplayManager asks the DisplayAttributes for the attributes 21. DisplayAttributes returns the attributes 22. DisplayManager distributes the attributes
Entry condition	⊥ Interface creates DisplayManager
Exit condition	⊥ DisplayManager is displayed
Quality requirements	

Figure 2.15: Use case for the DisplayManager. The use case is for a user displaying a 2D viewer from the interface, using a display description.

different classes.

The sequence diagram for the Display Manager is shown in Figure 2.17. As can be seen, the sequence diagram follows the use case. Time runs from top to bottom. The Interface creates a DisplayManager and passes in the file name for the Display Description. The DisplayManager creates a DisplayDescription and passes in the file name. From there the DisplayDescription creates a VariableList, EvaluatorList, WidgetList, DisplayAttributes and ResourceDescriptionList, all the while passing in the file name. The 5 classes will read the file name and pick out the relevant data from the file. From there the DisplayManager will create a Panel. Then the DisplayManager will asks the WidgetList for the parameters, which will be the inputs to the Panel. The DisplayManager then set's the inputs to the Panel. Then the DisplayManager asks the WidgetList for the widget type. From there the DisplayManager calls useWidget, passing in the widgetType, which for this example is 2DViewer. Then the Panel will create a 2DViewer. Then the Panel will ask the Evaluator for the data and will eventually get the data. From there the Panel will call setInput of the 2DViewer and pass in the data. Finally the DisplayManager will ask the DisplayAttributes for the attributes and then distribute the attributes to the respective panels.

2.2.5   NIREP Display Description Document

The Class diagram for the Display Description is shown in Figure 2.18. The Display Description consists of one Evaluator List, one Widget List, one Variable List, one Display Attributes and one Resource Description List. The Resource Description

Display Manager class diagram
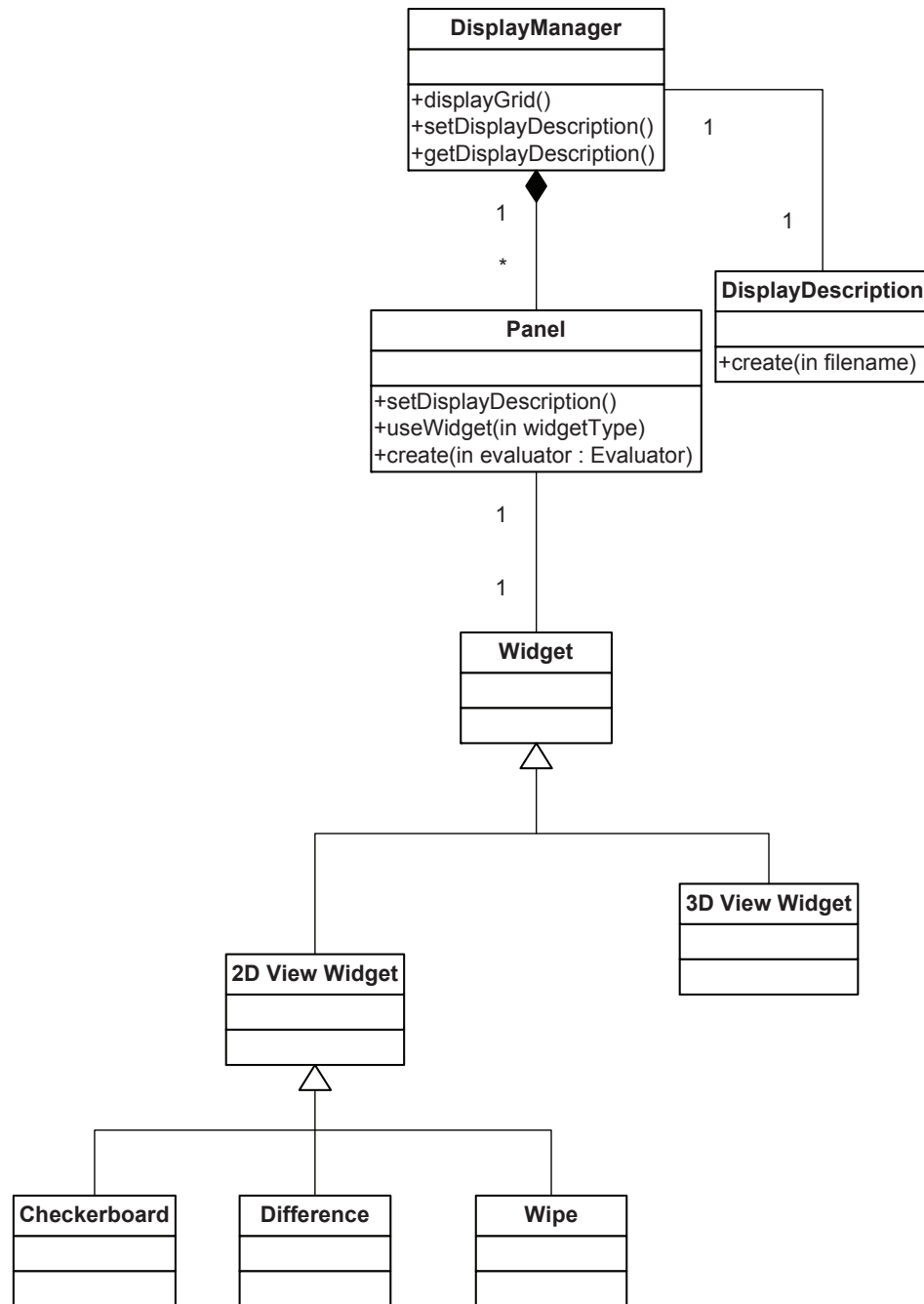
Figure 2.16: Class diagram for the Display Manager.
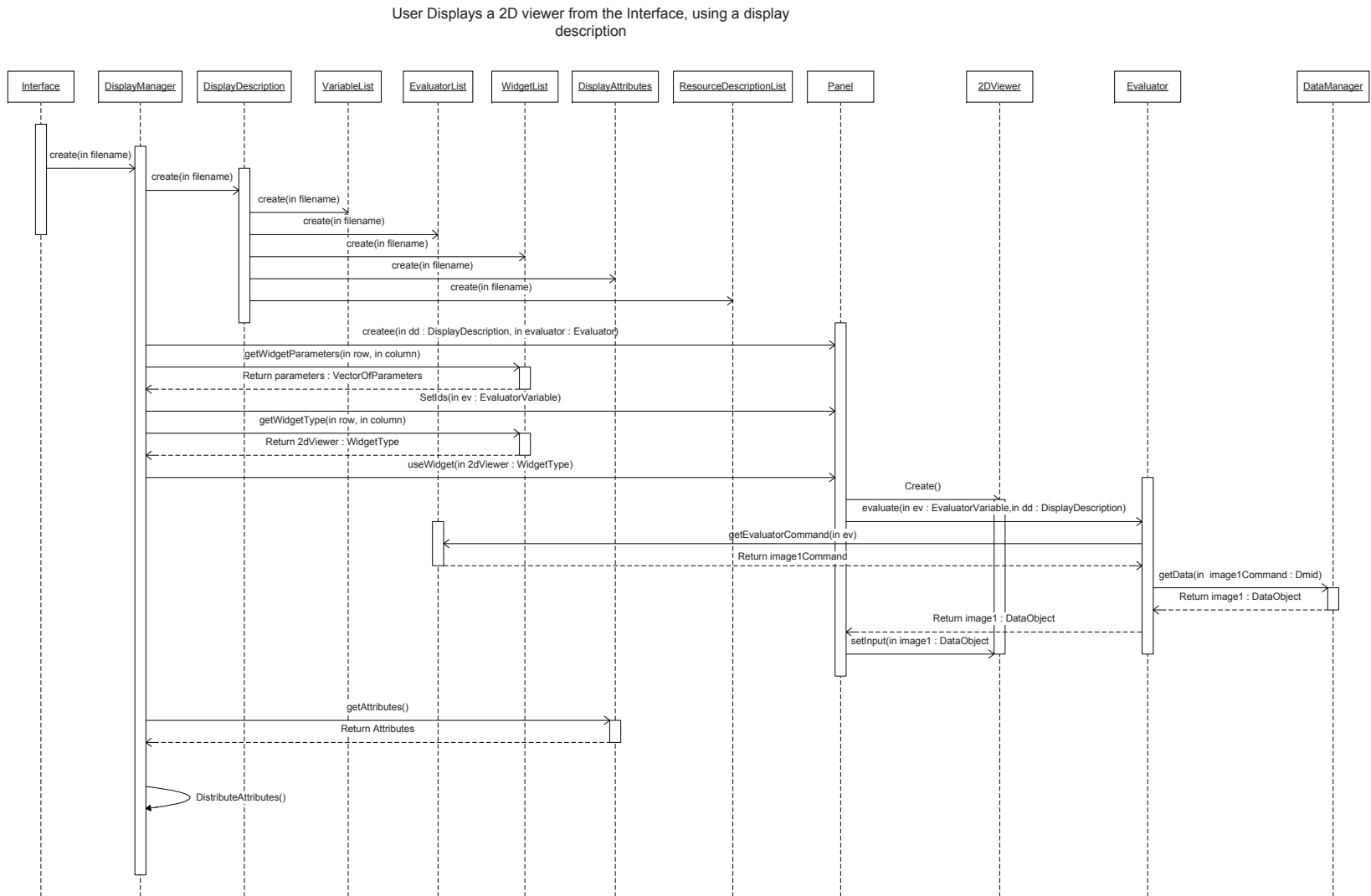
Figure 2.17: Sequence diagram for the evaluator use case, which is the user displaying a 2D viewer from the interface using a display description.

Display Description class diagram
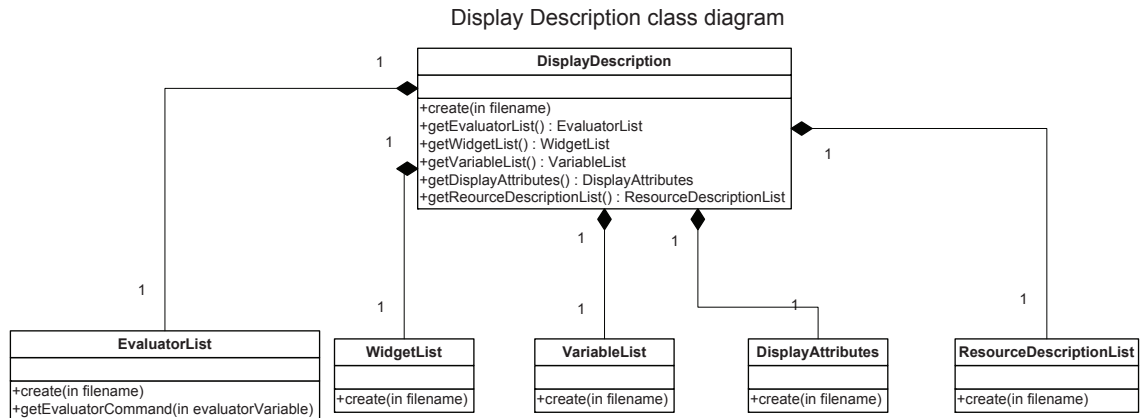
Figure 2.18: Class diagram for the Display Description. The Display Description is composed of the EvaluatorList, WidgetList, VariableList, Display Attributes and ResourceDescriptionList. The Display Description is a copy of the ND3 file but read into memory.

list consists of multiple Basic Descriptions and multiple Resource Descriptions. The Display Description in a sense, is a holder class for the various other classes that make up how the display will be presented to the user.

