VIOLINS: Visualizing Intra-Operative Line-of-cut In Nasal Surgery

Felix Jonathan and Michael Norris
Mentors: Narges Ahmidi, Dr. Masaru Ishii, Dr. Lisa Ishii

May 6, 2016

Abstract

Resident surgeons observe surgeries performed by experienced surgeons as part of their training. For surgeries performed in confined spaces, like septoplasty which is performed inside the nose to correct a deviated septum, it is difficult for an onlooker to get a good feel for how the surgeon is moving surgical tools and performing maneuvers, and thus observing these surgeries is less instructive for surgical residents. During septoplasty, the surgeon first releases (i.e. dissects) the cartilage (septum) from the surrounding tissue and then removes the deviated cartilage for treatment using a scissor. The cut of the cartilage should be performed with extreme dexterity. A larger-than-necessary removal of the cartilage will leave behind only a shallow cartilage left under the nose bridge, causing potential collapse of the nose bridge which is a harmful condition for patients. A smaller-than-necessary removal can potentially leave behind defected parts of the cartilage and consequently cause an incomplete treatment. Thus it is important that surgical residents are able to observe and learn how deep experienced surgeons properly cut the septal cartilage with respect to the nasal bridge, a factor that is unobservable in conventional septoplasty procedures. In this paper, we design algorithms that predict the line-of-cut for surgical scissors on the surface of the septal cartilage in real-time. We also discuss our implementation of a system that incorporates these algorithms and provides a visualization of the septal cartilage, surgical scissors, and the line-of-cut of the scissors, and we provide an error analysis for the individual algorithm components.

1 Statement of Purpose

Surgical residents gain competency through observing surgeries and through performing surgeries. Due to the cramped environment of an operating room, it is often difficult for an onlooker to observe how a surgeon is performing certain motions with surgical tools.

Septoplasty is a type of nasal surgery that corrects a deviated septum which straightens the nose and improves breathing. The importance of increasing the skill and intution of

surgical residents who learn how to perform septoplasty is apparent, since septoplasty is a common surgery. It is estimated that 260,000 septoplasties were performed in 2006 [1].

During septoplasty, the surgeon first releases (i.e. dissects) the cartilage (septum) from the surrounding tissue and then removes the deviated cartilage for treatment using a scissor. The cut of the cartilage should be performed with extreme dexterity. The region left behind is called the L-strut region. A larger-than-necessary removal of the cartilage will leave behind only a shallow cartilage left under the nasal bridge, causing potential collapse of the nose bridge which is a harmful condition for patients. A smaller-than-necessary removal can potentially leave behind defected parts of the cartilage and consequently cause an incomplete treatment.

The following diagram shows the L-strut region and illustrates how the L-strut region is important to retaining the shape of a nose.

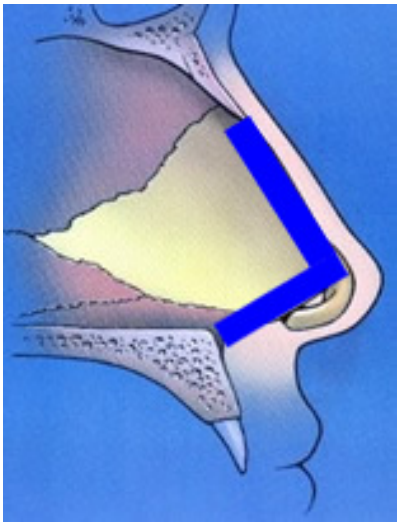


Figure 1: L-strut region
http://sandiegorhinoplasty.com/wp-content/uploads/2011/07/RSD-Strut.jpg

The L-strut region needs to be appropriately preserved in the cartilage, and surgical residents need to learn through observation how senior surgeons cut the septal cartilage to preserve the L-strut region. The acceptable width for this region is between 10 and 15 mm [3, p. 1].

During septoplasty, it is difficult for a surgical resident to observe a senior surgeon's tool movements inside of the nose because the surgeon's tool motions are obstructed by the

patient's nose. Thus it is important that surgical residents are able to observe and learn how deep experienced surgeons properly cut the septal cartilage with respect to the nasal bridge, a factor that is unobservable in conventional septoplasty procedures.

The importance of providing appropriate visualization of the septal cartilage is noted in the paper by Russell and Kangelaris. The paper notes that different visual techniques can lead to different dimensions of the L-struc region. [3, p. 1]

While experienced surgeons can do this step without any difficulty, residents might have a lot of trouble due to their limited experiences in this procedure. Additionally, the surgeons who observe surgical residents performing this procedure also have difficulty evaluating the resident's skill level.

The need to provide motivates the need for the development of a system which is capable to provides visual representation of the septal nasal cartilage surface, the surgical tool pose, and the estimated line of cut to the observing surgeons and the operating surgeon.

2 Technical Approach

In this paper, we describe our approach to developing a real-time visual feedback system for nasal surgery. Our approach uses an NDI Aurora EM Tracker, an EM sensor mounted on a pair of surgical scissors, an EM Pointer, and an EM sensor attached to a patient's forehead.

To visualize the septal nasal cartilage, we approximate the shape of the cartilage as a plane and fit a convex hull to it. This assumption is made for easier verification of the accuracy of the line-of-cut prediction. In the future development, it is possible to use mesh instead of flat plane to have a better approximation of the septum, especially the deviated one, since the concepts that concerns the prediction does not change when we use mesh instead of flat plane.

After approximating the shape of the septum cartilage, we want to estimate the line of cut from the surgical scissor. To achieve accurate line of cut predition, it is necessary to do scissor model training first, given the variety of scissor type and the uncertainty of the sensor placement. There are 2 main scissor property necessary for line of cut estimation, which is scissor pinch point position relative to the scissor EM sensor and the scissor blade plane normal direction.

By having the scissor property, it is possible to predict the line of cut by doing plane to plane intersection (or plane to mesh intersection if septum cartilage is represented as mesh) between scissor blade plane and cardboard plane.

