Intraoperative Registration of Pathology for Adjuvant Postoperative Radiotherapy

Computer Integrated Surgery II
Spring 2014
Project Report
Team 4

Team Members: Kareem Fakhoury, Matthew Hauser, Steven Lin
Mentors: Dr. Harry Quon, Dr. Jeremy Richmon, Dr. Junghoon Lee

Introduction

Our project focused on producing a workflow for evaluating tissue deformation. This workflow includes the methodology of actions to be taken pre-, intra-, and postoperatively, as well as a method for postoperative image processing. The latter was accomplished in two ways.

First, it was completed as originally planned in the initial project plan presentation. This included registration of intraoperative tracking data to outline the pathology to the preoperative CT image and then registration of the preoperative CT to the postoperative CTs to evaluate the movement of the points taken intraoperatively.

Second, we used a method of registration, detailed more fully in the approach section, which included the registration of intraoperative data to the postoperative open-wound CT image, followed by two image registrations: one from the postoperative closed-wound image to the preoperative image and the next from the resultant image to the postoperative open-wound image. Although this method does not represent a realistic scenario for clinical use because in practice open-wound CT scans are rarely taken, it represents the ideal process.

It was found that the original method was found to contain fairly accurate results. Determination of the locations of the points of pathology was less than 1 cm from ground truth. The second method did not yield as accurate results, but did provide insight into small changes in the workflow could improve the quality result. We have begun to address the problem of providing better guidelines on where to deliver postoperative radiotherapy.

Background and Problem

With regards to cancers in the head and neck, the general procedure for eliminating the pathology is to perform surgical resection of the tumor as well as pre- and post-operative radiotherapy. Adjuvant radiotherapy is delivered based on a plan created after the surgery and on pre-operative and post-operative CT scans as well as on reports from the operating surgeon. However, post-operative tissue deformation – shifts in the anatomy surrounding the surgical area – makes the previous location of the tumor difficult to identify. Because of this uncertainty and in order to ensure that none of the remaining cancer cells are missed, the area identified for radiotherapy is overestimated. This is harmful to patients because the volume of irradiated tissue dictates the toxicity affecting the patient, which has negative downstream consequences, such as intense pain and the inability to swallow and eat autonomously. The goal of our project is to show how the tissue around the surgical area deforms from pre- to post-operative CT scans. This will allow radiation oncologists and dosimetrists to more accurately localize the area containing the remaining cancer cells. This information, in turn, will inform planning to allow tighter and more accurate volumes for adjuvant radiotherapy. Small decreases in irradiated volume will lead to significant decreases in toxicity.

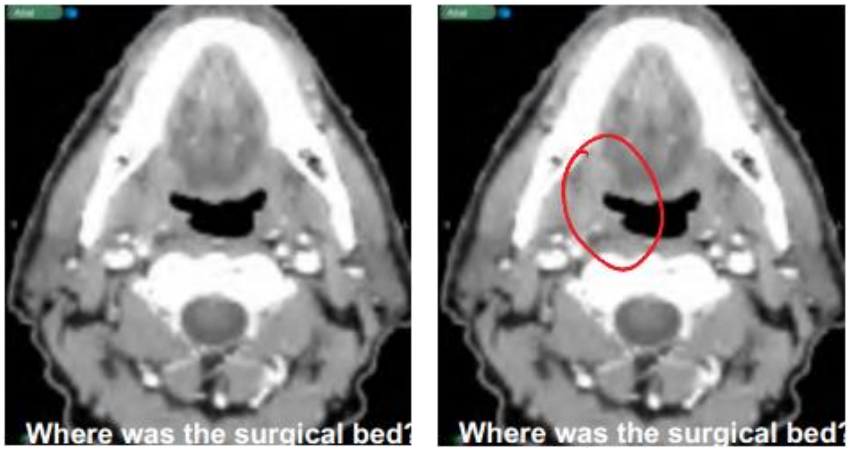

Figure 1. The above images demonstrate the difficulty with which radiation oncologists are faced in determining where the tumor was before resection and, therefore, where to deliver radiotherapy

Approach

Our approach involves actions taken before, during, and after tumor resection, as well as post-processing of image data. As mentioned in the introduction, the post-processing of the data was completed with two distinct methodologies.

The experimental setup required three CT scans of the subject: a preoperative scan, a postoperative scan taken before the wound was closed (i.e. open-wound), and a postoperative scan taken after the wound was closed (i.e. closed-wound). Intraoperatively, the pathology was outlined by recording the position of various points around the area of the removed tissue using the Polaris optical tracking system. In post-processing, the intraoperative Polaris data outlining the pathology was registered to the preoperative CT scan for use with the primary method and to the postoperative-open CT for the secondary method. Using an open-source medical imaging software package, Elastix, the preoperative CT scan was then registered to each of the postoperative scans. Transformix was used to determine the movement of the points outlining the pathology.

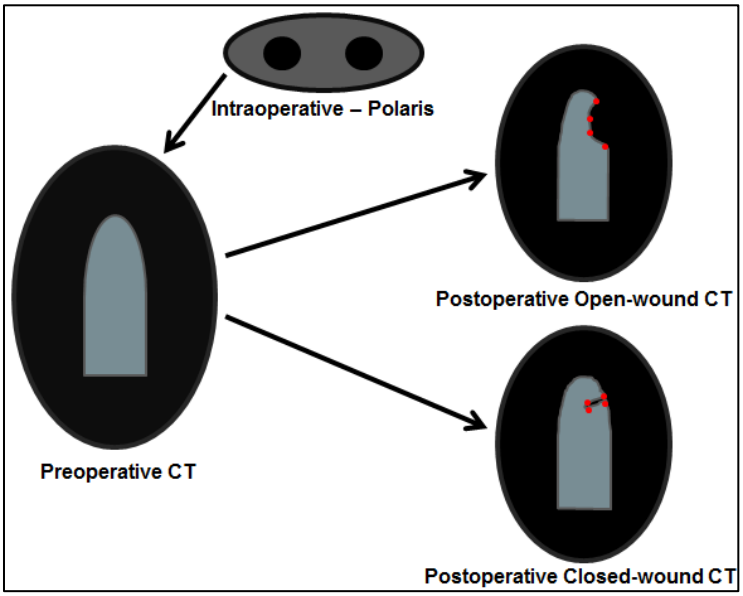

Figure 2. This diagram summarizes the directions of registration: intraoperative to preoperative, and preoperative to each of the postoperative

To test our methodology, the experiment was carried out on three pig heads, which have a size and shape similar to human heads. For each pig head, we placed five fiducials on the surface of the head (see Figure 3) and took preoperative CT scans. We then simulated surgical resection by removing a semicircular portion of the tongue. Four radio-opaque markers were placed at the positions at which we would record points with the Polaris to outline the pathology (see Figure 4); the markers served as the ground truth. In the same frame, we recorded the positions of the markers on the tongue using the Polaris, as well as the positions of the five fiducials on the head to allow a point cloud to point cloud transformation in post-processing, which allowed us to then register the Polaris points outlining the pathology to the preoperative CT image.