Users will tell NIREP how to display the data by creating a NIREP Display Description Document (ND3). The ND3 file is responsible for controlling the behavior of the Display Manger, Evaluator and DataManger and the human-readable file typically looks like the following:

```
Begin ResourceDescriptionList

  Databases\NA0.2\resources.rdl

  Transformations\NA0.2\AIR5_Param1\air5.rdl

  Transformations\NA0.2\SICLE_Param1\sicle.rdl

  Transformations\NA0.2\BRAINSDemons_Param1\demons.rdl
```

```
End ResourceDescriptionList


Begin DisplayAttributes

columnSize(2)

rowSize(2)

LockAll()

End DisplayAttributes


Begin VariableList

image1 = na0|008

image2 = na0|014

modality = MRI

End VariableList


Begin WidgetList

W1,1 = view(_image1,Image ${image1})

W1,2 = checkerboard(_image1,_image2,Images ${image1} and ${image2})

W2,1 = view(_image2,Image ${image2})

W2,2 = wipe(_image1,_image2,Images ${image1} and ${image2})

End WidgetList


Begin EvaluatorList
```

```
_image1 = SpatialData(image1,modality)

_image2 = SpatialData(image2,modality)

_A3 = Transformation(image1,image2,SICLE_Param1)

_A20 = Transformation(image2,image1,SICLE_Param1)

_A7 = SpatialData(image1,OBJMAP)

_A8 = SpatialData(image2,OBJMAP)

_A9 = Jacobian(_A3)

_A10 = TransformImage(_image1,_A3)

_A27 = InverseConsistencyErrorImage(_A20,_A3,inv)

_A29 = InverseConsistencyErrorImage(_A3,_A20,comp)

_A30 = InverseConsistencyErrorImage(_A20,_A3,comp)

End EvaluatorList
```

The ND3 file is divided up into five different segments:

1. Resource Description List

2. Display Attributes

3. Variable List

4. Widget List

5. Evaluator List

2.2.5.1   Resource Description List

This section is demarcated by Begin ResourceDescriptionList and End ResourceDescriptionList operatives. Each line in this section lists relative paths to Resource Description List files which provide information about how data will be read into NIREP.

2.2.5.2   Display Attributes

This section is denoted by Begin DisplayAttributes and End DisplayAttributes. There are seven Display Attribute commands that can be specified:

1. RowSize(int r) Row size of the NIREP Display Widget grid. e.g. RowSize(3)

2. ColumnSize(int c) Column size of the NIREP Display Widget grid. e.g. ColumnSize(3)

3. Lock(Widget w1; Widget w2; ...; Widget wN) Locks behaviors of specified widgets. e.g. Lock(W1,1;W1,2;W1,3)

4. CursorLocation(Widget w; int x; int y) Default cursor location of specified widget. e.g. CursorLocation(W1,1;50;128)

5. Slice(Widget w; int z) Default image slice location of specified widget. e.g. Slice(W2,2;150)

6. Orientation(Widget w; Orientation o) Default orientation of specified widget. e.g. Orientation(W3,1;TRANSVERSE)

7. LockAll() Locks all Display widgets.

RowSize() and ColumnSize() are required while the remaining commands are optional.

2.2.5.3 Variable List

This section is specified between Begin VariableList and End VariableList operatives. The variables that will be shared between the Widget List and the Evaluator List will be specified on each line, following a variable_name = variable format. The variables can be changed dynamically during runtime in NIREP, making mass switch of variables simple (e.g. simultaneously changing the display of multiple widgets with a single variable change).

2.2.5.4 Widget List

This section is defined between Begin WidgetList and End WidgetList. Each line of the Widget List specifies the behavior of each widget, in the following format: widget_name = widget_function(vars). Widget names take the form Wr,c, where r and c are row and column indices (e.g. W1,1 for first row and column). The following is the list of NIREP Display Widget functions:

1. View(image, title) 2DViewer that displays the image specified in the variable image, with widget title title. The image data can be viewed from three different image orientations: Sagittal, Coronal and Transverse.

   image - Input image variable. The variable should either be defined in the Evaluator List or the Variable List, and should point to Image or ObjectMap data.

title - Title of the widget. Variables from the Variable List can be used using the $var operative.

e.g. View(_A1, Image 001 MRI) View(img, Image $image $modality)

2. Checkerboard(image1, image2, title) Displays two images in a checkerboard grid for comparison. All three image orientations supported by the View() widget are supported.

   image1 - First input image variable. The variable should either be defined in the Evaluator List or the Variable List, and should point to Image or ObjectMap data.

   image2 - Second input image variable. The variable should either be defined in the Evaluator List or the Variable List, and should point to Image or ObjectMap data.

   title - Title of the widget. Variables from the Variable List can be used using the $var operative.

   e.g. Checkerboard(_A1, _A2, Checkerboard Image 001 CT & Image 002 CT)

3. Wipe(image1, image2, title) Displays two images in a wipe widget for comparison. All three image orientations supported by the View() widget are supported.

   image1 - First input image variable. The variable should either be defined in the Evaluator List or the Variable List, and should point to Image or ObjectMap data.

image2 - Second input image variable. The variable should either be defined in the Evaluator List or the Variable List, and should point to Image or ObjectMap data.

title - Title of the widget. Variables from the Variable List can be used using the $var operative.

e.g. Wipe(_A1, img, Wipe Image 001 CT & Image $image$modality)

4. ObjectMap(objmap, title) 2D object map viewer with colorized object regions. All three image orientations supported by the View() widget are supported.

objmap - Input object map variable. The variable should either be defined in the Evaluator List or the Variable List, and should point to ObjectMap data.

title - Title of the widget. Variables from the Variable List can be used using the $var operative.

e.g. ObjectMap(_A3, ObjectMap 001)

5. OverlayObjectMap(image, objmap, title) 2D image viewer with colorized object map overlay. All three image orientations supported by the View() widget are supported.

image - Input image variable. The variable should either be defined in the Evaluator List or the Variable List, and should point to Image data.

objmap - Input object map variable. The variable should either be defined in the Evaluator List or the Variable List, and should point to ObjectMap data.

title - Title of the widget. Variables from the Variable List can be used using the $var operative.

e.g. OverlayObjectMap(_A1, _A3, ObjectMap Overlay 001 CT)

6. OverlayImage(image1, image2, title) 2D image viewer with second image overlay. All three image orientations supported by the View() widget are supported.

   image1 - First input image variable. The variable should either be defined in the Evaluator List or the Variable List, and should point to Image data.

   image2 - Second input image variable. The variable should either be defined in the Evaluator List or the Variable List, and should point to Image data.

   title - Title of the widget. Variables from the Variable List can be used using the $var operative.

   e.g. OverlayImage(_A1, _A2, Image Overlay 001 CT 002 CT)

7. RelativeOverlap(objmap1, objmap2, title) 2DViewer showing the relative overlap difference between two object maps. All three image orientations supported by the View() widget are supported.

   objmap1 - First input object map variable. The variable should either be defined in the Evaluator List or the Variable List, and should point to ObjectMap data.

   objmap2 - Second input object map variable. The variable should either be defined in the Evaluator List or the Variable List, and should point to ObjectMap data.

title - Title of the widget. Variables from the Variable List can be used using the $var operative.

e.g. RelativeOverlap(_A3, _A4, Relative Overlap 001 002)

8. BlankWidget() The BlankWidget is used to specify that there is no display widget for a particular panel location in the grid. A view widget will be displayed without any inputs. With being a view widget, a popup context menu can be activated with a right click to bring up the standard options for a widget. This context menu will allow the user, among other things, to replace the view with another widget.

2.2.5.5 Evaluator List

This section is defined between Begin EvaluatorList and End EvaluatorList. Each line of the Evaluator List specifies evaluator functions, some of which may not be in use by any of the widgets, in the following format: evaluator_var = evaluator_function(vars). The following is the list of NIREP Evaluator functions:

(a) SpatialData(cs_id, label)

(b) Transformation(cs1_id, cs2_id, algorithm)

(c) Diff(image1, image2)

(d) XDisp(transform)

(e) YDisp(transform)

(f) ZDisp(transform)

(g) Jacobian(transform)

(h) TransformImage(image, transform, interpolation) [interpolation currently not implemented]

(i) InvertTransformation(transform)

(j) InverseConsistencyErrorImage(fwd_transform, rev_transform, method)

(k) TransitivityErrorImage(transform12, transform23, transform31, method)

(l) IntensityVarianceTramsform(refimage, images, label, algorithm)

(m) IntensityVarianceImage(images, label)

CHAPTER 3
IMPLEMENTATION

The NIREP software is implemented in C++ using ITK, VTK, wxWidgets and vtkINRIA3D. ITK was selected for the numerous file formats that can be read in and for the extensive amounts of algorithms for image analysis. VTK was selected for displaying images. KWWidgets was originally used for the GUI. However the creation of GUIs was too hard, too slow and didn't provide as nice of a GUI. wxWidgets replaced KWWidgets because there is a GUI builder, the GUI looks nicer and wxWidgets is widely used. Some of the previous code from KWWidgets was re-used and the concept used with KWWidgets was expanded. vtkINRIA3D was selected for the extension of VTK such as locking of render windows, display of cross hairs and display of image information. The following sub chapters will talk about the four specific main classes, Data Manager, Display Manager, Evaluator and Interface, and their supporting classes.

3.1 Data Manager

The Data Manager is responsible for managing all image and metric data within NIREP. This includes responsibility for handling queries from other components for access to various types of data, as well as memory management functions such as dynamic movement of images between memory and disk and compression/decompression of data. The Data Manager is implemented to retain the results of computationally intensive operations on data elements in order to avoid

re-computation of these operations to satisfy future requests. The DataObject class is a polymorphic superclass that stores the in memory data - this means that data classes will inherit from DataObject and implement certain functions. By having a polymorphic superclass all data have a common alias, the data is stored in a quick lookup map, and the data is passed around as a single entity instead of as many data types as supported in NIREP. The ITK software library is used for reading in most of the data. When ITK can not read in the data, for example certain transformation co-efficient files, the derived class of DataObject will be responsible for reading in the data.

3.2   Display Manager

The Display Manager is a GUI, based on wxWidgets frame class, and a manager of panels and labels. The manager is implemented to place panels and labels into a grid format (see Figure 2.7) based on a Display Description, keep track of which panels are located in the grid, link the panels, turn off and on the titles, add panels and delete panels. The GUI part displays menu options to take care of opening a new Display Manager, to open a Display Description, to turn on and off the cursors, to turn on and off the titles, to edit a panel, to lock the cursors between panels, to change the variable content of a Display Description and to bring up a standard evaluation display Single Algorithm Comparison. A new instance of the Display Manager is created every time the user opens a new display - either from the Interface or from a Display Manager. The Display Manager relies on a lot of supporting classes to help with the interaction with the user.

3.2.1   Supporting classes

Each of the following sections discusses the supporting classes that the Display Manager uses. Some of the supporting classes are GUI's, based on wxWidgets, and some are views based on vtkINRIA3D and VTK, that are displayed within the Display Manager.

3.2.1.1   Checkerboard

The Checkerboard view widget is implemented by extending NIREPvtkViewImage2D to support two images and to use VTK's Checkerboard Widget. VTK's Checkerboard Widget is implemented where if a user wants they can slide the sliders of the border located on the top and bottom and to the right and left of the image to change the number of rows and columns in the checkerboard. The checkerboard is used to find where two images do not align properly. If two images align properly then the two images will become one image (see Figure 3.1), whereas if the two images do not align properly then the checkerboard format appears in the view (see Figure 3.2 and Figure 3.3).