This following figure shows an example of our system setup in Operating Room. "W" letter describes the World/Aurora, "S" for Scissor, "P" for pointer, and "F" for forehead coordinate system.

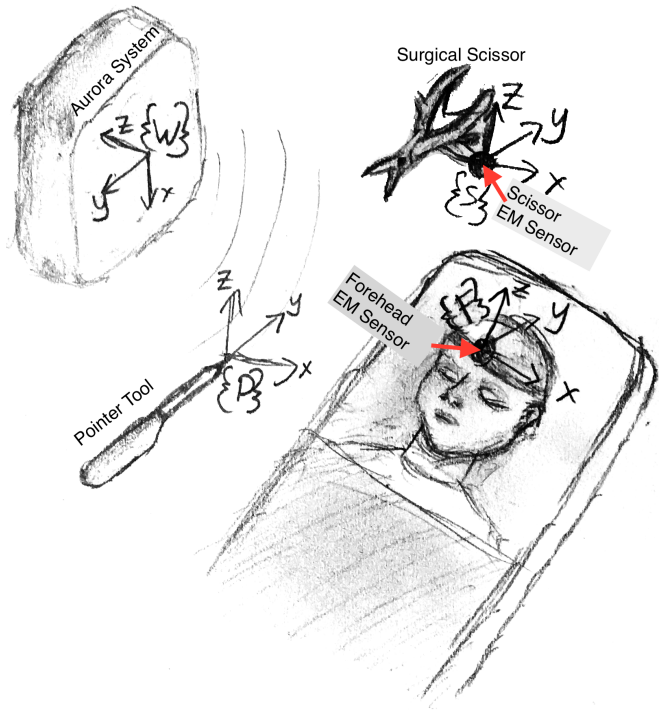


Figure 2: Operating Room Setup and Frame Annotation

2.1 Plane Estimation

This subsection describes the OR protocol that a surgeon must perform to generate the septum surface model that is used in the septum line-of-cut prediction. This document also contains a mathematical model describing how this data is used.

In this subsection we use "world coordinate system," and frame transformations superscripted with "w," to mean the coordinate system of the NDI Aurora system.

2.1.1 Setup

- EM sensor attached rigidly on patient's forehead
- EM pointer tool, pivot calibrated with the Aurora NDI Tracker software

2.1.2 Input Data

- T_F^W: EM sensor on patient's forehead provides the patient's forehead local coordinate frame transformation relative to the world coordinate frame (Aurora).
- P_{PT}^W: EM sensor on the pointer provides the pointer frame transformation relative to the world coordinate frame (Aurora). The translation data from the frame transformation is extracted while rotation data is neglected.

2.1.3 OR Protocol

1. The points on the septal nasal cartilage should be independent of the patient's head, therefore the effect of patient's head movement need to be eliminated. To achieve this, the EM sensor must be attached securely to the patient's forehead so that it does not move during the data collection. In the data collection process, the patient's forehead EM Sensor will collect points T_F^W at times $i : t$, which will be used to transform all points to the patient forehead coordinate system.
2. The L-strut region of the nose, which is a very critical structure for septoplasty procedure is tightly tied to the nasal bridge position (see Fig. 1). To establish the estimated position of the L-strut, the nasal bridge (NB) needs to be traced with the EM pointer. At time i, $(T_F^W)^{-1} P_{N_B}^W$ is the point in nasal bridge in the patient's forehead reference frame.

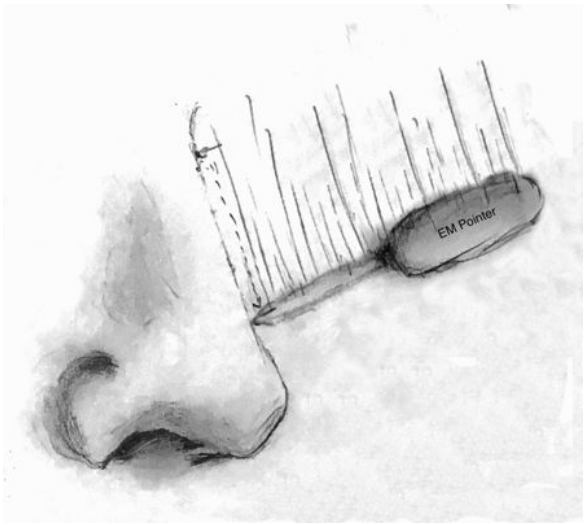


Figure 3: Pointer tracing along outside of nose from nasal bridge to the tip of the nose

3. To approximate the shape of the septal nasal cartilage (NS), points in that region are needed. This data can be generated by tracing the septum surface using the EM pointer tool with a zig-zag pattern to generate a point cloud. The point cloud is used to generate a convex hull or mesh that will approximate the shape of cartilage. Since the cartilage points collected from the EM pointer ($P^W_{N_S}$) is still in world coordinate system, it needs to be transformed to the patient's forehead coordinate system at time i (by doing a frame transformation from the data in the sensor from 1 at time i) with the transformation: $(T^W_F)^{-1}P^W_{N_S} = P^F_{N_S}$.

Figure 4: Pointer tracing inside the nose along the septum plane

2.1.4 PseudoCode list

This function is used to change the reference frame of a point, vector, or transformation:

Function *ChangeReference (Data input, Transform reference)*
  Data: point(P^W_{data}), vector($\vec{v}^W_{data}$), or transformation(T^W_{data})) and the desired reference frame in the world coordinate system (T^W_{ref})
  Result: Data in reference coordinate system: P^W_{data} or $\vec{v}^W_{data}$ or T^{ref}_{data}
  switch *Data type* **do**
    case *Data type == transform*
      $T^{ref}_{data} = (T^W_{ref})^{-1} T^W_{data}$ **return** T^{ref}_{data}
    end
    case *Data type == point*
      $P^{ref}_{data} = (T^W_{ref})^{-1} P^W_{data}$ **return** P^{ref}_{data}
    end
    case *Data type == vector*
      $\vec{d}^{ref}_{data} = (R^W_{ref})^{-1} \vec{d}^W_{data}$ **return** $\vec{d}^{ref}_{data}$
    end
  endsw

Algorithm 1: Change Reference Frame

Estimating Plane Parameters The first iteration of our project generates a simplified version of the septum cartilage, which treats the cartilage as a plane. Following is the pseudocode for the algorithms to generate the plane which specify the input and output data, and the PCL methods that are used to generate the plane.