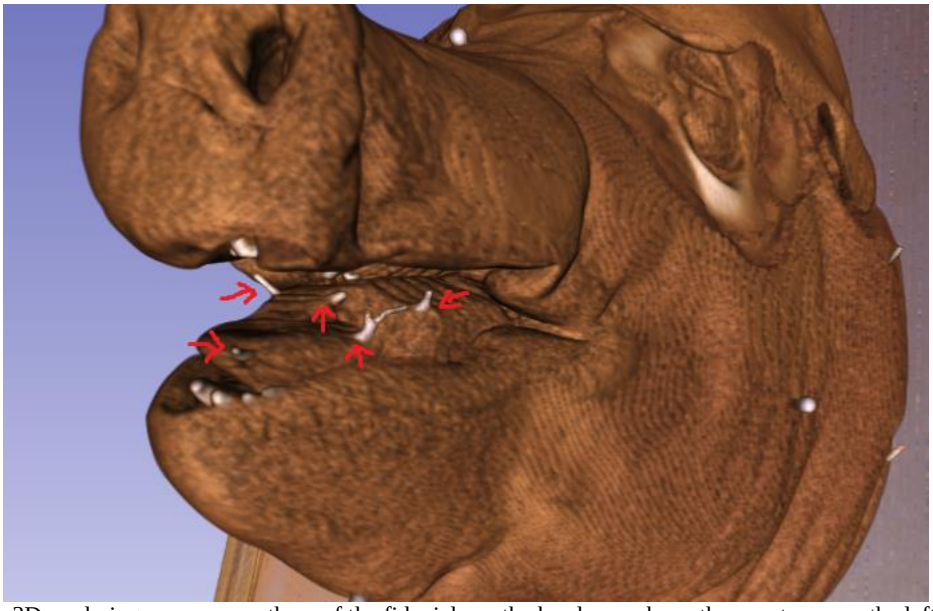

Figure 3. In this 3D rendering, one can see three of the fiducials on the head: one above the snout, one on the left cheek and one under the chin. There are also ones on the right cheek and on the forehead

Figure 4. The above image shows the tissue resection from the tongue as well as the placement of the four gold standard markers on the tongue, placed anterior, posterior, medial, and medial deep

In post-processing, we digitally removed the ground truth clips (see Figure 5) and registered the intraoperative Polaris tongue data to the preoperative image. Then, using Elastix open source software, we determined the parameter set (see Appendix) that would allow for the best registration between the preoperative CT image and the postoperative CT images. We would determine the accuracy of the rigid registration by computing root mean square (RMS) error between measured coordinates of the fiducials and computed coordinates of the fiducials. For the deformable registration, we would compute the RMS error between the ground truth markers coordinates and the computed tongue coordinates after having moved during the registration.

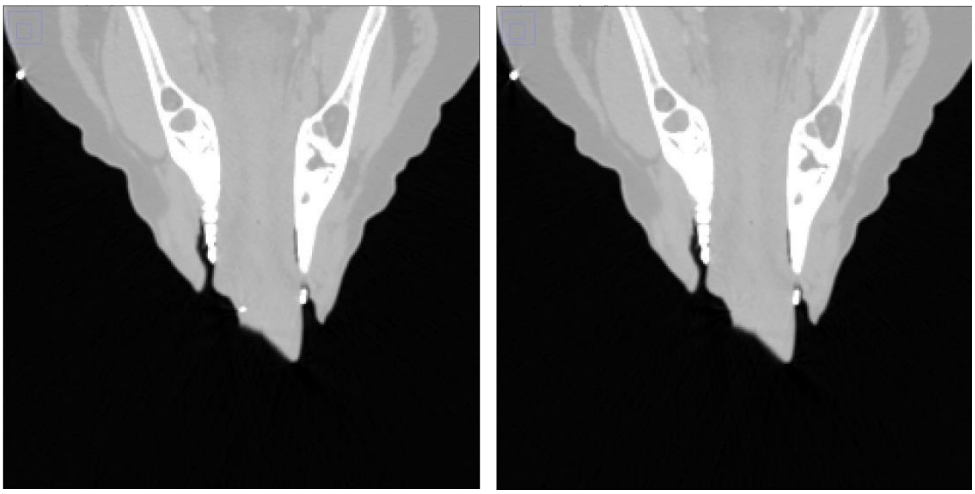

Figure 5. This figure demonstrates the digital removal of a clip. The image on the left shows the image before clip removal and the image on the right shows after clip removal

The points determined using the Polaris system that were registered to the preoperative image were then transformed to their corresponding points on the postoperative image using the transformation determined from the image registration. The accuracy of their movement was determined by means of a comparison with the actual locations of the ground truth surgical clips.

Since the points around the pathology were recorded after surgical resection, it is expected that one would achieve a more accurate image registration by first registering the intraoperative Polaris data to the postoperative open-wound image and then subsequently registering that image to the preoperative image followed by a registration to the postoperative closed-wound image. This represented our second method for determining pathology.

See the results section and the following discussion for further information on this alternative approach.

Results

The results from both of our methods indicate that it is possible to intraoperatively monitor the location of pathology. Additionally, both of the two methods indicate both possibilities and constraints on the method.

The first method which involved directly registering the postoperative images to the preoperative images provided, on average, fairly accurate results. The results from each of the three trials are indicated in the tables below with various parameter sets being used for the registration. The default parameter file involves using advance mean squares as its metric, and a B-Spline transform as its transformation and manual defined fiducial points to guide the rigid transformation. The parameter lists the element(s) that are different from this default parameter list. Only deformable transformations following a standard rigid transformation are shown.