3.2.1.2   EditTextPanels

The EditTextPanels is a GUI that, for those widgets that are displaying text, allows a user to make changes to the font. A user can change the font by clicking on a wxFontPickerCtrl button, which pops open a GUI with all the changes that a user can do to the font. As well, a button is displayed that, when clicked, brings up a PanelForm to make changes to the panel.
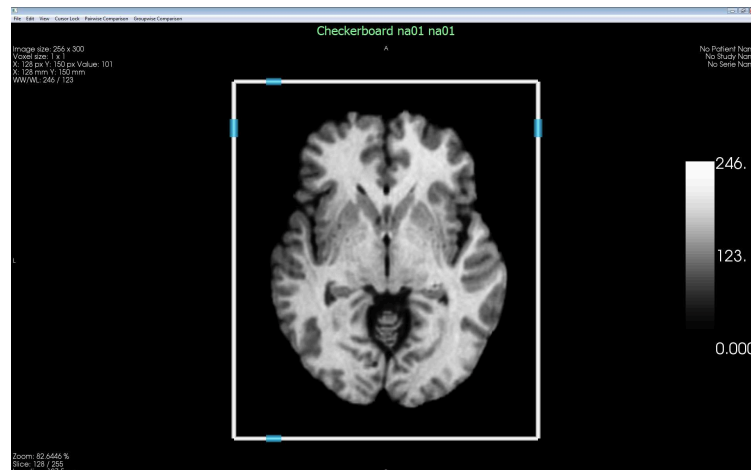
Figure 3.1: Checkerboard view widget, of two images na01 and na01. This is showing when two images align perfectly, the checkerboard view widget will look like one image.
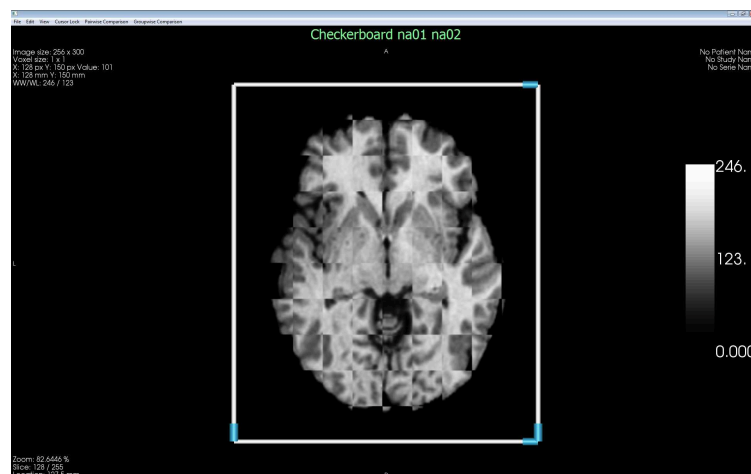
Figure 3.2: Checkerboard view widget of two images na01 and na02. This is showing two images being evaluated, using the Checkerboard view widget, before registration. As can be seen along the borders, the two images do not align perfectly.

Figure 3.3: Checkerboard view widget of two images na01 and demons na01_def_na02. This is showing one image and one transformation being evaluated using the Checkerboard view widget. As can be seen along the borders, the two images do not align perfectly.

3.2.1.3 GetPanelFromUser

The GetPanelFromUser is a GUI that is opened up from a Display Manager menu option. This allows a user to select which panel they want to modify. Depending on the type of panel chosen either a EditTextPanels GUI or a PanelForm GUI will be opened.

3.2.1.4 Grid

The Grid is a view that allows a user to view a transformation in a grid layout. A normal grid is first created and then transformed based on the user chosen transformation. This helps to show how the transformation deforms horizontal and vertical lines.
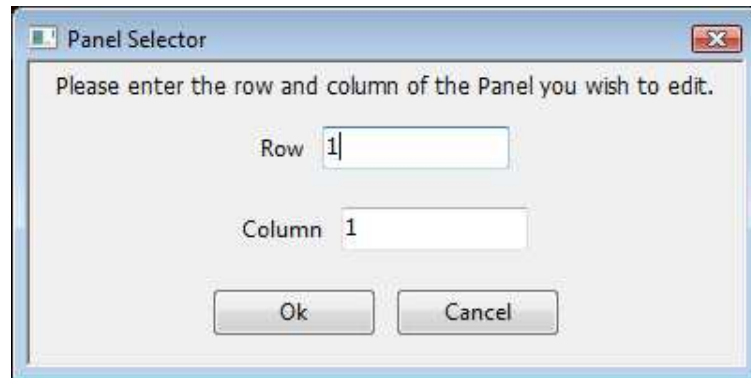
Figure 3.4: Panel selector GUI within NIREP. The panel selector GUI allows a user to select which panel they want to modify by putting in the row and column of the panel.
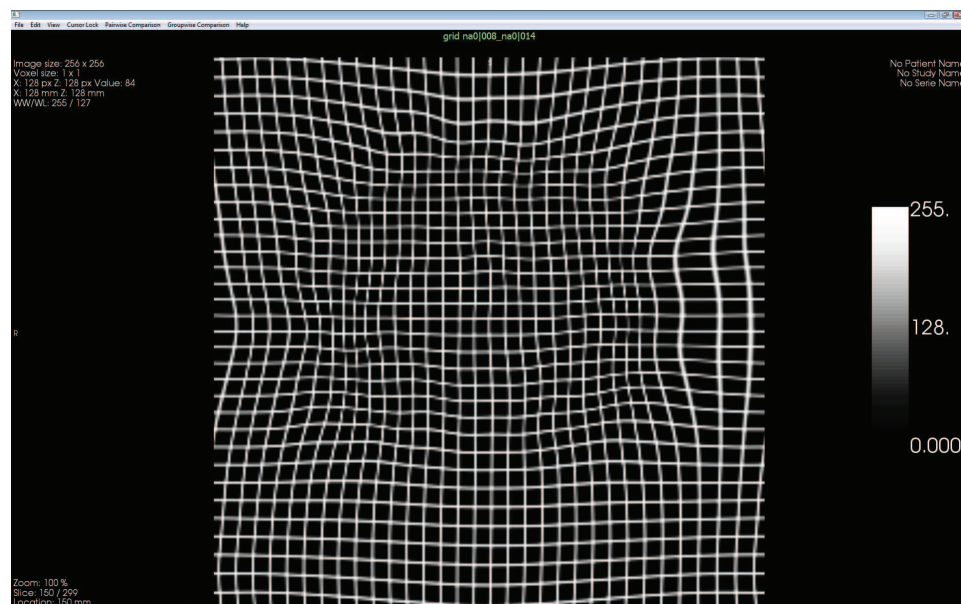
Figure 3.5: Grid view widget within NIREP. The grid shows how a registration transforms straight lines.
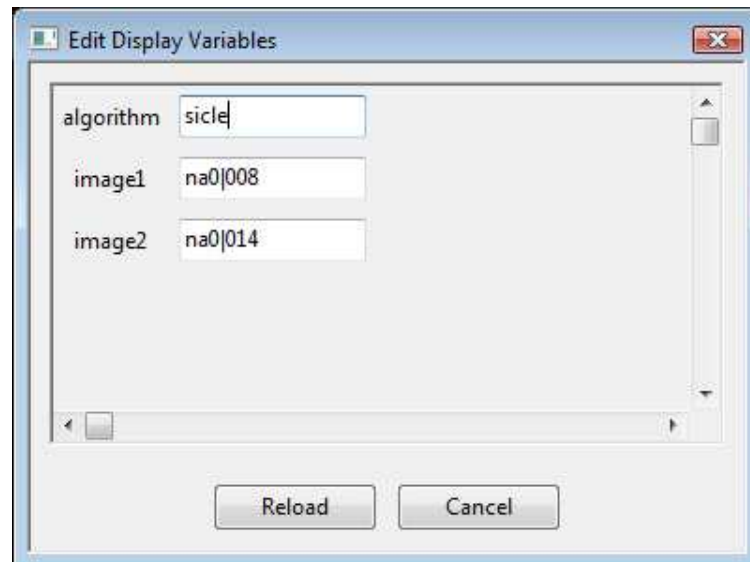
Figure 3.6: Edit Display Variables GUI within NIREP. The Edit Display Variables GUI allows a user to edit the values associated with a variable within a Display Description.

3.2.1.5 NIREPEditProjectFile

The NIREPEditProjectFile is a GUI that allows a user to edit the variables in the display description (See 2.2.5.3 for further details on variables). Once a user is done with editing the variables they can click on reload, which will reload the whole entire display description with the new variables. This is useful to change data quickly without having to add in new evaluator commands or manually changing evaluator commands.

3.2.1.6 NIREPvtkInteractorStyleImage2D

NIREPvtkInteractorStyleImage2D is an extension of vtkInteractorStyleImage and modified from vtkINRIA3D's vtkInteractorStyleImage2D. The modifications: included Panel so that popup menus are possible and commented out the items for right

clicking and put in a call to the panel to deal with right clicking. Also renamed the class from vtkInteractorStyleImage2D to NIREPvtkInteractorStyleImage2D so that the changes are kept but not have vtkINRIA3D in our source tree.

3.2.1.7 NIREPvtkViewImage

NIREPvtkViewImage is modified from vtkINRIA3D's vtkViewImage. As well, NIREPvtkViewImage is the base class for 2D/3D image viewers, see NIREPvtkViewImage2D for the 2D image viewer. The modifications: changed the variable ScalarBar from being a vtkScalarBarActor* to a vtkScalarBarWidget* which meant that the get and set functions were changed.

3.2.1.8 NIREPvtkViewImage2D

NIREPvtkViewImage2D is an extension of NIREPvtkViewImage2D and modified from VTKINRIA3D's vtkViewImage2D. NIREPvtkViewImage2D is the main class that is passed around in NIREPDisplay. The modifications: changed the inherited class from vtkViewImage to NIREPvtkViewImage.

3.2.1.9 NIREPvtkViewImage2DCommand

NIREPvtkViewImage2DCommand is an extension of vtkCommand and modified from vtkINRIA3D's vtkViewImage2DCommand. The modifications: this file contains references to vtkViewImage2D and vtkInteractorSytleImage2D but those files were brought into NIREP with modifications so the references were changed to NIREPvtkViewImage2D and NIREPvtkInteractorSytleImage2D.
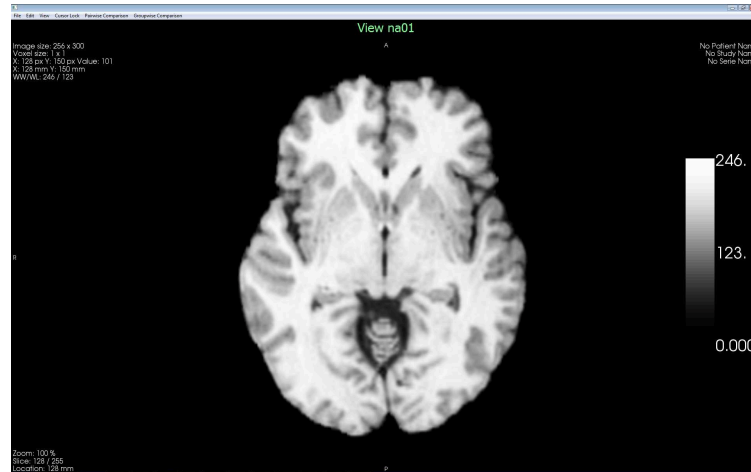
Figure 3.7: View widget. In this figure the na01 brain image is being displayed in the Coronal orientation.