Function *SeptumPlaneGeneration (Array of pointArray Input, TransformArray WorldToFaceRef)*

Data: an array of nasal septum cartilage traced points $P_{N_S}^W[1..n]$ (n: number of points collected from EM sensor) from EM Pointer; collection of transformations $T_F^W[1..n]$ from patient's forehead EM Sensor.

Result: A Septum Plane in the form $ax + by + cz + d = 0$

Initialize array of points in patient's forehead coordinate system $P_{N_S}^F[1..n]$

forall the $P_{N_S}^W$ *in* $P_{N_S}^W[1..n]$ **do**

  $P_{N_S}^F[i] = \text{ChangeReference}(P_{N_S}^W[i], (T_F^W[i])^{-1})$

end

Initialize and convert the array of points in patient's forehead coordinate system to PCL point cloud data. Run Point Cloud Library's function pcl::SACSegmentation<pcl::PointXYZ>::segment

Algorithm 2: Septum Shape Estimation

Estimating Septum Plane Convex Hull After we estimate the plane normal, we can reproject the points from the nasal bridge to the estimated plane and estimate the shape of septum cartilage with a convex hull between the septum plane and the nasal bridge.

Function *SeptumConvexHullEstimation (Array of pointArray nasalBridgePoints, Array of pointArray septumCartilagePoints, TransformArray WorldToFaceRef_nasal, TransformArray WorldToFaceRef_septum)*

Data: an array of nasal bridge points $P^W_{N_B}[1..n]$ (n: the number of points collected from the EM sensor) from the EM Pointer;
a collection of transformations $T^W_{F_{NB}}[1..n]$ from the patient's forehead EM Sensor when doing the nasal bridge trace;
an array of nasal septum cartilage points $P^W_{N_S}[1..m]$ (m: the number of points collected from the EM sensor) from the EM Pointer;
a collection of transformations $T^W_{F_{NS}}[1..m]$ from the patient's forehead EM Sensor when doing the nasal septum cartilage trace.
Result: Convex hull points $P^F_{convexHull}[1..n]$

Initialize array of nasal bridge points in the patient's forehead coordinate system $P^F_{N_B}[1..n]$ **forall the** $P^W_{N_B}$ *in* $P^W_{N_B}[1..n]$ **do**

  $P^F_{N_B}[i]$ = ChangeReference$(P^W_{N_B}[i], (T^W_{F_{NB}}[i])^{-1})$

end

Calculate the septum plane parameters with SeptumPlaneGeneration$(P^W_{N_S}[1..m], T^W_{F_{NS}}[1..m])$

Initialize and convert the array of nasal bridge trace points in the patient's forehead coordinate system to a PCL point cloud.

Initialize and convert the array of nasal septum cartilage trace points in the patient's forehead coordinate system to a PCL point cloud.

Use pcl::ProjectInliers<pcl::PointXYZ>::filter to project the point cloud from the nasal bridge trace to the estimated septum plane.

Use pcl::ProjectInliers<pcl::PointXYZ>::filter to project the inliers of SACsegmentation from the septum plane estimation to the estimated septum plane.

Concatenate the projected nasal bridge trace point cloud with the projected inliers of the septum plane estimation

Use pcl::ConvexHull<pcl::PointXYZ>::reconstruct to estimate the convex hull of the nasal septum cartilage.

Convert the pcl point cloud result from the pcl convex hull reconstruct method to an array of points $P^F_{convexHull}[1..n]$.

Algorithm 3: Septum Convex Hull Estimation

Generating The Septum Mesh The second iteration of our project will generate a septum mesh. Following is the psuedocode for the algorithms to generate the mesh which specify the input and output data, and the PCL methods that will be used to generate the mesh.

1 **Function** *Septum Mesh Generation (Array of pointArray Input, TransformArray WorldToPlaneRef)*

Data: an array of points $P_{N_B}^W$, collected with the EM Pointer by tracing the nasal bridge at times $1:t$
an array of points $P_{N_S}^W$, collected with the EM Pointer by tracing the nasal septum at times $1:t$
an array of points P_F^W, collected with the EM Sensor positioned on the patient's forehead at times $1:t$
Result: A Septum Plane in the form $ax + by + cz + d = 0$

2 Run PCL's function
pcl::GreedyProjectionTriangulation<pcl::PointNormal>::reconstruct

Algorithm 4: Septum Mesh Generation From Data Collected By Surgeon

2.2 Scissor Calibration

This subsection describes the steps that must be performed to generate the scissor model that is used in the septum line of cut prediction. This does not need to be done every time the surgery is performed, but different models of scissors will have different scissor models.

In this subsection there are three frames that will be used in the frame transformation: "S" for Scissor EM sensor frame, "CB" for the Cardboard/Phantom EM sensor frame, "PT" for the pointer EM sensor frame, and "W" for the world/aurora coordinate system.

There are two scissor parameters that need to be calculated from the scissor calibration. First, we must determine the normal of the scissor plane of cut, which is the normal of the scissor's blade for scissors with flat blades. Second, we must determine the scissor pinch point position. The calibration for the scissor's plane of cut normal is done by tracing the surface of one of the the scissor's blades with the pointer tool to estimate the scissor blade's plane normal. The calibration for the pinch point position is done by drawing training lines on the phantom to estimate the scissor's pinch point position, tracing them with the pointer tool, and then placing the scissor blades along those lines like the scissors are cutting them.

These two scissor parameters are referred to as a scissor model.