Table 1

Pig 1- Error in mm					
Parameters	Clip 1	Clip 2	Clip 3	Clip 4	RMS
AdvancedMeanSqaures	32.88	29.64	29.55	28.24	30.13
SplineKernelTransform	6.69	6.99	8.43	9.89	8.10

Table 2

Pig 2- Error in mm					
Parameters	Clip 1	Clip 2	Clip 3	Clip 4	RMS
AdvancedMeanSqaures	12.29	10.85	5.69	6.20	9.21
SplineKernelTransform	2.97	5.05	4.22	4.51	4.25

Table 3

Pig 3- Error in mm					
Parameters	Clip 1	Clip 2	Clip 3	Clip 4	RMS
AdvancedKappaStatistic	9.64	7.99	8.00	6.59	8.13
AdvancedMattesMutualInformation	9.64	8.18	8.21	6.86	8.28
AdvancedMeanSqaures	13.80	12.23	4.27	6.59	10.02
AdvancedMeanSquares-WithoutFixedPoints	8.91	11.33	6.98	4.51	8.32
KappaStatisticSplineKernelTransform	23.16	23.25	21.20	27.84	23.99
SplineKernelTransform	8.14	7.64	7.93	5.94	7.46

From the optimization it became clear that the best parameters for the transformation involved using thin plate splines for the deformable registration.

Table 4

Spline Kernel Transform					
	Clip 1	Clip 2	Clip 3	Clip 4	RMS
Pig 1	6.69	6.99	8.43	9.89	8.10
Pig 2	2.97	5.05	4.22	4.51	4.25
Pig 3	8.14	7.64	7.93	5.94	7.46

The results for the experiment prove to be fairly accurate for all three trials. The average distance an optical tracker marker ended up falling from its ground truth location was 6.53 mm which is less than the margins used in many current radiation treatment plans. Let's now turn our attention to the second method.

The second method involved first determining the location of the Polaris points on the postoperative open CT scan. The closed CT scan was then registered to the preoperative scan and subsequently registered to the postoperative open scan. This methodology allowed the Polaris points to have a more accurate initial placement because the open CT scan more closely resembles that of the intraoperative situation.

Table 5

STD, AMS					
	Clip 1	Clip 2	Clip 3	Clip 4	RMS
Pig 1	8.651725	10.22334	12.97421	12.2157	11.14588
Pig 2	33.49595	29.86056	10.97079	17.66517	24.72877
Pig 3	24.68685	16.94226	6.201226	10.91687	16.23356

Table 6

STD, AMS2					
	Clip 1	Clip 2	Clip 3	Clip 4	RMS
Pig 1	9.05125	11.94012	15.53985	12.2157	12.40163
Pig 2	35.41654	29.60225	11.75918	17.64705	25.39843
Pig 3	20.99748	10.49664	7.115601	7.24926	12.78923

Table 7

STD, AMS, Fixed Points					
	Clip 1	Clip 2	Clip 3	Clip 4	RMS
Pig 1	25.56007	13.73505	4.578349	9.156698	15.38485
Pig 2	19.16119	14.07881	10.89262	9.06512	13.84006
Pig 3	29.94147	20.39446	7.379377	11.9283	19.42399

Results from this, are not as accurate as those received using the first method, but this fact is both reconcilable and informative. Since the Polaris points more accurately initially fall onto the open CT scan, it would be expected that the results would be more consistent because this allows for consistent placement of the points before image registration. With that being said, it would be expected that because this method requires multiple registrations, it is more likely to lead to a less accurate overall registration due to compounding of error.

Discussion of Error

In addition, throughout the experiment, it became apparent that the preoperative tongue points from the Polaris were not in the exact proper location. In fact, in some instances the points were above the tongue, floating in the air. These points were not successfully transformed by the deformable registration. It is imperative that this be remedied in future development.

One possible cause is that the points are taken from the curved tip of the surgical clips to allow for consistency and reproducibility. However, the tips protruded about 5 mm from the tongue and, thus, the Polaris data did not represent points on the actual surface of the tongue. This is an issue in the methodology, and should be corrected for future experiments. In terms of surgical workflow, this should not be an issue as the surgeon would identify the points directly on the tissue. However, in order to have more successful data, simply taking the points at the base of the clip on the tongue should aid the situation.

Another source of error is the inherent difference in the location of the tongue during the intraoperative stage when the Polaris data on the tongue was collected, and the location of the tongue preoperatively, which is when the image to which the Polaris points are registered is taken. In the current methodology, the tongue is simply approximately inserted back into its original location. Efforts were made to view the extent of this issue by registering the preoperative image to the postoperative open wound image so as to identify the correct points on the tongue. Ultimately, due to the extra image registration that must occur, the method proved to be far less successful. This is likely due to the compounding of error over multiple steps.

Significance

The results from this experiment could potentially have large scale clinical impact. Although our method is still far from complete and is in need of more development, the study serves as a strong indicator of proof of feasibility. If successful, the method could be adapted to other regions of the body and aid in radiation treatment planning for patients with a wide array of cancers.

Management Summary

Project Roles:

Matthew Hauser: Matt was responsible for image processing prior to registration. He also led the effort in registration of the preoperative image to the postoperative closed image and analysis of the resulting transformations.

Kareem Fakhoury: Kareem led the effort in coordinating data collection between the mentors, the technicians, the Polaris users, and the butcher shop. He played a role in registering the preoperative image to postoperative closed image.

Steven Lin: Steven led in registering the intraoperative Polaris points to the preoperative CT image. He played a role in registering the preoperative image to postoperative closed image.

Expected versus Accomplished Deliverables:

A key deliverable is the collection of data, which was accomplished. We compared the use of the ANTs software package to the Elastix software and decided to focus on Elastix because it provided better initial results and appeared more versatile. We were able to compare various metrics and parameters using the Elastix software and successfully created fairly optimized parameter files. Assessment of the feasibility of the methodology was accomplished. The deliverable of submitting a paper was not pursued, but may be pursued in the near future.

Future Development:

From the cumulative results of the trial, it seems that providing fixed points for the registration is beneficial. A next step could be to use other rigid structures that are closer to the tongue in order to aid the rigid registration. Additionally, Dr. Quon has submitted a grant to fund further research on the matter including funding for experiments on 60 pig heads. The experiments will be done to further evaluate the approach as well as to determine the feasibility of other similar scenarios.