3.2.1.10   ObjectmapForm

The ObjectmapForm is a GUI that is opened from a user right clicking on an image and selecting "Object Map" and takes care of changing the different aspects of an object map. The aspects are: turning off and on the entries, changing the names, changing the color and changing the number of shadows. When a user turns off or on the entries they will have to scroll through the image to make the changes happen.

3.2.1.11   PairwiseComparison

The PairwiseComparison (Figure 3.8) is a GUI that allows a user to select the inputs for a pre-defined pairwise comparison display. The intention is to use a predefined display description that tells how the display should be laid out, however the user is asked for the algorithm, modality, image1 and image 2 they want to use to do the comparison. Then the information will be passed to the Display Manager
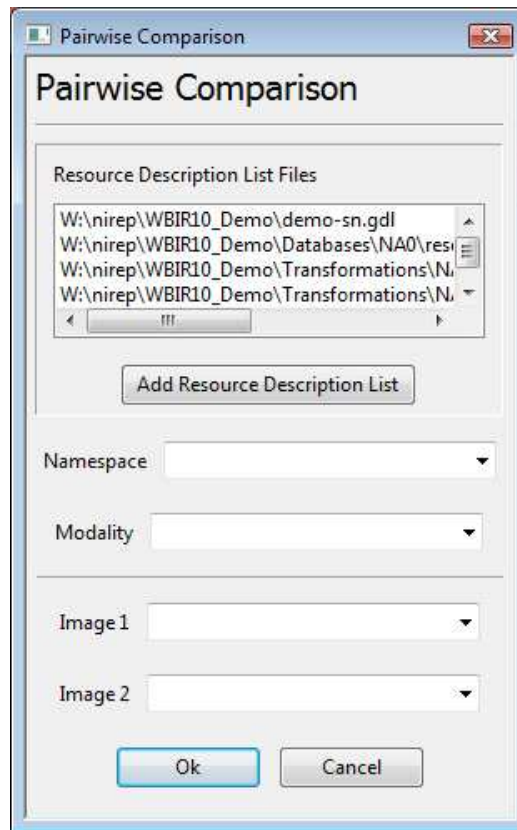
Figure 3.8: PairwiseComparison GUI. The PairwiseComparison GUI allows a user to select an RDL file, the namespace, modality, image1 and image2. When the user hits Ok, a 3x3 grid will appear allowing the user to evaluate the two images using the difference.

to take care of creating the display. The display created is a 3x3 grid, with column 1 being image 1, column 2 being the difference of image 1 and image 2, column 3 being image 2, row 1 being the transverse orientation of the views, row 2 being the sagittal orientation of the views and row 3 being the coronal orientation of the views.

3.2.1.12 Panel

The Panel is an extension of wxPanel and is an individual cell in the grid in the Display Manager. The panel takes care of managing the cell within the grid. This

includes a view widget (see 2.16), a render window (that the view widget is rendered in) and pop up menus (when the user right clicks in the panel). As well, the panel makes calls to the Evaluator asking for the specified data to be displayed. The view widget and data is specified in the Display Description.

3.2.1.13   PanelForm

The PanelForm is a GUI that changes the different items of the panel to the users desire. To bring up PanelForm, either right click in the Panel and choose 'Edit m_widget' or go to the menu option 'Edit->EditPanel..', in the Display Manager, and put in the row and column of the panel. The items are: the view widget, the orientation, the inputs, linking widgets together, changing the color map, changing the title and changing the font of the title. When the user has chosen all the options, they hit Ok and the Panel creates everything the user chose.

3.2.1.14   VectorField

The VectorField is a view widget that allows a user to view a transformation as a vector field. To create the vector field, a grid of points is created and then set as the starting points, and then figure out the corresponding transformation points which are set as the end points. The number of points used, the spacing and length of the vector can be specified by the user.

3.2.1.15   Wipe

The Wipe view widget is implemented by extending NIREPvtkViewImage2D to support two images and to use VTK's Rectilinear Wipe Widget. VTK's Rectilinear

Figure 3.9: PanelForm GUI. The PanelForm GUI allows the user to make changes to the current Panel. The user can can change the view/widget, the orientation, the input images, the color map, the title of the Panel, and even the attributes of the title.
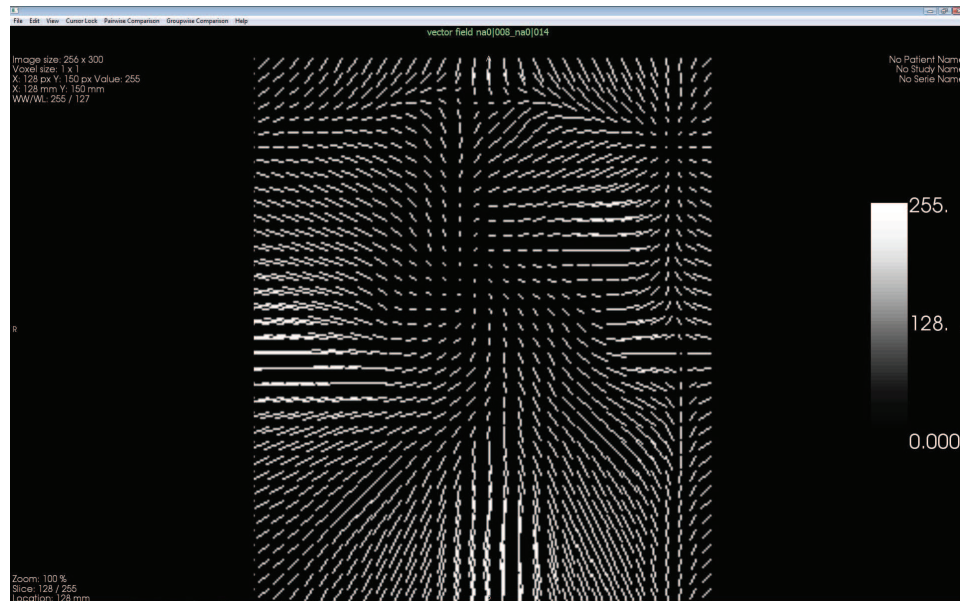
Figure 3.10: VectorField view widget. The VectorField shows what a transformation looks like by using vectors to describe how the points in a grid are moved.

Wipe Widget is implemented as a 2x2 checkerboard and if a user wants they can grab the divider between the checkerboard cells and wipe across the image. As with the checkerboard, the wipe helps to see if the two image align properly. If the two image align perfectly only one image will be displayed but if they do not align perfectly then the differences can be seen as the user wipes across the image.

3.2.1.16 wxVtkImageFlip

The wxVtkImageFlip is a GUI that has been modified from vtkINRIA3D, to get rid of memory issues, and is a dialog window that helps flipping and rotating a 3D Image to correct geometry problems. The GUI takes a VTK image as input and pops up the 3 orientation views - sagittal, coronal and transverse. The image can
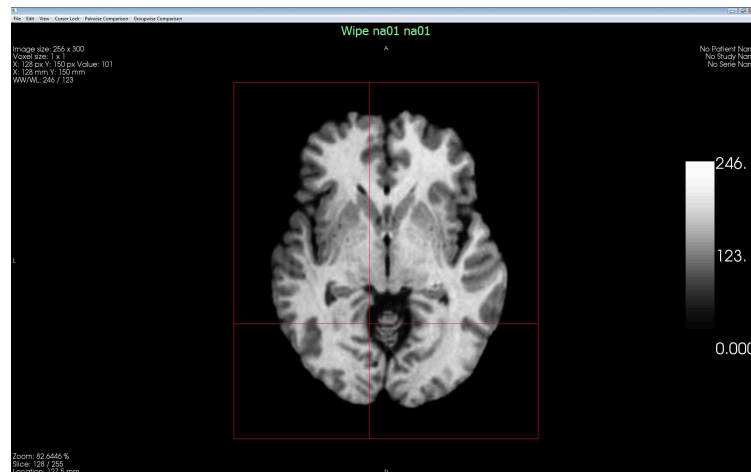
Figure 3.11: Wipe view, of two images n01 and na01. This is showing when two images align perfectly, the Wipe view widget will look like one image. The user is allowed to wipe across to find where the two images do not align along the borders.
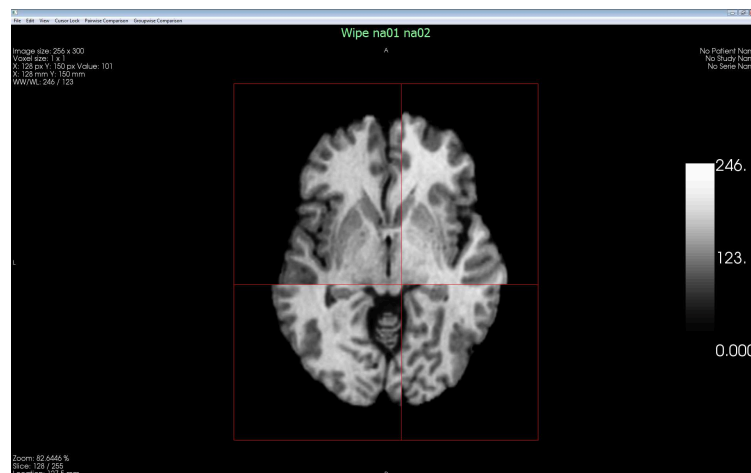
Figure 3.12: Wipe view widget of two images na01 and na02. This is showing two images being evaluated, using the Wipe view widget, before registration. The user is allowed to wipe across to find where the two images do not align along the borders. As can be seen along the borders, the two images do not align perfectly.
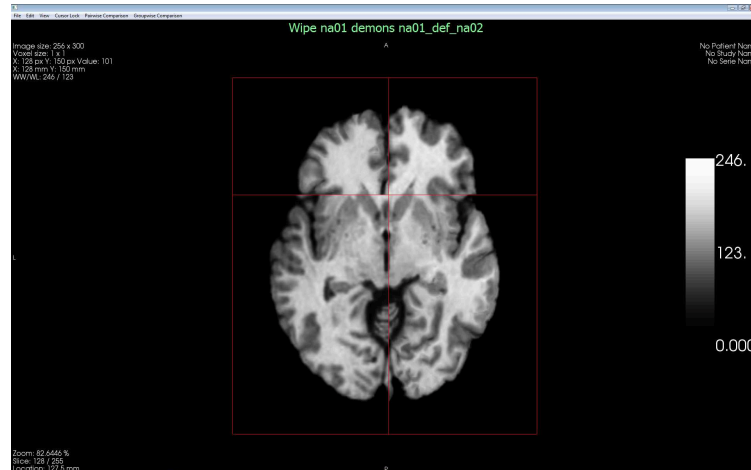
Figure 3.13: Wipe view widget of two images na01 and demons na01_def_na02. This is showing one image and one transformation being evaluated using the Wipe view widget. The user is allowed to wipe across to find where the two images do not align along the borders. As can be seen along the borders, the two images do not align perfectly.

be flipped into the 3 different orientations. Rotations are possible by choosing an acquisition flag. The modifications: inside of the function SetImage (vtkImageData* image) the this->UpdateReslicer(); was moved and the this->Reslicer->Update(); was added in so that the View1->SetImage would stop complaining about the Extents of the images being 0,-1, etc. When the user hits ok, the image gets updated with the changes.