2.2.1 Setup

- Scissors with EM sensor attached rigidly
- Phantom (flat cardboard) with EM sensor attached rigidly
- EM pointer tool, which has been pivot-calibrated with Aurora software

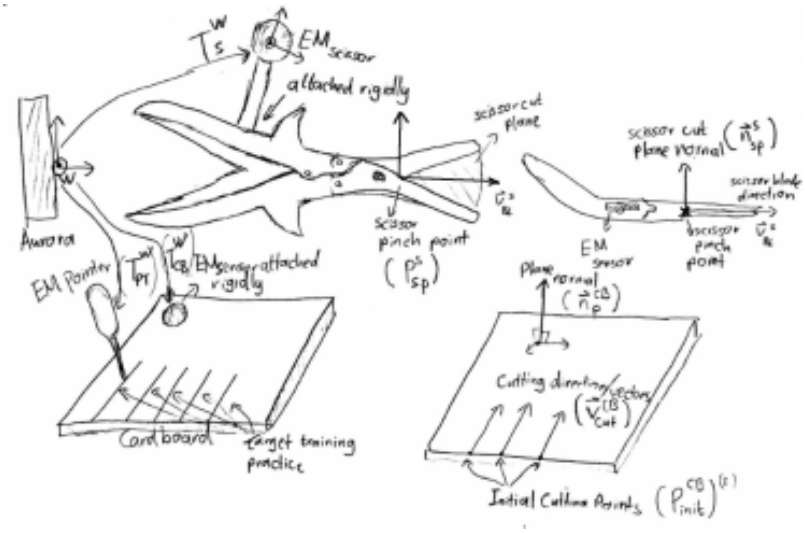


Figure 5: Schematic Diagram Of Scissor Training Setup

2.2.2 Input Data

- T_S^W: The EM sensor on the scissors provides a transformation between the world coordinate frame (Aurora) with the scissor local coordinate frame.
- T_{CB}^W: The EM sensor on the scissors provides a transformation between the world coordinate frame (Aurora) with the cardboard local coordinate frame.
- P_{PT}^W: The EM sensor provides the end point position in the world coordinate frame (Aurora). The sensor provides the frame transformation to the EM pointer, however the rotation of the pointer is not used in this training.

2.2.3 Data Collection Procedure

1. The scissor cutting plane normal vector can be estimated by fitting a plane to the scissor blade's point cloud. To get the scissor blade's point cloud data, trace the scissor blade surface with pointer tip while recording the scissor EM sensor data to transform all points to the scissor coordinate system. Repeat this data collection several time to make sure the scissor cutting plane normal is consistent.
2. Given that all points that will be used for the scissor calibration should be on the cardboard surface, the cardboard surface's plane parameters need to be estimated. To estimate the cardboard's plane parameters, the cardboard surface's point cloud needs to be collected. The point cloud can be collected by tracing the cardboard surface

with the pointer tip while recording the cardboard EM sensor data to transform all points to the cardboard coordinate system.

Next we need to determine the translation to the pinch point. We will draw multiple lines on the phantom (cardboard). The start of the line will be referred to as the initial position.

1. To determine the initial position of each of the cutting points, move the pointer tip to each initial cutting point. Record the data of each point individually while recording the cardboard EM sensor data to transform all points to the cardboard coordinate system.

2. To estimate the line vector that starts from the initial cutting point, use a ruler to trace along each line with pointer tip. Record the data of each line vector individually while recording the cardboard EM sensor data to transform all points to the cardboard coordinate system.

3. The scissor pinch point estimation is done by moving the scissors such that the scissor pinch point is coincident with the initial cutting point position on the cardboard, and then record the scissor and cardboard EM sensor data. This sensor data will be used to transform the initial point position on the cardboard to the scissor's coordinate system. Given that the scissor pinch point is coincident to the initial cutting point, the transformed initial cutting point will provide an approximation to the scissor pinch point position.

4. To estimate the error of the scissor plane normal estimation, move the scissors to each initial cutting point positions and align the scissor such that it will cut the target line. Record the data of the scissor's EM sensor in that position for each traced line individually. The scissor cutting plane error can be calculated by estimating the difference between the direction of the line of cut prediction (from the cross product between the scissor cutting plane normal and the cardboard plane normal) and the actual traced line direction.

2.2.4 PseudoCode list

This subsection contains pseudocode that will be used to perform scissor model training, which specify the input, output data, and some mathematical calculations.

This function is used to generate an estimated 3D best line fit from a point cloud:

Function *CalculateVectorFromPointCloud (pointArray Input, transformArray worldToRef)*

Data: Point cloud $P^W_{data}[1..n]$ (n: number of points), array of transformations from the world to the reference frame T^W_{ref}

Result: a vector that represents line direction $\vec{v}^{ref}_{line}$

Initialize $P^{ref}_{data}[1..n]$

forall the $P^W_{data}[i]$ *in* $P^W_{data}[1..n]$ **do**

  $P^{ref}_{data}[i] = \text{ChangeReferenceFrame}(P^W_{data}[i], T^W_{ref}[i])$

end

Calculate the mean of the point cloud $P^{ref}_{data}[n]$: $\overline{P}^{ref}_{data}$

Initialize $\widetilde{P}^{ref}_{data}[1..n]$

forall the $P^{ref}_{data}[i]$ *in* $P^{ref}_{data}[1..n]$ **do**

  $\widetilde{P}^{ref}_{data}[i] = P^{ref}_{data}[i] - \overline{P}^{ref}_{data}$

end

Calculate the Covariance Matrix of C:

$$C = \frac{(\widetilde{P}^{ref}_{data}[n])^T(\widetilde{P}^{ref}_{data}[n])}{n-1}$$

Calculate SVD of C to get the first eigenvector of the point cloud, which represents $\vec{v}^{ref}_{line}$

return $\vec{v}^{ref}_{line}$

Algorithm 5: Estimating A Vector From A Point Cloud

This function is used to estimate the initial cutting point from a point cloud:

Function *estimateScissorPinchPoint (Training Lines, Array of Transformations from Scissor to Cardboard)*

Data: Collection of point clouds $P^{CB}_{data}[1..m][1..n]$ (m: number of training lines, n: points in one initial cutting point), collection of transformations to training line point cloud $T^{CB}_S[1..m]$

Result: the pinch point of the scissors in the scissor coordinate system P^s_{sp}

Initialize Array of Points $\overline{P}^s_{data}[1..m]$

forall the $P^C_{data}B[i]$ *in* $P^C_{data}B[1..m]$ **do**

  $\overline{P}^S_{data}[i] = \text{ChangeReference}($ Mean of the point cloud $P^C_{data}B[i][1..n], (T^{CB}_S[i])^{-1})$

end

Algorithm 6: Estimating the Scissor Pinch Point From the Point Clouds of the Initial Cut Points

This function is used to validate the scissor cutting plane normal:

Function *validateScissorNormal (vector LineOfCutDirection, normal scissorNormal, normal PlaneRef, TransformArray WorldToScissor, TransformArray WorldToPlaneRef)*

Data: Estimated line direction $\vec{v}_{cut}^{ref}$, scissor normal $\vec{n}_{sp}^{S}$, plane normal data $\vec{n}_{P}^{ref}$, collection of transforms $T_S^W[1..n], T_{ref}^W[1..n]$ when doing the scissor cut

Result: $\vec{v}_{cut-est}^{ref}$, angular error ε

Initialize $\vec{v}_{cut-est}^{ref}[1..n]$

forall the $T_S^W[i]$ *in* $T_S^W[1..n]$ **do**

  T_S^{ref} = ChangeReference$(T_S^W[i], T_{ref}^{W\ -1}[i])$

  $\vec{v}_{cut-est}^{ref}[i]$= $\vec{n}_P^{ref}$ ×ChangeReference$(\vec{n}_{sp}^{S}, T_S^{ref}\ )$

end

Calculate the mean of collection of the vector $\vec{v}_{cut-est}^{ref}[1..n]$: $\overline{\vec{v}}_{cut-est}^{ref}$

Calculate the angular error of the vector (assumming vector is already unit vector)

$\varepsilon = acos(\overline{\vec{v}}_{cut-est}^{ref} \circ \vec{v}_{cut}^{ref})$ **return** $\overline{\vec{v}}_{cut-est}^{ref}, \varepsilon$

Algorithm 7: Estimating Scissor plane of cut normal error

2.2.5 Data Processing

1. Use the function ChangeReference to transform all collected data (the scissor transformation and the pointer tip point cloud result from tracing the cardboard surface and a training line) into the cardboard coordinate system.

2. Estimate the cardboard plane normal from the point cloud collected when tracing the cardboard surface by using the function SeptumSurfaceGeneration described in the septum tracing procedure.

3. Estimate the line of cut vectors from the point cloud from the line tracing with the function CalculateVectorFromPointCloud

4. Estimate the scissor pinch point from the initial cutting position data from the point clouds collected from the initial cutting point trace by using function estimateScissorPinchPoint.

5. Estimate the scissor normal from the point cloud collected when tracing the scissor blade surface by using the function SeptumSurfaceGeneration described in the subsection on the Septum Tracing Procedure.

6. Validate the scissor normal data with the function validateScissorNormal from the estimated line of cut vectors, the cardboard plane, and the transformation from the cardboard sensor to the scissors.

2.3 Line of Cut Prediction

Using similar concepts from the scissor training, it is possible to generate the prediction of the line of cut of surgical scissors.

2.3.1 Setup

1. Scissor with EM sensor attached rigidly
2. Patient / Phantom with EM sensor attached rigidly
3. EM pointer tool for generating septum surface data

2.3.2 Input Data

1. scissor pose from EM data (T_S^W)
2. patient head pose from EM data (T_F^W)
3. generated septum surface normal ($\vec{n}_{ss}^F$) and plane equation of the septum ($ax + by + cz + d = 0$)
4. scissor training data ($P_{sp}^s, \vec{n}_{sp}^s$)
5. scissor blade length ($BL \in \mathbb{R}$)

2.3.3 PseudoCode list

This function is used for finding the point and vector (a parameterization of a line) that result from the intersection of two planes defined in the same coordinate system.

Function *FindIntersection (plane paramater1, plane parameter2)*

Data: Plane parameter1: (a_1, b_1, c_1, d_1) and Plane parameter2: (a_2, b_2, c_2, d_2)

Result: $P_{intersection}, \vec{v}_{intersection}$, Validity (boolean)

Initialize $\vec{n}_1 = [a_1, b_1, c_1]^T$, $\vec{n}_2 = [a_2, b_2, c_2]^T$

if $||n_1 - n_2|| < \varepsilon$ **then**

Print warning: "The planes are parallel to each other, therefore there is no possible line of cut."

return None, None, Valid=false

end

else

$\vec{v}_{intersection} = \vec{n}_1 \times \vec{n}_2$

Initialize plane parameter3 as a plane with $\vec{n}_3 = \vec{v}_{intersection}$ and $d_3 = 0$.

Initialize matrix $X = [-d_1, -d_2, -d_3]^T$

Initialize matrix:

$$M = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{bmatrix}$$

Since $\vec{n}_1, \vec{n}_2$, and $\vec{n}_3$ are always linearly independent, M will have an inverse.