What was Learned:

The registration between CT images is a complex field. Based on the background papers, it appears that current CT to CT registration cannot account for drastic alterations such as a surgical removal of tissue. Some papers suggested generating a hole in the tissue; however this would not coincide with the purpose of the experiment in this paper. The team was able to explore several different metrics of comparison as well as different optimizers, which were very interesting as a high level view of current registration techniques.

Appendices

Appendix A: Rigid Transformation File- Performed before every registration

```
(FixedInternalImagePixelType "short")
(MovingInternalImagePixelType "short")
(FixedImageDimension 3)
(MovingImageDimension 3)
(UseDirectionCosines "true")

// *************** Main Components **************************

(Registration "MultiMetricMultiResolutionRegistration")
(Interpolator "BSplineInterpolator")
(ResampleInterpolator "FinalBSplineInterpolator")
(Resampler "DefaultResampler")
(FixedImagePyramid "FixedSmoothingImagePyramid")
(MovingImagePyramid "MovingSmoothingImagePyramid")

(Optimizer "StandardGradientDescent")
(Transform "EulerTransform")
(Metric "AdvancedMeanSquares")

// ***************** Transformation **************************

(HowToCombineTransforms "Compose")

// ****************** Similarity measure *********************

(NumberOfHistogramBins 64)
(ErodeMask "false")

// ******************* Multiresolution ***********************

(NumberOfResolutions 3)
(ImagePyramidSchedule 4 4 4  2 2 2  1 1 1 )

// ****************** Optimizer ****************************

(MaximumNumberOfIterations 1000)
//(MaximumStepLength 1.0)
//(RequiredRatioOfValidSamples 0.05)

// *************** Image sampling **********************

(NumberOfSpatialSamples 2000)
(NewSamplesEveryIteration "true")
(ImageSampler "Random")

// ************* Interpolation and Resampling ***************

(BSplineInterpolationOrder 1)
(FinalBSplineInterpolationOrder 3)
```

```
(DefaultPixelValue -1024)

//SP: Param_a in each resolution level. a_k = a/(A+k+1)^alpha
(SP_a 0.2)

//SP: Param_A in each resolution level. a_k = a/(A+k+1)^alpha
(SP_A 50)

//SP: Param_alpha in each resolution level. a_k = a/(A+k+1)^alpha
(SP_alpha 0.6 )

(WriteResultImage "true")
(WriteResultImageAfterEachResolution "false")
(WriteTransformParametersEachIteration "false")
(WriteTransformParametersEachResolution "false")

// The pixel type and format of the resulting deformed moving image
(ResultImagePixelType "float")
(ResultImageFormat "nii")
```

Appendix B: Parameter Files for Registration of Preoperative to Postoperative Closed-Wound

```
// Parameter file for B-spline registration

// The internal pixel type, used for internal computations
(FixedInternalImagePixelType "short")
(MovingInternalImagePixelType "short")

// The dimensions of the fixed and moving image
(FixedImageDimension 3)
(MovingImageDimension 3)

// *************** Main Components **************************

(Registration "MultiMetricMultiResolutionRegistration")
(Interpolator "BSplineInterpolator")
(ResampleInterpolator "FinalBSplineInterpolator")
(Resampler "DefaultResampler")

(FixedImagePyramid "FixedRecursiveImagePyramid")
(MovingImagePyramid "MovingRecursiveImagePyramid")

(Optimizer "QuasiNewtonLBFGS")

// Manually definded landmark based points
(Metric "AdvancedMeanSquares" "CorrespondingPointsEuclideanDistanceMetric")
// ***************** Transformation **************************

// Thin Plate Splines
(Transform "SplineKernelTransform")

// Combination of Transforms
(HowToCombineTransforms "Compose")


// ******************** Multiresolution **********************

// The number of resolutions.
(NumberOfResolutions 4)
(ImagePyramidSchedule 32 32 32 16 16 16)

// The control point spacing of the bspline transformation in
// the finest resolution level.
// Unit: mm.
(FinalGridSpacingInPhysicalUnits 5.0)
(GridSpacingSchedule 4.0 4.0 2.0 1.0)

// ******************* Similarity measure *********************

// Number of grey level bins in each resolution level,
(NumberOfHistogramBins 32)

// Mask serves as region of interest, set it to false.
(ErodeMask "false")

// ****************** Optimizer ****************************
```

```
// Maximum number of iterations in each resolution level:
(MaximumNumberOfIterations 500)

// *************** Image sampling **********************

// Number of spatial samples used to compute the mutual
// information (and its derivative) in each iteration.
(NumberOfSpatialSamples 2000)

// Refresh these spatial samples in every iteration, and select
(NewSamplesEveryIteration "true")
(ImageSampler "Random")

// ************* Interpolation and Resampling ***************

// Order of B-Spline interpolation used during registration/optimisation.
// 1 gives linear interpolation.
(BSplineInterpolationOrder 1)

// Order of B-Spline interpolation used for applying the final deformation
(FinalBSplineInterpolationOrder 3)

//Default pixel value for pixels that come from outside the picture:
(DefaultPixelValue -1000)

//SP: Param_a in each resolution level. a_k = a/(A+k+1)^alpha
(SP_a 0.1)

//SP: Param_A in each resolution level. a_k = a/(A+k+1)^alpha
(SP_A 50.0 )

//SP: Param_alpha in each resolution level. a_k = a/(A+k+1)^alpha
(SP_alpha 0.6 )

// Output
(WriteResultImage "true")
(WriteResultImageAfterEachResolution "false")
(WriteTransformParametersEachIteration "false")
(WriteTransformParametersEachResolution "false")

// The pixel type and format of the resulting deformed moving image
(ResultImagePixelType "float")
(ResultImageFormat "nii")
```

Appendix C: Parameter File for Registration of Postoperative Open-Wound to Preoperative to Postoperative Closed-Wound

Parameter files for registering postoperative open-wound CT to preoperative CT then registering preoperative CT to postoperative closed-wound CT. Elastix provides the transformation parameters from the fixed image to the moving. Therefore the flow of work should be:

1. Register moving preoperative to fixed postoperative open-wound.
2. Transform points from postoperative open-wound to preoperative.
3. Register moving postoperative closed-wound to fixed preoperative.
4. Transform from preoperative to postoperative close-wound.