3.3   Evaluator

The Evaluator is a creator of data and intermediate step for asking the Data Manger for data. All the statistics and deformations are created in the Evaluator. Before the Evaluator creates data, it asks the Data Manger if it has the data because some deformations or statistics are based on previous data that may be in the Data
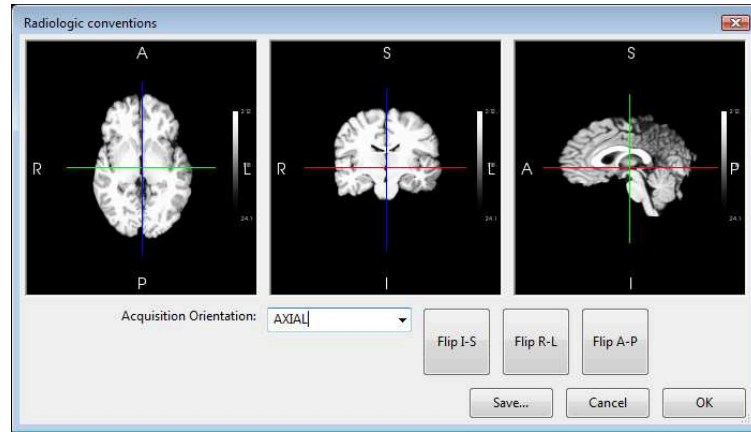
Figure 3.14: Image Flip GUI. The Image Flip GUI allows users to change the acquisition orientation of the image, flip the images based on I-S, R-L, and A-P.

Manger. Evaluator Commands are used to figure out what operations are to be performed. See appendix for the full list of Evaluator Commands. The Evaluator relies on supporting classes to help with the creation of statistics and deformations.

3.3.1 Supporting classes

Each of the following sections discusses the supporting classes that the Evaluator uses. Each statistic and deformation has its own class which helps with integrating new statistics and deformations into NIREP. All the deformations are in the gecLibs library and all the statistics are in NIREP.

3.3.1.1 NIREPDiceCoefficient

The alignment of objects, structures, organs, regions of interest (ROIs), etc., are a good indicator of how well two images are registered. These subvolumes are defined by partitioning or segmenting an image into objects or ROIs. The Dice Coefficient[11, 15, 5, 6] figures out the region of overlap by taking twice the intersection

over the union of the segmented region of the deformed and target volumes.

The Dice Coefficient is computed using equation 2.11.

3.3.1.2 NIREPIntensityVariance

The Intensity Variance measures the similarity between a population of images based on voxel intensity difference. In image registration applications driven by voxel intensity features, the ideal registration should result in zero voxel intensity difference between the registered images. The variance takes in a list of images that are used to compute the variance.

The Intensity Variance is computed using equation 2.10.

3.3.1.3 NIREPInverseConsistencyError

The Inverse Consistency Error[2, 3, 1] measures the consistency of the correspondence defined by forward and reverse transformations between two coordinate systems. In an ideal case, the forward transformation should equal the inverse of the reverse transformation. There are two type of Inverse Consistency Errors (ICE) computed, equation 2.6 and equation 2.7.

3.3.1.4 NIREPRelativeOverlap

The alignment of objects, structures, organs, regions of interest (ROIs), etc., are a good indicator of how well two images are registered. These subvolumes are defined by partitioning or segmenting an image into objects or ROIs. The Relative Overlap[12, 13, 8, 5, 6, 11] assesses how well two equally likely segmentations of the same region of interest (ROI) agree or disagree with each other. Ideally, RO of all

regions should be 1.0.

The relative overlap is computed using equation 2.12.

3.3.1.5 NIREPStatistics

The NIREPStatistics is an adapter class that all NIREP statistics classes inherit from. All classes should implement the function ComputeStatistic.

3.3.1.6 NIREPTargetOverlap

The alignment of objects, structures, organs, regions of interest (ROIs), etc., are a good indicator of how well two images are registered. These subvolumes are defined by partitioning or segmenting an image into objects or ROIs. The Target Overlap[11, 5, 6] assesses the intersection between two similarly labeled regions in S and T divided by the volume of the region in T, where || indicates volume computed as the number of voxels.

Assume S_i and T_i are defined as the ith segmented region in the deformed source and target volumes.

The target overlap in equation form:

$$TO_i(S_i, T_i) = \frac{|S_i \cap T_i|}{|T_i|} \tag{3.1}$$

3.3.1.7 NIREPTransitivityError [3, 10, 7]

The Transitivity Error (TE) measures the consistency of the correspondence defined by compositions of transformations. In an ideal case, the compositions of the transformations should result in an identity transformation. The Transitivity Error

is computed using equation 2.8.

3.4 Interface

The Interface is the first GUI presented to the user and initializer of the Evaluator and DataManager. The GUI part displays the in memory data in a grid layout and has menu options:

- File

    - Load Resource Description List

    - Save Resource Description List

    - Edit Resource Description List

    - Edit Display

    - Quick View

    - Load Display

    - Exit

- Help

    - Help

The Interface is responsible for initially loading in a display description, allowing a user to create a display description and showing and allowing users to release data. The Interface is the controller of the Evaluator and DataManager and when the Interface is closed, NIREP ends.

3.4.1 Supporting Classes

3.4.1.1 wxNirepApp

Is the startup file that wxWidgets requires. This will create the Interface which is the first interactive screen the user will see. Also, this will display a splash screen when NIREP is started. Finally, this file allows a user to double click an nd3 file and have NIREP either start and create the display or if NIREP is already running, create the display.

3.4.1.2 QuickView2

The QuickView2 is a GUI that allows a user to create a display without having to manually write a Display Description. The user can change the number of rows and columns, which will in turn modify the grid that is being displayed. The grid that is being displayed corresponds to the final grid with the different views. From there the user can go through each location in the grid and choose the view they want displayed. When a user chooses a view, then the possible inputs to the grid are displayed. The possible inputs are: load in files that the user wants or a creation of evaluator commands. There is an advanced area that the user can make changes to the RDL that is being created. When the user is ready, they click on Show to bring up the newly created display.

3.5 Display Description

The Display Description contains the classes that pertain to the structure of the display description. See the previous chapter for the picture of the structure.

Some of the files that the Display Description reads in are XML files, which is actually done by TinyXML.

- BasicDescription

    - This class holds descriptive information about namespaces, algorithms, and coordinate systems. Object of this class are filled with information by parsing a resource description list. All non-required tags and their values are stored in the attribute list.

- DisplayAttributes

    - This class holds the attributes of a display. These would be the number of rows, columns, locking the cursors, turning on and off the cursors, etc.

- DisplayDescription

    - This class holds all the information that is in the display description file and RDL. There are other classes which hold the specific information, but this class holds references to those other classes. By holding the references, we can pass this class around and have all the information about a display.

- EvaluatorList

    - Evaluator is responsible for maintaining the evaluator list (human-readable), stored as a map.

- ResourceDescription

– This class holds information about the resources stored in the database. Each resource must contain a unique resource id and a list of data descriptions that describe the data (images, object maps, contours, landmarks, etc.) associated with the resource. Optionally, a resource can contain a list of non-required user defined attributes such as age, race, handedness, etc.

- ResourceDescriptionList

    – This class keeps a list of the data files available to be loaded into memory. This class, lists the data descriptions of images, object maps, landmark files, and contour files. This class creates a map of ResourceDescription objects by parsing a resource description file.

- VariableList

    – Stores the variables in the display description and replaces variables with values.

- WidgetList

    – Stores the widget commands in a vector. The location in the vector corresponds to the location in the display grid. Also will be able to return the widget type and the parameters of the widget command.

CHAPTER 4
FUTURE AND SUMMARY

NIREP is not fully implemented which will require future features to be incorporated into the program. There are new statistics that will need to be added in, new views, ranking of the different registration algorithms and new readers for deformation files.

The addition of new statistics will be relatively easy because of the framework built into NIREP. To add in new statistics, first create a new class that inherits from NIREPEvaluatorTask. From there override the function Evaluate. Then, a new evaluatorCommand name will be added into the map EvalCmd in NIREPDefinitions.h. Also, in Evaluator.cpp, in the constructor, there needs to be an addition of `m_EvaluatorTaskList[EvalCmd["evaluatorCommandName"]] = new NewClass(this);}` where `evaluatorCommandName` will need to be changed to the evaluatorCommand used in the addition to NIREPDefinitions.h and `NewClass` needs to be changed to the class name of the new statistic.

To add in new visualization views, first create a new class that inherits from NIREPvtkViewImage. The new file will probably have some changes done to SetImage but it depends on what the new view is going to do. To be able to use the new view, then inside Panel.cxx, in the function UseWidget, the if statement will have to be modified to use the new view.

Currently NIREP does not have any ranking of the results of the evaluation

methods. There are plans on different ways that the results of the evaluation methods can be ranked. However, so far, none of the plans have been implemented in NIREP.

There was not enough time to add in graphs. There needs to be a study of the different open source graph programs - wxWidgets graphs, VTK graphs, C++ library's and independent programs. The best program will be able to be viewed within the grid with all the different types of graphs.

With the help of CPack and NSIS there have been some installers created to test how well NIREP can be distributed on Window's machines. Well doing the tests, it was found that some of the instructions feed to CPack needed to be changed. With the addition of having dynamically created menu's, it was found that there needed to be files moved from the source directory into the installer. NIREP has been tested for creating NIREP from source but to this point there have not been any tests for binaries or packages for Unix or Macintosh.

NIREP started off as a concept to solve a problem, that was then designed and finally implemented. The program has been created as a tool for users to evaluate non-rigid image registration algorithms. The design of NIREP is conducive for the add-on of evaluations, views and data. The implementation has met most of the specifications using C++ and the libraries ITK, VTK, vtkINRIA3D and wxWidgets. Finally, the project will continue on and be improved with the help of the community. The evaluation of non-rigid image registration algorithms will benefit greatly with the addition of NIREP.

APPENDIX A
NIREP USER MANUAL

NIREP User Manual

1.0

Generated by Jeffrey A. Hawley

July 18, 2011

A.1 Overview

Non-rigid Image Registration Evaluation Program (NIREP) is a standalone image registration evaluation software that offers users high customizability of image registration evaluations and presentation interface. NIREP offers standardized set of image registration evaluation statistics and tools that serve as benchmarks for image registration performance. Users of NIREP are able to configure the behavior and arrangement of the software display such that comparisons of multiple image registration algorithms can be made simultaneously, both qualitatively and quantitatively. Built-in capabilities to read and write images, transformations and supplementary data such as landmarks and object maps, with the ability to perform advanced operations such as deformations, Jacobians, inverse of transformations, relative overlap and intensity variance, and the wide array of evaluation tools such as checkerboard, wipe, object overlay, charts and graphs make NIREP a self-contained tool to perform all that are necessary to evaluate image registration algorithms. Human-readable configuration files that allow users to "script" evaluation projects enable users to batch process large evaluation projects. NIREP supports not only all major image formats supported by ITK, but also supports AnalyzeTM Object Maps and various transformation formats.