$P_{intersection} = M^{-1} * X$

return $P_{intersection}, \vec{v}_{intersection}, Valid = true$

end

Algorithm 8: Finding Line Intersection between 2 planes

This function is used for finding a point projection to a 3D line defined in the same coordinate system

Function *PointToLineProjection (point pointA, point anyPointInLineB, vector LineB)*

Data: point A $P_A = (x_a, y_a, z_a)$, anyPointInLineB $P_0 = (x_0, y_0, z_0)$, and vector of lineB: $\vec{b}$

Result: $P_{projection}$

return $P_{projection} = P_A = P_0 + \frac{(P_A - P_0) \circ \vec{b}}{\vec{b} \circ \vec{b}} * \vec{b}$

Algorithm 9: Finding point projection on 3D line

This function is used for finding the line end point based on an initial line point, line magnitude, and line direction defined in the same coordinate system

1 **Function** *FindLineEndPoint (point pointA, realNumber lineMagnitude, vector lineB)*
  Data: point A $P_A = (x_a, y_a, z_a)$, lineMagnitude $l \in \mathbb{R}$, and vector of lineB: $\vec{b}$
  Result: $P_{endPoint}$
2 **return** $P_{endPoint} = P_A + \frac{\vec{b}}{||\vec{b}||} * lineMagnitude$

Algorithm 10: Finding a point projection on 3D line

This function is used for finding the line of cut prediction based on the scissor pinch point position, scissor plane normal, septum plane normal, scissor sensor pose and patient's forehead sensor pose defined in various coordinate system.

Function *GenerateLineOfCut (point scissorPinchPoint, vector scissorNormal, plane septumPlaneParameters, transform scissorSensorPose, transform faceSensorPose, realNumber bladeLength)*

Data: scissorPinchPoint: P_{sp}^{S}, scissorNormal $\vec{n}_{sp}^{S}$, septumPlaneParameters: (a^f, b^f, c^f, d^f), scissorSensorPose T_S^W, faceSensorPose T_f^W, bladeLength Bl

Result: Scissor pinch point in septum plane coordinates P_{sp}^{f}, a pair of points defining the of line of cut $P_{start}^{f}, P_{end}^{f}$, boolean variable defining lineValidity $valid$

Calculate scissor pose relative to the patient's forehead coordinate system. $T_S^f = ChangeReference(T_S^W, (T_f^W)^{-1})$

Calculate scissorPinchPoint position and scissorNormal relative to the patient's forehead coordinate system. $P_{sp}^{f} = ChangeReference(P_{sp}^{S}, T_S^f)$, $\vec{n}_{sp}^{f} = ChangeReference(\vec{n}_{sp}^{S}, T_S^f)$.

Initialize scissor plane parameters: $scissorPlaneParameters = (a^f, b^f, c^f, d^f)$. $(a^f, b^f, c^f) = \vec{n}_{sp}^{f}$ and $d_f = -\vec{n}_{sp}^{f} * P_{sp}^{f}$

Initialize the point that exists in both scissor and patient's forehead plane ($P_{intersection}^{f}$), and line of cut vector ($\vec{v}_{cut}^{f}$).

Calculate point, vector, and validity of the line. $P_{intersection}^{f}, \vec{v}_{cut}^{f}, valid = FindIntersection(scissorPlaneParameters, septumPlaneParameters)$

if $valid == false$ **then**

  return $P_{sp}^{f}, P_{sp}^{f}, P_{sp}^{f}, valid$

end

else

  Calculate the scissor initial cutting point: $P_{start}^{f} = PointToLineProjection(P_{sp}^{f}, P_{intersection}^{f}, \vec{v}_{cut}^{f})$.

  Calculate squaredLineMagnitude: $lineMagnitude^2 = Bl^2 - (P_{start}^{f} - P_{sp}^{f})^2$

  if $squaredLineMagnitude \leq 0$ **then**

    valid = false.

    return $P_{sp}^{f}, P_{sp}^{f}, P_{sp}^{f}, valid$

  end

  else

    Calculate LineMagnitude: $lineMagnitude = \sqrt{lineMagnitude^2}$

    Calculate the scissor end cutting point: $P_{end}^{f} = FindLineEndPoint(P_{start}^{f}, lineMagnitude, \vec{v}_{cut}^{f})$ **return** $P_{sp}^{f}, P_{start}^{f}, P_{end}^{f}, valid$

  end

end

Algorithm 11: Line of Cut Prediction

2.3.4 Data Processing

1. Estimate the septum surface plane parameters.
2. Load current scissor parameters.
3. Collect the scissor and septum pose sensor data.
4. Use the GenerateLineOfCut function to generate the line of cut.

3 Software Implementation

3.1 Hardware Dependencies

Our project encompasses a system that includes an Aurora EM Tracker, an EM Pointer, surgical scissors with a mounted EM Sensor, and an EM Sensor attached to a patient for tracking the position of the septum relative to the patient.