Deformable transformation parameter file:

```
// Example parameter file for B-spline registration
// C-style comments: //

(FixedInternalImagePixelType "short")
(MovingInternalImagePixelType "short")

(FixedImageDimension 3)
(MovingImageDimension 3)

// ***************** Main Components **************************

// The following components should usually be left as they are:
(Registration "MultiMetricMultiResolutionRegistration")
(Interpolator "BSplineInterpolator")
(ResampleInterpolator "FinalBSplineInterpolator")
(Resampler "DefaultResampler")

(FixedImagePyramid "FixedRecursiveImagePyramid")
(MovingImagePyramid "MovingRecursiveImagePyramid")

(Optimizer "StandardGradientDescent")
(Transform "BSplineTransform")
(Metric "AdvancedMeanSquares")

// ***************** Transformation **************************

// ******************* Multiresolution **********************

(NumberOfResolutions 4)
(ImagePyramidSchedule 8 8 8  4 4 4  2 2 2  1 1 1 )

(FinalGridSpacingInPhysicalUnits 5.0)
(GridSpacingSchedule 4.0 4.0 2.0 1.0)

(HowToCombineTransforms "Compose")
```

```
// ******************* Similarity measure *********************

(NumberOfHistogramBins 32)

(ErodeMask "false")

// ****************** Optimizer ****************************

(MaximumNumberOfIterations 2000 4000 8000 8000)

(MaximumStepLength 1.0)

// *************** Image sampling **********************

(NumberOfSpatialSamples 2000)

(NewSamplesEveryIteration "true")
(ImageSampler "Random")

// ************* Interpolation and Resampling ***************

(BSplineInterpolationOrder 1)

(FinalBSplineInterpolationOrder 3)

(DefaultPixelValue -1024)

//SP: Param_a in each resolution level. a_k = a/(A+k+1)^alpha
(SP_a 0.8)

//SP: Param_A in each resolution level. a_k = a/(A+k+1)^alpha
(SP_A 53)

//SP: Param_alpha in each resolution level. a_k = a/(A+k+1)^alpha
(SP_alpha 0.6 )

(WriteResultImage "true")
(WriteResultImageAfterEachResolution "false")
(WriteTransformParametersEachIteration "false")
(WriteTransformParametersEachResolution "false")

// The pixel type and format of the resulting deformed moving image
(ResultImagePixelType "float")
(ResultImageFormat "nii")
```

Appendix D: Code for Registration of Intraoperative Polaris Data to Preoperative Image

Cartesian.m

```
classdef Cartesian
    %   Cartesian Math Package. Contains functions and operations for
    %   points in the 3-D space
    properties
    end
    methods(Static)
        %reformat rotation and translation in a 4x4 matrix
        function mat = rearrange(R,p)
            [r,c] = size(R);
            %checks if the given rotation, R is a valid matrix  that is 3x3
            if r == 3 && c == 3
            else
                errMsg = ['Invalid rotation matrix, need a 3x3 matrix'...
                    'for a 3D space'];
                error(errMsg);
            end ;

            [isP, p] = Cartesian.isVec(p);

            %check if p is a valid translation
            if ~isP
                error('Invalid translation vector');
            end

            mat = [R; 0 0 0];
            p = [p;1];
            mat = [mat, p];
        end
        %reformat a matrix into rotation and translation
        function [R,P] = fDecompose(F)
            [r,c] = size(F);
            %checks if cloud or vector contain 4 points in either
            %directions
            if r == 4
                %if F is a vector
                if c == 1
                    R = F(1:3,1);
                    P = R;
                else
                    R = F(1:3,1:c-1);
                    P = F(1:3,c);
                end
            else
                error('Invalid vector of point cloud');
            end
        end
        %reformat a matrix into cloud or point
        function [R,P] = decompose(F)
            [r,c] = size(F);
            %checks if cloud or vector contain 4 points in either
            %directions
```

```
    if r == 4
        %if F is a vector
        if c == 1
            R = F(1:3,1);
            P = R;
        else
            R = F(1:3,1:c);
            P = [];
        end
    else
        error('Invalid vector of point cloud');
    end
end
%frame transformation given rotation/translation
function r = transform(vector,R,p)
    %if R is rotation, P is translation
    if nargin == 3
        R = Cartesian.rearrange(R,p);
        if R == 0
            error('Invalid transformation\n');
        end
    end
    %if R is a combined transformation
    if nargin == 2
        [rows,columns] = size(R);
        if rows ~= 4 || columns ~= 4
            error('Invalid transformation\n');
        end
    end

    %checks if input is a point cloud
    [isCloud, cloud] = Cartesian.isCloud(vector);
    %checks if input is a vector
    [isVec, vec] = Cartesian.isVec(vector);

    if isCloud
        width = size(cloud,2);
        r = R*[cloud; ones(1,width)];
        %decomponses, and returns a matrix without extra 1's
        r = Cartesian.decompose(r);
    elseif isVec
        r = R*[vec;1];
        [empty,r] = Cartesian.fDecompose(r);
    else
        error('Invalid input vector or cloud');
    end
end
%calculates rotation matrix based on desired angles of rotation
%relative to each axis (x,y,z)
function rMat = rotateMat(xang, yang, zang, type)
    if nargin == 4
        if strcmp(type,'Degree') || strcmp(type,'degree')
            xang = xang/180*pi;
            yang = yang/180*pi;
            zang = zang/180*pi;
        end
```

```
    end

    %rotation matrix based on each angle
    Rx = [1 0 0;
        0 cos(xang) -sin(xang);
        0 sin(xang) cos(xang)];

    Ry = [cos(yang) 0 sin(yang);
        0 1 0;
        -sin(yang) 0 cos(yang)];

    Rz = [cos(zang) -sin(zang) 0;
        sin(zang) cos(zang) 0;
        0 0 1];

    %returns combination of matrices
    rMat = Rx * Ry * Rz;
end
%rotates vector or group of vectors based on provided angles,
%rejects if p is not a vector
function r = rotate(p, xang, yang, zang, type)
    if nargin == 5
        Rtot = Cartesian.rotateMat(xang,yang,zang,type);
    else
        Rtot = Cartesian.rotateMat(xang,yang,zang);
    end

    Rtot = Cartesian.rearrange(Rtot, [0 0 0]);
    r = Cartesian.transform(p, Rtot);
end
%translates a vector or points cloud, rejects if p or P is not
%a vector or a singular value
function r = translate(p, P)
    r = Cartesian.transform(p,eye(3),P);
end
%rotate vector at an angle around x axis
function r = xrotate(p,xang,type)
    if nargin == 3
        r = Cartesian.rotate(p,xang,0,0,type);
    else
        r = Cartesian.rotate(p,xang,0,0);
    end
end
%rotate vector at an angle around y axis
function r = yrotate(p,yang,type)
    if nargin == 3
        r = Cartesian.rotate(p,0,yang,0,type);
    else
        r = Cartesian.rotate(p,0,yang,0);
    end
end
%rotate vector at an angle around z axis
function r = zrotate(p,zang,type)
    if nargin == 3
        r = Cartesian.rotate(p,0,0,zang,type);
    else
```

```
        r = Cartesian.rotate(p,0,0,zang);
    end
end
%find the inverse of a transformation
function Finv = inverse(r,p)

    Finv = 0;
    %if there are two inputs
    if nargin == 2
        [isP, p] = Cartesian.isVec(p);
        [row,col] = size(r);

        if ~(row == 3 && col == 3)
            fprintf('Invalid rotation matrix, need a ');
            fprintf('3x3 matrix for a 3D space\n');
            return;
        else
            R = r;
        end

        if ~isP
            fprintf('Invalid transformation\n');
            return;
        end
    %if there is only one combined transformation r
    else
        [R,P] = Cartesian.fDecompose(r);
    end

    %calculate inverse rotation/translation
    Rinv = R^-1;
    Pinv = -1*Rinv*P;

    Finv = [[Rinv; 0 0 0],[Pinv;1]];
end
%inverse frame transformation of a vector or cloud
function r = invTransform(vector,R,p)

    if nargin == 3
        F = Cartesian.inverse(R,p);
    else
        F = Cartesian.inverse(R);
    end

    r = Cartesian.transform(vector,F);
end
%average of 3D point cloud
function avr = cloudAvr(p)
    avr = [mean(p(1,:));
        mean(p(2,:))
        mean(p(3,:))];
end
%error of 3D point cloud set
function [avr,err] = cloudErr(p)
    avr = Cartesian.cloudAvr(p);
```

```
            %subtract average from all vectors
            err = p - avr(:,ones(1,size(p,2)));
        end
    end
    methods(Static, Access = private)
        %checks if an array is a 3-D vector
        function [cond, vector] = isVec(p)
            [r,c] = size(p);
            vector = p;
            %horizontal vector
            if r == 1 && c == 3
                cond = true;
                %returns vertical vector
                vector = p';
            %vertical vector
            elseif r == 3 && c == 1
                cond = true;
            else
                cond = false;
            end
        end
        %checks if a matrix is a 3-D cloud
        function [cond, cloud] = isCloud(p)
            cond = false;
            cloud = p;
            [r,c] = size(p);

            %if not a vector
            if ~Cartesian.isVec(p)
                %if already in correct orientation
                if r == 3
                    cond = true;
                %if not in correct orientation, but valid cloud
                elseif c == 3
                    cond = true;
                    cloud = p';
                end
            end
        end
    end
end
```