A.2 Background

Evaluating non-rigid image registration performance is a challenging task because of the lack of ground truth to correlate with. The current image registration evaluation scheme either involves users to submit their registration results to the

party responsible for evaluating the registrations, or have the evaluator perform all tasks from registration to evaluation. Such schemes are cumbersome and inconvenient because users often have to convert their data to the format required by the evaluator, and because users do not have immediate access to evaluation results. Non-rigid Image Registration Evaluation Project (NIREP) has been initiated to address this challenge, giving access to researchers the most comprehensive analysis of non-rigid image registration performance to date at the convenience of their own workstations, through the NIREP software. Non-rigid Image Registration Evaluation Program (also, NIREP from here on) is a publicly available software package that evaluates image registration performance based on the image data, supplementary data (e.g. landmarks, segmentations, etc.) and transformations generated by image registration algorithms. NIREP contains all necessary features to read and evaluate image registration data, and output evaluation reports in different formats such as images, charts and graphs. The public accessibility of NIREP not only standardizes image registration evaluations, but also, effectively distributes workload to individual users.

A.3 System Requirements

The NIREP software is currently supported only on 64-bit Windows platforms with at least 4GB of RAM. Images and transformations are read into memory during runtime and thus have high memory requirements. For example, suppose you want to display a source, target and transformed image of size 256x256x256. You will have to load 8-bit source and target images which both require 16MB and a transformation which will require 12 times this amount. In addition, you will compute a deformed

image. For all this, the software requires at least 240MB of memory plus another 240MB of memory for overhead. Now, if you want to compare a second algorithm, you will require double the amount of memory. In the future, the amount of overhead storage will be dramatically reduced. Typically, you will want to perform more advanced operations and have more comparisons on the screen. Therefore, having at least 4GB of RAM is advisable.

A.4 Acquire NIREP software

Download the NIREP installer from www.nirep.org/download/. Default install location of the NIREP software is C:\Program Files\NIREP.

A.5 Before you use NIREP

The goal of using the NIREP software is to evaluate image registration performance of one or more registration algorithms. First, you need to gather together the image volumes that you want to use as your evaluation database. You can use your own set of image data or download images from websites such as www.nirep.org. Keep in mind the images should be in a file format that is supported by NIREP. NIREP supports many different image formats. NIREP is written using ITK and therefore supports all ITK supported data formats, plus additional formats (See Supported Data Formats).

After gathering the data, you have to decide which image registration algorithms you want to evaluate. This could be one that you wrote yourself or downloaded from external sources such as www.na-mic.org. Once the image registration

algorithms are installed, registrations should be performed on the evaluation database prepared in the previous step. Registrations in the forward and reverse direction (i.e. source to target and target to source) are highly recommended because this enables the evaluation of inverse consistency properties of the algorithms used. Furthermore, registrations of all permutations of the evaluation database (e.g. 240 registrations for a dataset of 16 images) are strongly recommended for transitivity analysis. NIREP currently supports the following types of transformations: 1) displacement field images with scalar floating-point voxels containing displacements for each component of the image space, 2) single displacement field image with vector-valued floating-point voxels containing displacements for all components of the image space, 3) SICLE coefficient file, and 4) AIR warp file. The displacements can be in physical space units (mm), image space units (voxels) and unit cube (1/voxels). In the future, NIREP will support more varieties of transformation formats.

When NIREP is first started, the Interface screen is displayed. From here, a user has a number of choices to display data. The user can choose to select menu option File and choose between Load Display or Quick View to get to a Display or the user can choose to select to Edit a Display or the user can choose to edit or create an RDL file (see section blah for more information) by selecting Edit Resource Description List. The Load Display option will load in an ND3 file (see section blah for more information) and display what is described in the ND3 file. Quick View will allow a user to create a Display quickly. (see section blah for more details)

A.6 Quick View

A.7 Edit Resource Description List

The Resource Description List (RDL) is one of two files that NIREP requires to evaluate image registration. The Edit Resource Description List tries to edit or create an RDL based on what the user wants.

To get to the Edit Resource Description List, follow these steps:

1. If NIREP is not already started, start the NIREP software by double-clicking on the NIREPApp.exe executable.

2. In the Interface window go to the menu option File -¿Edit Resource Description List.

Now that we have started the Edit Resource Description List, here are some sample steps that a user can perform.

For regular, atomic, non transformation files:

1. Select Datatype to be "image."

2. Click on Browse to select any one of the images in the evaluation database. The file name will then appear in the Filename pattern 1 box, ready to be edited for templating.

3. Suppose the image files for this evaluation database are named na01.hdr, na02.hdr, ..., na16.hdr. Then the Filename pattern 1 should be modified to "naa.hdr," and you will notice the variable "a" appearing in the Variable list.

4. Type all variable values (e.g. "01,02,03" for files named "na01.hdr," "na02.hdr" and "na03.hdr") into the Values box for Variable "a." This way, RDL entries for all of the image data will be generated.

5. Now, click on Associated Variable for Coordinate System 0, and type in variable "a" as the associated variable.

6. The next thing to do is to define the Default namespace for the image set. A namespace defines the evaluation database association of each image data. For this example, let's name the default namespace as "na0."

7. The final entry to be completed is the Label, which is used to identify what the image data contains (e.g. CT, MRI, ObjectMap, etc.) For this example, let's enter "MRI" as the Label.

8. The Default DMID (DMID stands for Data Manager ID, which is a unique ID within the NIREP software to associate with each image data) should be "spatialdata(na0—a,MRI)," and everything is ready for RDL file generation. If you click on Generate, a ViewRDL window will pop up and the output will look like the following:

```xml
<?xml version="1.0" ?>
<ResourceDescriptionList namespace="na0">
    <data_description label="mri" dmid="spatialdata(na0|01,mri)">
        <coordinate_system index="0" ns="na0" id="01" />
```

```
        <algorithm index="0"></algorithm>

        <filename index="0">na01.hdr</filename>

        <datatype>image</datatype>

    </data_description>

    <data_description label="mri" dmid="spatialdata(na0|02,mri)">

        <coordinate_system index="0" ns="na0" id="02" />

        <algorithm index="0"></algorithm>

        <filename index="0">na02.hdr</filename>

        <datatype>image</datatype>

    </data_description>


    <data_description label="mri" dmid="spatialdata(na0|16,mri)">

        <coordinate_system index="0" ns="na0" id="16" />

        <algorithm index="0"></algorithm>

        <filename index="0">na16.hdr</filename>

        <datatype>image</datatype>

    </data_description>

</ResourceDescriptionList>
```

9. Click on Save to save this RDL file to the directory where the image files are

   location. For this example, the RDL file was saved in D:

   Data

   Databases

NA0

resources.rdl.

10. Optionally, if you want to add additional entries to the RDL file (such as object maps), you can repeat the process from step 2, changing the parameters accordingly, and copying and pasting to the existing RDL file (just make sure you do not have duplicate entries).

Next, you will need to generate RDL files for the transformations generated by registration algorithms:

1. select Datatype to be "transformation."

2. Select the appropriate transformation Format from the pull-down menu.

3. If only Filename pattern 1: Click on Browse to select any one of the transformations/displacement images in the algorithm output directory. The file name will then appear in the Filename pattern 1 box, ready to be edited for templating. If Filename pattern 1-3: For each Filename pattern N, click on Browse to select any one of the Nth component displacement images in the algorithm output directory. The file name will then appear in the Filename pattern N box, ready to be edited for templating.

4. Suppose the transformation files are named na01_To_na02.coeff, na02_To_na01.coeff, , na15_To_na16.coeff, na16_To_na15.coeff. Then the Filename pattern 1 should be modified to "naa_To_nab.coeff," and you will notice variables "a" and "b" appearing in the Variable list.

5. Type all variable values (e.g. "01,02,03" for transformations named "na01_To_na02.coeff," "na02_To_na03.coeff," etc.) into the Values box for Variables "a" and "b." This way, RDL entries for all permutations of transformations will be generated. To prevent identity map transformation entries (e.g. na01_To_na01.coeff) from being inserted into the RDL, check Remove Identity Maps.

6. Now, click on Associated Variable for Coordinate System 0, and type in the variable name (e.g. "a") associated with the first (i.e. source) coordinate system.

7. Then, click Add Coordinate to add Coordinate System 1, and type in the variable name (e.g. "b") associated with the second (i.e. target) coordinate system.

8. The next thing to do is to define the Default namespace associated with the transformation set. A namespace defines the evaluation database association of each image data. For this example, lets name the default namespace as "na0."

The display is the heart of displaying data to the user. Within the display are individual panels that control the specific type of widgets that the user displays. Currently each widget either displays a 2 dimensional or a slice of a 3 dimensional image or text. The types of widgets that a user can display are:

Rectilinear wipe widget

- Inputs: image1, image2, title

- Description: Displays two images in a 2x2 checkerboard style, with vertical and horizontal lines that can be wiped across the images.

  Checkerboard

- Inputs: image1, image2, title

- Description: Displays two images in a checkerboard style, with scrolls around the image to increase or decrease the number of tiles in the row(s) or column(s). The top and bottom scrolls increase or decrease the number of tiles for the columns. The scrolls on the left and right increase or decrease the number of tiles for the rows.

  View

- Inputs: image, title

- Description: Displays an image (see blah for the types of images).

  Objectmap

- Inputs: objectmap, title

- Description: Displays an Analyze Object Map.

  overlayObjectMap

- Inputs: image, objectmap, title

- Description: Displays an Analyze Object Map overlaid on top of an image.

overlayImage

- Inputs: image1, image2, title

- Description: Displays image1 overlaid on top of image2.

relativeOverlap

- Inputs: objectmap1, objectmap2, title

- Description: Displays the result of taking the relative overlap of objectmap1 with objectmap2.

relativeOverlapTextWidget

- Inputs: objectmap1, objectmap2, title

- Description: Displays the text result, sorted by objects, of taking the relative overlap of objectmap1 with objectmap2.

inverseConsistencyTextWidget

- Inputs: inverseConsistencyImage, objectmap, title

- Description: Displays the text result, sorted by objects, of taking the inverse consistency of the inverseConsistencyImage.

transitivityErrorTextWidget

- Inputs: transitivityErrorImage, objectmap, title

- Description: Displays the text result, sorted by objects, of taking the transitivity error of the transitivityErrorImage.

  varianceTransformText

- " Inputs: referenceImage, images, label, algorithm, title

- Description: Displays the text result, sorted by objects, of taking the transformation of the collection of images and finding the variance between them. The input "images" is a collection of images in the following syntax: "[image1;image2:image5;image8]". To further explain the collection, there must be "[]" and between them can either be the single name of an image or a range or images which is delimited by ":"

  varianceImageText

- " Inputs: images, label, title

- Description: Displays the text result, sorted by objects, of finding the variance of the input "images". The input "images" is a collection of images in the following syntax: "[image1;image2:image5;image8]". To further explain the collection, there must be "[]" and between them can either be the single name of an image or a range or images which is delimited by ":"

  blankWidget

- Inputs:

- Description: Displays a black widget.

All widgets have the same commands which are:

If the user wants to make global or in some cases individual changes to the display, they can use either the menu options or right clicking in each panel. The following are the menu options:

- File

    - Open NIREPDisplay Description

        * Description: Opens an ND3 file and display what is specified in the ND3 file.

    - Open new window

        * Description: Opens a new Display and leaving the current display alone.

    - Close

        * Description: Closes the current display.

- Edit

    - Edit Panel...

        * Description: Pops open a window that asks the user which panel they wish to edit and then brings up the appropriate editing window for the specified panel.

    - Edit Display Variables...

          ∗ Description: Pops open a window that displays the current variables being used by the ND3 file and allows a user to change the value of the variables on the fly and will update the current display with the changes. See blah for further information about variables in an ND3 file.

    – Add Row

          ∗ Description: Adds a row of blankWidget's to the end of the current display.