3.2 Software Dependencies

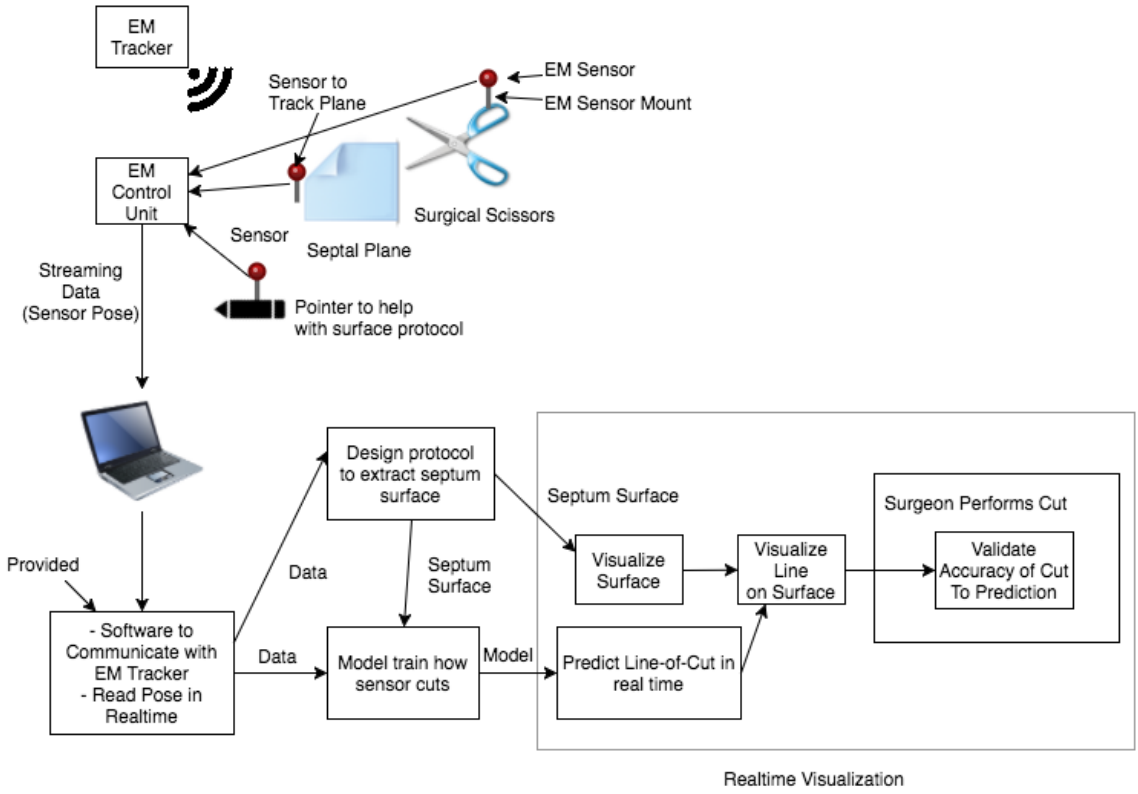
Our project relies on the SeptoServer code, which processes data from the EM Tracker and provides messages to our application with sensor pose data. We did not implement the SeptoServer.

Library dependencies for our C++11 code are QT for the user interface, Point Cloud Library (PCL), Eigen for numerical calculations, boost.

Library dependencies for our Python 2.7 code are VTK and numpy.

3.3 System Design

There are three independent software projects that comprise the system that we developed. There is the SeptoServer, which we did not develop. SeptoServer processes data from the EM Tracker and provides messages to our application with sensor pose data. The format of these packets is outlined in the packet section of this document.



Systems Architecture Diagram.

3.4 Software Threads

There are four threads in the VIOLINS system.

SeptoServer Thread

The SeptoServer thread is a standalone application which produces sensor packets which are sent to our system through a unix pipe.

The SeptoServer generates packets which are specified in the Packet section of this document.

GUI Thread

The GUI thread contains the Feedback Application object which is a child class of QApplication (a QT class). This spawns off the Main Window object which contains the controls

for the Data Collection Modes (in the next section). The Feedback Application also starts a thread which calls the Platform::mainloop function.

Processing Thread

The Platform class routes all data from the SeptoServer to the Septum Surface Generator, Line of Cut Generator for calibration and visualization. Where the data from the SeptoServer, and what data is collected depends on the current mode of the application (specified in the next section).

The format of the packets sent from the Processing Thread to the Visualization thread are specified in the Packet section of this document.

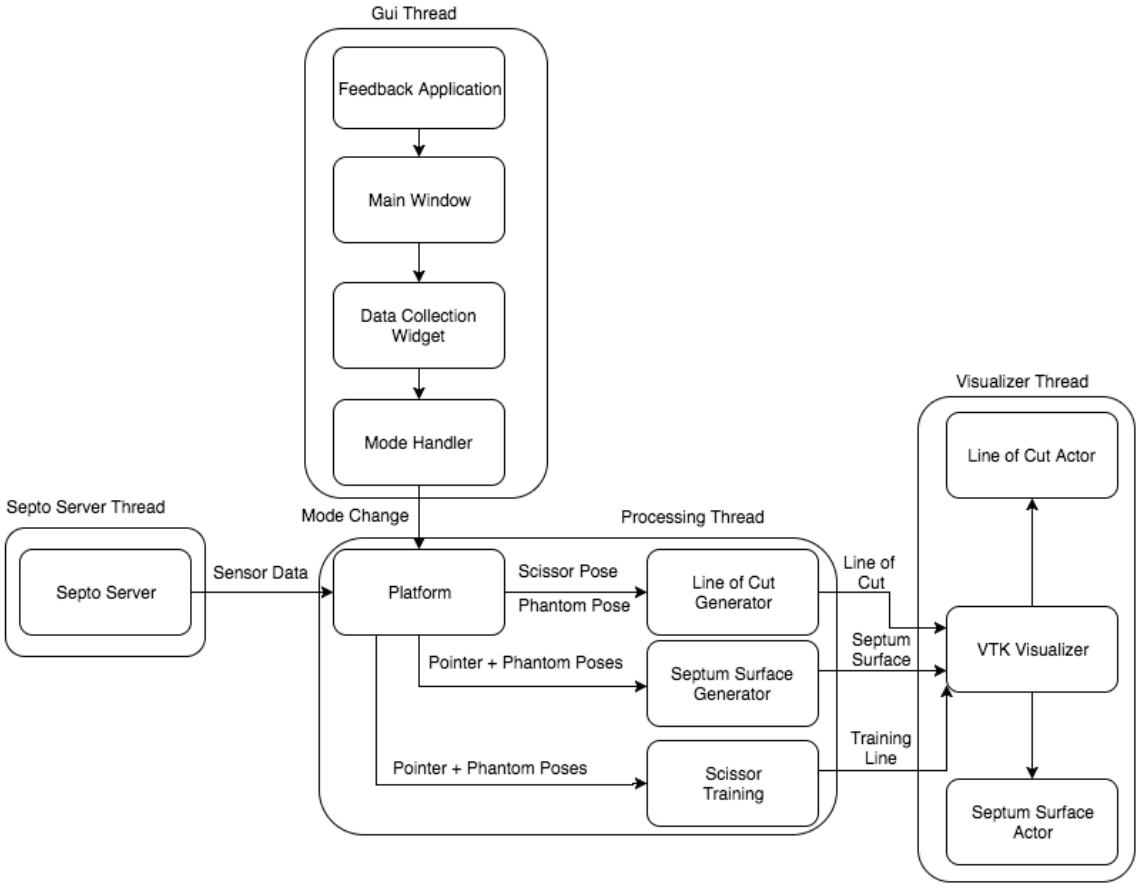
Visualization Thread

The visualization thread receives line of cut and septum surface packets from the Processing Thread and renders the scene in real-time.