SetsReg.m

```
function F = SetsReg(a,b)
%     @param  a   3D point cloud
%     @param  b   transformed 3D point cloud
%     @return F   a 4x4 containing R,P

%   calculates the average location and
%   the error of each vector
%   step 1

    [avgA, qA] = Cartesian.cloudErr(a);
    [avgB, qB] = Cartesian.cloudErr(b);

    if size(a,2) ~= size(b,2)
        error('The clouds contained different amounts of points');
    end

%   Calculation for the Rotation portion of the transformation

%   summation of error
%   step 2 from paper
    H = zeros(3);
    for i = 1:size(a,2) % 1:num_columns
        H = H + qA(:,i)*transpose(qB(:,i));
    end

%   Calculation of the delta and the G matrix
%   step 3
    G = zeros(4);
    delta = zeros(1,3);
    delta = [H(2,3)-H(3,2), H(3,1)-H(1,3), H(1,2)-H(2,1)];
    G = [trace(H), delta;
        delta', H + H' - trace(H)*eye(3)];

%   Finding the quaternion by finding the largest eigenvalue of G and its
%   corresponding eigenvector
%   step 4
    eigvects = zeros(4);
    eigvals = zeros(4);
    [eigvects, eigvals] = eig(G);
    max_eig = -inf;
    max_col = 0;
    for i = 1:4
        if eigvals(i,i) > max_eig
            max_eig = eigvals(i,i);
            max_col = i;
        end
    end
    quat = eigvects(:,max_col);

%   Computing R using the quaternion
    q0 = quat(1); q1 = quat(2); q2 = quat(3); q3 = quat(4);
    R = [q0^2+q1^2-q2^2-q3^2, 2*(q1*q2-q0*q3), 2*(q1*q3+q0*q2);
        2*(q1*q2+q0*q3), q0^2-q1^2+q2^2-q3^2, 2*(q2*q3-q0*q1);
```

```
        2*(q1*q3-q0*q2), 2*(q2*q3+q0*q1), q0^2-q1^2-q2^2+q3^2];

%   Compute the translation vector
    P = avgB - R*avgA;

%   Combine total transformation into a 4x4
    F = Cartesian.rearrange(R,P);
end
```