    – Add Column

          ∗ Description: Adds a column of blankWidget's to the end of the current display.

    – Turn Off Titles

          ∗ Description: Makes the titles disappear and the previously vacated space is filled by expanding the widgets.

    – Turn On Titles

          ∗ Description: Makes the titles appear and contrast the widgets to make room for the titles.

- View

    – Show Cursors

          ∗ Description: Turns on the vertical and horizontal cursors that are in the widgets.

- – Hide Cursors

  - ∗ Description: Turns off the vertical and horizontal cursors that are in the widgets.

- • Cursor Lock

  - – Lock All

    - ∗ Description: Locks all the panels together. For example when a user clicks in one widget, the location will be distributed to the other panels and be updated. Another example, when a user is changing the slice of a widget, all the other widgets will switch to the user selected slice.

  - – Lock Rows

    - ∗ Description: For each row in the display, lock all the panels within that row. See Lock All for what locking does.

  - – Lock Columns

    - ∗ Description: For each column in the display, lock all the panels within that column. See Lock All for what locking does.

  - – Unlock All

    - ∗ Description: Unlocks the panels so that each panel is the only panel that will be updated when a user does something in the panel.

  - – Unlock Rows

    - ∗ Description: Unlocks the rows so that each panel is the only panel that will be updated when a user does something in the panel.

- – Unlock Columns

  * Description: Unlocks the columns so that each panel is the only panel that will be updated when a user does something in the panel.

- Pairwise Comparison

  - – Single Algorithm Comparison

    * Description: A pre-defined ND3 file that the user is able to change the variables to the images that the user wants displayed. A single algorithm comparison is a 3x3

  - – Absolute Difference

  - – Window Wipe

  - – Checkerboard

  - – R, Y, G fusion

  - – Relative Overlap

- Groupwise Comparison

The following are the right click menu options:

- Edit m_widget

  - – Description: Allows the widget to be changed, change the color mapping, change the inputs

- Left Click

    - Pointer

        * Description: Allows the user to click on an image to gather the pixel value, the pixel location and also allows the user to scroll through the various slices of the image.

    - Level/Window

        * Description: Allows the user to changes the level/window of the image by scrolling up or down or left or right. The level/window is a way to adjust the contrast of the image.

    - Zoom

        * Description: Allows the user to zoom in and out of the image. As well, the user can press shift and move the image around.

    Also, if an object map is one of the inputs to the widget then:

- Object Map

    - Description: Displays the individual object information.

NIREP largely consists of four main modules: Interface, Display Manager, Evaluator and Data Manager. Of the four, three of the modules,Display Manger, Evaluator and DataManger, are controlled using two different configuration files: NIREP Display Description Document (ND3) and Resource Description List (RDL).

The ND3 file is responsible for controlling the behavior of the three components of NIREP, and the human-readable file typically looks like the following:

```
Begin ResourceDescriptionList

  Databases\NA0.2\resources.rdl

  Transformations\NA0.2\AIR5_Param1\air5.rdl

  Transformations\NA0.2\SICLE_Param1\sicle.rdl

  Transformations\NA0.2\BRAINSDemons_Param1\demons.rdl

End ResourceDescriptionList


Begin DisplayAttributes

columnSize(2)

rowSize(2)

LockAll()

End DisplayAttributes


Begin VariableList

image1 = na0|008

image2 = na0|014

modality = MRI

End VariableList


Begin WidgetList
```

```
W1,1 = view(_image1,Image ${image1})

W1,2 = checkerboard(_image1,_image2,Images ${image1} and ${image2})

W2,1 = view(_image2,Image ${image2})

W2,2 = wipe(_image1,_image2,Images ${image1} and ${image2})

End WidgetList



Begin EvaluatorList

_image1 = SpatialData(image1,modality)

_image2 = SpatialData(image2,modality)

_A3 = Transformation(image1,image2,SICLE_Param1)

_A20 = Transformation(image2,image1,SICLE_Param1)

_A7 = SpatialData(image1,OBJMAP)

_A8 = SpatialData(image2,OBJMAP)

_A9 = Jacobian(_A3)

_A10 = TransformImage(_image1,_A3)

_A27 = InverseConsistencyErrorImage(_A20,_A3,inv)

_A29 = InverseConsistencyErrorImage(_A3,_A20,comp)

_A30 = InverseConsistencyErrorImage(_A20,_A3,comp)

End EvaluatorList
```

The ND3 file is divided up into five different segments:

1. Resource Description List

2. Display Attributes

3. Variable List

4. Widget List

5. Evaluator List

A.8   Resource Description List

This section is demarcated by Begin ResourceDescriptionList and End ResourceDescriptionList operatives. Each line in this section lists relative paths to Resource Description List files which provide information about how data will be read into NIREP.

A.9   Display Attributes

This section is denoted by Begin DisplayAttributes and End DisplayAttributes. There are seven Display Attribute commands that can be specified:

1. RowSize(int r) Row size of the NIREP Display Widget grid. e.g. RowSize(3)

2. ColumnSize(int c) Column size of the NIREP Display Widget grid. e.g. ColumnSize(3)

3. Lock(Widget w1; Widget w2; ; Widget wN) Locks behaviors of specified widgets. e.g. Lock(W1,1;W1,2;W1,3)

4. CursorLocation(Widget w; int x; int y) Default cursor location of specified widget. e.g. CursorLocation(W1,1;50;128)

5. Slice(Widget w; int z) Default image slice location of specified widget. e.g. Slice(W2,2;150)

6. Orientation(Widget w; Orientation o) Default orientation of specified widget. e.g. Orientation(W3,1;TRANSVERSE)

7. LockAll() Locks all Display widgets.

RowSize() and ColumnSize() are required while the remaining commands are optional.

A.10   Variable List

This section is specified between Begin VariableList and End VariableList operatives. The variables that will be shared between the Widget List and the Evaluator List will be specified on each line, following a variable_name = variable format. The variables can be changed dynamically during runtime in NIREP, making mass switch of variables simple (e.g. simultaneously changing the display of multiple widgets with a single variable change).

A.11   Widget List

This section is defined between Begin WidgetList and End WidgetList. Each line of the Widget List specifies the behavior of each widget, in the following format: widget_name = widget_function(vars). Widget names take the form Wr,c, where r and c are row and column indices (e.g. W1,1 for first row and column). The following is the list of NIREP Display Widget functions:

1. View(image, title) 2D image viewer that displays the image specified in the variable image, with widget title title. The image data can be viewed from three different image orientations: Sagittal, Coronal and Transverse.

   image - Input image variable. The variable should either be defined in the Evaluator List or the Variable List, and should point to Image or ObjectMap data.

   title - Title of the widget. Variables from the Variable List can be used using the $var operative.

   e.g. View(_A1, Image 001 MRI) View(img, Image $image $modality)

2. Checkerboard(image1, image2, title) Displays two images in a checkerboard grid for comparison. All three image orientations supported by the View() widget are supported.

   image1 - First input image variable. The variable should either be defined in the Evaluator List or the Variable List, and should point to Image or ObjectMap data.

   image2 - Second input image variable. The variable should either be defined in the Evaluator List or the Variable List, and should point to Image or ObjectMap data.

   title - Title of the widget. Variables from the Variable List can be used using the $var operative.

   e.g. Checkerboard(_A1, _A2, Checkerboard Image 001 CT & Image 002 CT)

3. Wipe(image1, image2, title) Displays two images in a wipe widget for comparison. All three image orientations supported by the View() widget are supported.

   image1 - First input image variable. The variable should either be defined in the Evaluator List or the Variable List, and should point to Image or ObjectMap data.

   image2 - Second input image variable. The variable should either be defined in the Evaluator List or the Variable List, and should point to Image or ObjectMap data.

   title - Title of the widget. Variables from the Variable List can be used using the $var operative.

   e.g. Wipe(_A1, img, Wipe Image 001 CT & Image $image$modality)

4. ObjectMap(objmap, title) 2D object map viewer with colorized object regions. All three image orientations supported by the View() widget are supported.

   objmap - Input object map variable. The variable should either be defined in the Evaluator List or the Variable List, and should point to ObjectMap data.

   title - Title of the widget. Variables from the Variable List can be used using the $var operative.

   e.g. ObjectMap(_A3, ObjectMap 001)

5. OverlayObjectMap(image, objmap, title) 2D image viewer with colorized object map overlay. All three image orientations supported by the View() widget are supported.

image - Input image variable. The variable should either be defined in the Evaluator List or the Variable List, and should point to Image data.

objmap - Input object map variable. The variable should either be defined in the Evaluator List or the Variable List, and should point to ObjectMap data.

title - Title of the widget. Variables from the Variable List can be used using the $var operative.

e.g. OverlayObjectMap(_A1, _A3, ObjectMap Overlay 001 CT)

6. OverlayImage(image1, image2, title) 2D image viewer with second image overlay. All three image orientations supported by the View() widget are supported.

   image1 - First input image variable. The variable should either be defined in the Evaluator List or the Variable List, and should point to Image data.

   image2 - Second input image variable. The variable should either be defined in the Evaluator List or the Variable List, and should point to Image data.

   title - Title of the widget. Variables from the Variable List can be used using the $var operative.

   e.g. OverlayImage(_A1, _A2, Image Overlay 001 CT 002 CT)

7. RelativeOverlap(objmap1, objmap2, title) 2D viewer showing the relative overlap difference between two object maps. All three image orientations supported by the View() widget are supported.

   objmap1 - First input object map variable. The variable should either be defined

in the Evaluator List or the Variable List, and should point to ObjectMap data.

objmap2 - Second input object map variable. The variable should either be defined in the Evaluator List or the Variable List, and should point to ObjectMap data.

title - Title of the widget. Variables from the Variable List can be used using the $var operative.

e.g. RelativeOverlap(_A3, _A4, Relative Overlap 001 002)

8. RelativeOverlapTextWidget(objmap1, objmap2, title) Text widget showing the relative overlap difference between two object maps per ROI.

   objmap1 - First input object map variable. The variable should either be defined in the Evaluator List or the Variable List, and should point to ObjectMap data.

   objmap2 - Second input object map variable. The variable should either be defined in the Evaluator List or the Variable List, and should point to ObjectMap data.

   title - Title of the widget. Variables from the Variable List can be used using the $var operative.

   e.g. RelativeOverlapTextWidget(_A3, _A4, Relative Overlap 001 002)

9. InverseConsistencyTextWidget(NEED TO REVISE INPUT ARGUMENTS, objmap, title) Text widget showing the maximum, minimum and average inverse consistency error of forward and reverse transformations per ROI.

NEED TO REVISE INPUT ARGUMENTS

objmap - Input object map variable. The variable should either be defined in the Evaluator List or the Variable List, and should point to ObjectMap data.

title - Title of the widget. Variables from the Variable List can be used using the $var operative.

e.g. InverseConsistencyTextWidget(NEED TO REVISE INPUT ARGUMENTS, ICE 001 002)

10. TransitivityErrorTextWidget(NEED TO REVISE INPUT ARGUMENTS, objmap, title) Text widget showing the maximum, minimum and average transitivity error of multiple transformations per ROI.