circlefit3d.m

```
function [center,rad,v1n,v2nb] = circlefit3d(p1,p2,p3)
% circlefit3d: Compute center and radii of circles in 3d which are defined by
three points on the circumference
% usage: [center,rad,v1,v2] = circlefit3d(p1,p2,p3)
%
% arguments: (input)
%  p1, p2, p3 - vectors of points (rowwise, size(p1) = [n 3])
%               describing the three corresponding points on the same circle.
%               p1, p2 and p3 must have the same length n.
%
% arguments: (output)
%  center - (nx3) matrix of center points for each triple of points in p1,
p2, p3
%
%  rad    - (nx1) vector of circle radii.
%           if there have been errors, radii is a negative scalar ( = error
code)
%
%  v1, v2 - (nx3) perpendicular vectors inside circle plane
%
% Example usage:
%
%  (1)
%      p1 = rand(10,3);
%      p2 = rand(10,3);
%      p3 = rand(10,3);
%      [center, rad] = circlefit3d(p1,p2,p3);
%      % verification, result should be all (nearly) zero
%      result(:,1)=sqrt(sum((p1-center).^2,2))-rad;
%      result(:,2)=sqrt(sum((p2-center).^2,2))-rad;
%      result(:,3)=sqrt(sum((p3-center).^2,2))-rad;
%      if sum(sum(abs(result))) < 1e-12,
%       disp('All circles have been found correctly.');
%      else,
%       disp('There had been errors.');
%      end
%
% (2)
%       p1=rand(4,3);p2=rand(4,3);p3=rand(4,3);
%       [center,rad,v1,v2] = circlefit3d(p1,p2,p3);
%       plot3(p1(:,1),p1(:,2),p1(:,3),'bo');hold
on;plot3(p2(:,1),p2(:,2),p2(:,3),'bo');plot3(p3(:,1),p3(:,2),p3(:,3),'bo');
%       for i=1:361,
%           a = i/180*pi;
%           x = center(:,1)+sin(a)*rad.*v1(:,1)+cos(a)*rad.*v2(:,1);
```

```
%           y = center(:,2)+sin(a)*rad.*v1(:,2)+cos(a)*rad.*v2(:,2);
%           z = center(:,3)+sin(a)*rad.*v1(:,3)+cos(a)*rad.*v2(:,3);
%           plot3(x,y,z,'r.');
%       end
%       axis equal;grid on;rotate3d on;
%
%
% Author: Johannes Korsawe
% E-mail: johannes.korsawe@volkswagen.de
% Release: 1.0
% Release date: 26/01/2012

% Default values
center = [];rad = 0;v1n=[];v2nb=[];

% check inputs
% check number of inputs
if nargin~=3,
    fprintf('??? Error using ==> cirlefit3d\nThree input matrices are
needed.\n');rad = -1;return;
end
% check size of inputs
if size(p1,2)~=3 || size(p2,2)~=3 || size(p3,2)~=3,
    fprintf('??? Error using ==> cirlefit3d\nAll input matrices must have
three columns.\n');rad = -2;return;
end
n = size(p1,1);
if size(p2,1)~=n || size(p3,1)~=n,
    fprintf('??? Error using ==> cirlefit3d\nAll input matrices must have the
same number or rows.\n');rad = -3;return;
end
% more checks are to follow inside calculation

% Start calculation
% v1, v2 describe the vectors from p1 to p2 and p3, resp.
v1 = p2 - p1;v2 = p3 - p1;
% l1, l2 describe the lengths of those vectors
l1 = sqrt((v1(:,1).*v1(:,1)+v1(:,2).*v1(:,2)+v1(:,3).*v1(:,3)));
l2 = sqrt((v2(:,1).*v2(:,1)+v2(:,2).*v2(:,2)+v2(:,3).*v2(:,3)));
if find(l1==0) | find(l2==0), %#ok<OR2>
    fprintf('??? Error using ==> cirlefit3d\nCorresponding input points must
not be identical.\n');rad = -4;return;
end
% v1n, v2n describe the normalized vectors v1 and v2
v1n = v1;for i=1:3, v1n(:,i) = v1n(:,i)./l1;end
v2n = v2;for i=1:3, v2n(:,i) = v2n(:,i)./l2;end
% nv describes the normal vector on the plane of the circle
nv = [v1n(:,2).*v2n(:,3) - v1n(:,3).*v2n(:,2) , v1n(:,3).*v2n(:,1) -
v1n(:,1).*v2n(:,3) , v1n(:,1).*v2n(:,2) - v1n(:,2).*v2n(:,1)];
if find(sum(abs(nv),2)<1e-5),
    fprintf('??? Warning using ==> cirlefit3d\nSome corresponding input
points are nearly collinear.\n');
end
% v2nb: orthogonalization of v2n against v1n
dotp = v2n(:,1).*v1n(:,1) + v2n(:,2).*v1n(:,2) + v2n(:,3).*v1n(:,3);
v2nb = v2n;for i=1:3,v2nb(:,i) = v2nb(:,i) - dotp.*v1n(:,i);end
```

```
% normalize v2nb
l2nb =
sqrt((v2nb(:,1).*v2nb(:,1)+v2nb(:,2).*v2nb(:,2)+v2nb(:,3).*v2nb(:,3)));
for i=1:3, v2nb(:,i) = v2nb(:,i)./l2nb;end

% remark: the circle plane will now be discretized as follows
%
% origin: p1                    normal vector on plane: nv
% first coordinate vector: v1n  second coordinate vector: v2nb

% calculate 2d coordinates of points in each plane
% p1_2d = zeros(n,2); % set per construction
% p2_2d = zeros(n,2);p2_2d(:,1) = l1; % set per construction
p3_2d = zeros(n,2); % has to be calculated
for i = 1:3,
    p3_2d(:,1) = p3_2d(:,1) + v2(:,i).*v1n(:,i);
    p3_2d(:,2) = p3_2d(:,2) + v2(:,i).*v2nb(:,i);
end

% calculate the fitting circle
% due to the special construction of the 2d system this boils down to solving
% q1 = [0,0], q2 = [a,0], q3 = [b,c] (points on 2d circle)
% crossing perpendicular bisectors, s and t running indices:
% solve [a/2,s] = [b/2 + c*t, c/2 - b*t]
% solution t = (a-b)/(2*c)

a = l1;b = p3_2d(:,1);c = p3_2d(:,2);
t = 0.5*(a-b)./c;
scale1 = b/2 + c.*t;scale2 = c/2 - b.*t;

% centers
center = zeros(n,3);
for i=1:3,
    center(:,i) = p1(:,i) + scale1.*v1n(:,i) + scale2.*v2nb(:,i);
end

% radii
rad = sqrt((center(:,1)-p1(:,1)).^2+(center(:,2)-p1(:,2)).^2+(center(:,3)-
p1(:,3)).^2);
```

columnReformat.m

```
function r = columnReformat(column, n)
%Convert column of data to matrix of 3 x N/3
%   Convert a Nx1 column of data into a processed column of n x N/n.
%   Example: input ([1;2;3;4;5;6], 3)
%            output [1 2 3; 4 5 6]

    N = length(column);
    r = ones(n, N/n);
    for i = 1:N/n
        adjust = n*(i-1);
        r(:,i) = column(1+adjust:n+adjust, 1);
    end
end
```