    NEED TO REVISE INPUT ARGUMENTS

    objmap - Input object map variable. The variable should either be defined in the Evaluator List or the Variable List, and should point to ObjectMap data.

    title - Title of the widget. Variables from the Variable List can be used using the $var operative.

    e.g. TransitivityErrorTextWidget(NEED TO REVISE INPUT ARGUMENTS, TE 001 002 003)

11. VarianceTransformText(refimage, images, label, algorithm, title) Text widget showing the maximum, minimum and average intensity variance of images transformed to the reference coordinate system per ROI.

refimage - Input reference image coordinate system variable. All other images will be transformed to this reference coordinate system for IV computation. The variable should either be defined in the Variable List or explicitly specified.

images - Input image coordinate systems variable. Multiple images are specified in this variable to be transformed to the reference coordinate system. This variable is delineated by [and] braces, with image coordinate systems specified inside with combinations of the following: 1) individual coordinate systems tokenized by semi-colons (e.g. [001;002;004;005;006]); 2) ranges of images specified by colons (e.g. [001:002;004:006]. Note this specifies the same set of image coordinate systems as the previous example). The variable should either be defined in the Variable List or explicitly specified.

label - Image label variable. This variable specifies which images for the given image coordinate systems should be used for IV computation (e.g. MRI). The variable should either be defined in the Variable List or explicitly specified.

algorithm - Algorithm variable. This variable specifies which image registration algorithm is to be used to transform the images to the reference image coordinate system. The variable should either be defined in the Variable List or explicitly specified.

title - Title of the widget. Variables from the Variable List can be used using the $var operative.

e.g. VarianceTransformText (001,[002;004;006:016],MRI,SICLE,Intensity Vari-

ance SICLE)

12. VarianceImageText(images, label, title) Text widget showing the maximum, minimum and average intensity variance of specified images per ROI.

    images - Input image coordinate systems variable. Multiple images are specified in this variable and all comparisons are done with respect to the first image coordinate system. This variable is delineated by [and] braces, with image coordinate systems specified inside with combinations of the following: 1) individual coordinate systems tokenized by semi-colons (e.g. [001;002;004;005;006]); 2) ranges of images specified by colons (e.g. [001:002;004:006]. Note this specifies the same set of image coordinate systems as the previous example). The variable should either be defined in the Variable List or explicitly specified.

    label - Image label variable. This variable specifies which images for the given image coordinate systems should be used for IV computation (e.g. MRI). The variable should either be defined in the Variable List or explicitly specified.

    title - Title of the widget. Variables from the Variable List can be used using the $var operative.

    e.g. VarianceImageText ([001;002;004;006:016],MRI,Intensity Variance)

13. BlankWidget() What it says.

A.12   Evaluator List

This section is defined between Begin EvaluatorList and End EvaluatorList. Each line of the Evaluator List specifies evaluator functions, some of which may not be in use by any of the widgets, in the following format: evaluator_var = evaluator_function(vars). The following is the list of NIREP Evaluator functions:

(a) SpatialData(cs_id, label)

(b) Transformation(cs1_id, cs2_id, algorithm)

(c) Diff(image1, image2)

(d) XDisp(transform)

(e) YDisp(transform)

(f) ZDisp(transform)

(g) Jacobian(transform)

(h) TransformImage(image, transform, interpolation) [interpolation currently not implemented]

(i) InvertTransformation(transform)

(j) InverseConsistencyErrorImage(fwd_transform, rev_transform, method)

(k) TransitivityErrorImage(transform12, transform23, transform31, method)

(l) IntensityVarianceTramsform(refimage, images, label, algorithm)

(m) IntensityVarianceImage(images, label)

A.13   Data

A.14   Transformation

A transformation between two coordinate systems is a function that deforms and maps each point in one coordinate system to another coordinate system grid. That means that a transformation is derived from at least two different coordinate systems and an algorithm. NIREP is able to deform coordinate systems but in order to do that, NIREP needs to load in transformation files (either displacement field image(s) or specialized transformation files) that were generated by image registration algorithm(s). There are two different ways to load in transformation files, one is to load the transformation files by hand each time NIREP is run; the other way is to create an XML file that holds a list of transformation file locations and load that file each time NIREP is run. The second option was chosen partially because databases of transformations will be handed out and have people test out NIREP and partially the time spent creating a file and using it over and over is less than always loading in the exact transformation each time. A benefit that was found from using XML files to tabulate transformation data is that the user is able to place in file notes and other information about the transformation(s). Another benefit is that the user is able to group transformations in whatever manner the user wants to.

The implementation that NIREP uses is composed of the following XML lines.

```
<transform_description dmid = "Transformation(na0|001,na0|002,
```

```
SICLE_NO_ICC)">

    <coordinate_system index = "0" ns = "na0" id="001"/>

    <coordinate_system index = "1" ns = "na0" id="002"/>

    <algorithm>SICLE_NO_ICC</algorithm>

    <filename>na01_na02.coeffs</filename>

    <datatype>TRANSFORMATION</datatype>

    <format> SICLE_COEFF </format>

    <transformation_units>IMAGE_SPACE</transformation_units>

</transform_description>
```

Each transformation data entry starts with the tag `transform_description` along with the attribute dmid. The dmid is a unique ID that identifies the transformation data within NIREP.

Inside the tag `<transform_description>`, in no particular order, are the data description tags: `coordinate_system`, `algorithm`, `filename`, `datatype`, `format` and `transformation_units`.

The `<coordinate_system>` tag describes the coordinate systems that are associated with the transformation data. Each coordinate system will have an "index" attribute that lets NIREP know the sequence of coordinate systems used to create the transformation. The starting coordinate system has index 0 and every subsequent index is incremented by one. Each coordinate system belongs to a data namespace which is described by the "ns" (namespace) attribute and the "id" attribute identifies one of the coordinate systems in the namespace.

The `<algorithm>` tag labels the algorithm that was used to generate the transformation.

The `<filename>` tag is a relative path location to the data on disk. An optional attribute is index, which defines the order of the filenames.

The `<datatype>` tag is a NIREP defined value. For all transformations use Transformation.

The `<format>` tag is a NIREP defined value that defines the type of algorithm that NIREP will use to deform the transformation. The values are: `DISPLACEMENT`, `DISPLACEMENT3`, `SICLE_COEFF`, and `AIR5`. If displacement3 is used, there should be 3 filename tags.

The `<transformation_units>` is a NIREP defined value that defines what units the transformation file is stored as. The values are: `UNIT_CUBE`, `PHYSICAL_SPACE` and `IMAGE_SPACE`.

Moving on to the format of the algorithm descriptions, which is the notes that users will place in the file. The algorithm description is composed of the following lines

```
<algorithm_description id="AIR5">

    <description>AIR5 algorithm</description>
</algorithm_description>
```

The descriptions are quite simple and consist of the tag `algorithm_description` with an attribute id that corresponds to the algorithm tag in the `transform_description`

tag. There can be many `algorithm_descriptions` depending on how many algorithms the user places in the RDL.

NIREP supports the following list, mostly supported by ITK (http://www.vtk.org/Wiki/ITK_File_Formats).

A.14.1   3D image file formats:

Analyze 7.5TM, DICOM, GDCM, GE, Gipl, IPL, MetaImage, NIfTI, Nrrd, Raw, Siemens Vision, Stimulate, VTK Structured Points

Analyze Object Map

A.14.2   Transformation file formats:

MetaImage SICLE coefficients (.coeff) AIR 5 warp (.warp)

A.14.3   Requirements

- Images, supplementary data and transformations should have same orientation, origin and voxel spacing.

- Transformations can be defined in physical space, image space or unit cube space.

REFERENCES

[1] G. E. Christensen. Consistent linear-elastic transformations for image matching. In A. Kuba and M. Samal, editors, *Information Processing in Medical Imaging*, LCNS 1613, pages 224–237, Berlin, June 1999. Springer-Verlag.

[2] G. E. Christensen and H. J. Johnson. Consistent image registration. *IEEE Trans. Med. Imaging*, 20(7):568–582, July 2001.

[3] G. E. Christensen and H. J. Johnson. Invertibility and transitivity analysis for nonrigid image registration. *Journal of Electronic Imaging*, 12(1):106–117, Jan. 2003.

[4] G. E. Christensen, R. D. Rabbitt, M. I. Miller, S.C. Joshi, U. Grenander, T.A. Coogan, and D.C. Van Essen. Topological properties of smooth anatomic maps. In Y. Bizais, C. Braillot, and R. Di Paola, editors, *Information Processing in Medical Imaging*, volume 3, pages 101–112. Kluwer Academic Publishers, Boston, June 1995.

[5] William R. Crum, Oscar Camara, and Derek L. G. Hill. Generalized overlap measures for evaluation and validation in medical image analysis. *IEEE Trans. Med. Imaging*, 25(11):1451–1461, November 2006.

[6] W.R. Crum, O. Camara, D. Rueckert, K.K Bhatia, M. Jenkinson, and D.L.G. Hill. Generalized overlap measures for assessment of pairwise and groupwise image registration and segmentation. In *MICCAI 2005*, pages 99–106. Springer, 2005.

[7] Xiujuan Geng, Dinesh Kumar, and Gary E. Christensen. Transitive inverse-consistent manifold registration. In Gary E. Christensen and Milan Sonka, editors, *Information Processing in Medical Imaging*, volume LNCS 3564, pages 468–479, Berlin, July 2005. Springer-Verlag.

[8] Guido Gerig, Matthieu Jomier, and Miranda Chakos. Valmet: A new validation tool for assessing and improving 3d object segmentation. In Wiro J. Niessen and Max A. Viergever, editors, *MICCAI 2001*, volume LNCS 2208, pages 516–528. Springer, 2001.

[9] Tristan Glatard, Xavier Pennec, and Johan Montagnat. Performance evaluation of grid-enabled registration algorithms using bronze-standards. In *MICCAI 2006*, pages 152–160. Springer, 2006.

[10] H. J. Johnson and G. E. Christensen. Consistent landmark and intensity-based image registration. *IEEE Trans. Med. Imaging*, 21(5):450–461, 2002.

[11] Arno Klein, Jesper Andersson, Babak A. Ardekani, John Ashburner, Brian Avants, Ming-Chang Chiang, Gary E. Christensen, D. Louis Collins, James Gee, Pierre Hellier, Joo Hyun Song, Mark Jenkinson, Claude Lepage, Daniel Rueckert, Paul Thompson, Tom Vercauteren, Roger P. Woods, J. John Mann, and Ramin V. Parsey. Evaluation of 14 nonlinear deformation algorithms applied to human brain MRI registration. *NeuroImage*, 46(3):786–802, July 2009.

[12] Joo Song, Gary Christensen, Jeffrey Hawley, Ying Wei, and Jon Kuhl. Evaluating image registration using nirep. In Bernd Fischer, Benot Dawant, and Cristian Lorenz, editors, *Biomedical Image Registration*, volume 6204 of *Lecture Notes in Computer Science*, pages 140–150. Springer Berlin / Heidelberg, 2010.

[13] Ying Wei, Gary E. Christensen, Joo Hyun Song, David Rudrauf, Joel Bruss, Jon G. Kuhl, and Thomas J. Grabowski. Evaluation of five non-rigid image registration algorithms using the nirep framework. volume 7623, page 76232L. SPIE, 2010.

[14] Jay West, J. Michael Fitzpatrick, et al. Comparison and evaluation of retrospective intermodality brain image registration techniques. *J. Comp. Asst. Tomog.*, 21(4):554–566, 1997.

[15] Alex P. Zijdenbos, Benoit M. Dawant, Richard A. Margolin, and Andrew C. Palmer. Morphometric analysis of white matter lesions in mr images: Method and validation. *IEEE Transactions on Medical Imaging*, 13(4):716–724, December 1994.