intra2pre.m

```
clc;
clear all;

% Voxel values
Xscale = 2.078;
Yscale = Xscale;
Zscale = 0.8;
p = 1;
t = p;

%User changed CT fiducial locations
ct = round([65 115 120
119 173 223
173 120 155
108 119 346
121 108 16]);
ct = ct';


% Read in Polaris points based on specific formating
% polarisData2pt takes name in String as an input
polaris = zeros(5,3);
for i = 1:5
    [polaris(i,1) polaris(i,2) polaris(i,3)] =
polarisData2Pt(strcat('Pig',num2str(p),'/fiducial',num2str(i),'/P0A-
3910F400.txt'));
    %[polaris(i,1) polaris(i,2) polaris(i,3)] =
polarisData2Pt(strcat('Pig3ErrorCheck/fiducial',num2str(i),'/P0A-
3910F400.txt'));

end
%adjust for y
polaris(:,2) = -1*polaris(:,2);
polaris = polaris';

%convert ct voxel values to proper dimensions
ctActual(1,:) = Xscale*ct(1,:);
ctActual(2,:) = Yscale*ct(2,:);
ctActual(3,:) = Zscale*ct(3,:);

% Find relationship between the polaris and CT
F = SetsReg(polaris, ctActual);

% Find translated CT
transP = Cartesian.transform(polaris,F);
transPCT(1,:) = transP(1,:)/Xscale;
transPCT(2,:) = transP(2,:)/Yscale;
transPCT(3,:) = transP(3,:)/Zscale;
transPCT = round(transPCT);

% Read in Tongue points based on specific formating
% TongueData2pt takes name in String as an input
```

```
tongue = zeros(4,3);
for i = 1:4
    [tongue(i,1) tongue(i,2) tongue(i,3)] =
TongueData2Pt(strcat('Pig',num2str(p),'/tongue', num2str(i),'/P0A-
3910F400.txt'));
    %S[tongue(i,1) tongue(i,2) tongue(i,3)] =
TongueData2Pt(strcat('Pig3ErrorCheck/tongue', num2str(i),'/P0A-
3910F400.txt'));

end

%expand and transform the polaris intraop points
tongue(:,2) = -1*tongue(:,2);
tongue = tongue';
transT = Cartesian.transform(tongue,F);

%convert back into voxel values
ctPath(1,:) = transT(1,:)/Xscale;
ctPath(2,:) = transT(2,:)/Yscale;
ctPath(3,:) = transT(3,:)/Zscale;
ctPath = round(ctPath);

% %Uncomment to see the translate polaris versus the CT in the
% %CT coordinate
% figure
% scatter3(transPCT(1,:),transPCT(2,:),transPCT(3,:),'x');
% labels = cellstr( num2str([1:5]') );
% text(transPCT(1,:),transPCT(2,:),transPCT(3,:),labels);
% hold on
% scatter3(ct(1,:),ct(2,:),ct(3,:),'o');
% xlabel('X');
% ylabel('Y');
% zlabel('Z');
% legend('Polaris','CT');
% title('Polaris and CT Matching');
% hold off

% %Uncomment to see the polaris versus tongue in original
% figure
% scatter3(polaris(1,:),polaris(2,:),polaris(3,:),'x');
% labels = cellstr( num2str([1:5]') );
% text(polaris(1,:),polaris(2,:),polaris(3,:),labels);
% hold on
% scatter3(tongue(1,:),tongue(2,:),tongue(3,:),'o');
% xlabel('X');
% ylabel('Y');
% zlabel('Z');
% legend('Polaris','Tongue');
% title('Polaris and Tongue');
% hold off

% %Uncomment to see the translate polaris fiducial and tongue versus
% %the CT in the CT coordinate
% figure
% scatter3(transPCT(1,:),transPCT(2,:),transPCT(3,:),'x');
% labels = cellstr( num2str([1:5]') );
```

```
% text(transPCT(1,:),transPCT(2,:),transPCT(3,:),labels);
% hold on
% scatter3(ct(1,:),ct(2,:),ct(3,:),'o');
% scatter3(ctPath(1,:),ctPath(2,:),ctPath(3,:),'o');
% xlabel('X');
% ylabel('Y');
% zlabel('Z');
% legend('Polaris','CT','Tongue');
% title('Polaris and CT Matching');
% hold off

% Records results
fID = fopen('RESULTS.txt','wt');
fprintf(fID, 'ct\n');
fprintf(fID, '%i %i %i\n', ct);
fprintf(fID, '\npolaris\n');
fprintf(fID, '%i %i %i\n', transPCT);
fprintf(fID, '\ntongue\n');
fprintf(fID, '%i %i %i\n', ctPath);
fclose(fID);
```

polarisData2Pt.m

```
% Retrieve polaris point based on data
function [X Y Z] = polarisData2Pt(filename)
    close all;
    %% Retrieve Data
    data = fileread(filename);
    oov = strfind(data,'OOV');
    data = data(oov+3:end);
    data = regexprep(data,' ','');
    data = sscanf(data,'%f,%f,%f,%f,%f,%f,%f,%f,%f,%f');
    data = columnReformat(data,10);
    [~,w] = size(data);

    x1 = data(6,1);
    y1 = data(7,1);
    z1 = data(8,1);
    dVec = [];

    %% Estimate diameter by using longest distance
    for i = 1:w
        dVec = [dVec, distance(x1,y1,z1,data(6,i),data(7,i),data(8,i))];
    end

    %% Fit Circle
    where = find(dVec == max(dVec));
    % Circle fit provided by Johannes Korsawe
    % http://www.mathworks.com/matlabcentral/fileexchange/34792-circlefit3d-
fit-circle-to-three-points-in-3d-space
    [c,~,~,~] = circlefit3d([x1,y1,z1], [data(6,5),data(7,5),data(8,5)],
[data(6,where),data(7,where),data(8,where)]);
    X = c(:,1);
    Y = c(:,2);
```

```
    Z = c(:,3);
end

function d = distance(x,y,z, x1, y1, z1)
    d = sqrt((x-x1)^2 + (y - y1)^2 + (z-z1)^2);
end
```

TongueData2Pt.m

```
function [X Y Z] = TongueData2Pt(filename)
    close all;
    %% Retrieve Data from Polaris Recording
    data = fileread(filename);
    oov = strfind(data,'OOV');
    data = data(oov+3:end);
    data = regexprep(data,' ','');
    data = sscanf(data,'%f,%f,%f,%f,%f,%f,%f,%f,%f,%f');
    data = columnReformat(data,10);

    %% Average Recorded Values
    X = data(6,:);
    Y = data(7,:);
    Z = data(8,:);
    X = mean(X);
    Y = mean(Y);
    Z = mean(Z);
end
```