The visualization thread runs the "animation.py" script, which uses VTK to render the scene. The visualizer allows the user to zoom in and out and rotate around the center of mass of the septum plane to change the angle of view.

The Visualizer has three timer callbacks running periodically which render the Line of Cut Actor and the Scissor Actor, and read new packets containing line of cut updates, scissor updates, septum surface updates, and training line updates. The formats of these packets are specified in the Packet section of this document.

The visualizer has been configured to drop any packets that are older than .15 seconds old.



Software Modules and Threads.

4 Result

4.1 Septum Plane Generation

4.1.1 Estimated Plane Parameter Validation

It is important to test that the estimated plane parameter is both stable and invariant to the pose of phantom relative to the world before using it in our data pipeline. The following list describes the procedure to validate the plane parameter estimation.

1. An EM sensor was positioned such that it had root mean square error less than 1mm for both the pointer and the phantom sensor.

2. The phantom (cardboard) sensor was fixed to a piece of cardboard by taping it to the cardboard and then placed about 15 cm in front of the Aurora.

3. The phantom was traced with the EM pointer.

   From the collected data, the phantom was approximately on the ZY plane

4. Sensor packet data was saved in the file "calibration_datasets/Aurora-flat-plane.txt"

5. Data was then run through "scripts/parse.py", which converts the pointer point cloud into the coordinate system for the phantom sensor.

6. Use SeptumPlaneGeneration to estimate the plane parameters with distance threshold 0.01 mm.

7. The result of the plane parameter estimation is the plane parameters a, b, c, d, $ax + by + cz = d$: -0.0123241 -0.0209593 0.999704 -4.52493. The units are in millimeters.

8. To validate that the frame transformation between the pointer and the phantom sensor was correct, we held the Phantom sideways, so that the cardboard was approximately in the XY plane in the Aurora coordinate system.

9. Sensor packet data was saved in the file "calibration_datasets/Aurora-sideways-plane.txt".

10. The resulting estimation of the plane parameters a, b, c, d was: 0.00471905 -0.0176695 0.999833 -2.72238.

11. The planes are both in the XY plane in the phantom sensor's coordinate system.

12. The angle between the two plane normals is 0.9949 degrees.

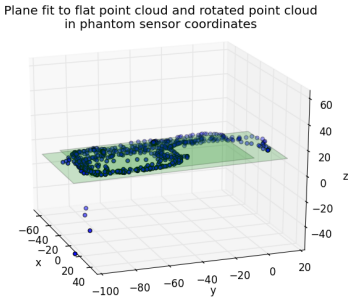


Figure 6: Plane estimation result

Given that the angle between the two plane normals is small, it is verified that the plane parameter estimation is invariant to the phantom pose relative to the world.

4.1.2 Estimated Convex Hull Validation

Since a convex hull is used to approximate the shape of the septum nasal cartilage, it is important to check the performance of the convex hull estimation. This following list describes the validation process of convex hull validation

1. Attach the sensor firmly to the testing target. A cardboard with a picture of nose anatomy is used as the testing target.

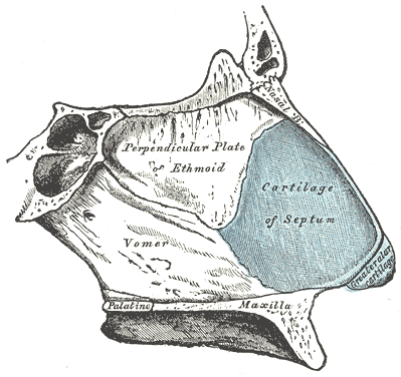


Figure 7: Convex hull testing target
https://upload.wikimedia.org/wikipedia/commons/6/65/Gray854.png

2. Use an EM pointer to trace the illustrated nasal bridge.
3. Use an EM pointer to trace the septum nasal cartilage.
4. Transform all points to be relative to the cardboard.
5. Use SeptumConvexHullEstimation to estimate the convex hull from the combination of traced points from the nasal bridge and septum nasal cartilage.
6. Visualize the raw traced point cloud (relative to the cardboard), the point cloud projected onto the plane estimation, and the result of the convex hull estimation.

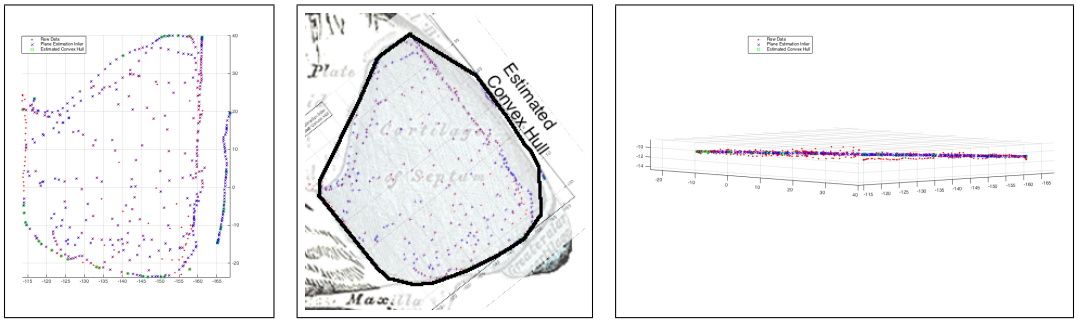


Figure 8: Visualized Convex Hull Result

4.2 Scissor Training

4.2.1 Estimating initial cutting point from point cloud data

To validate our decision to use averaging to estimate initial cutting point in the cardboard, it is important to verify the error distribution in the collected initial cutting point cloud is close to gaussian.
This following list describes the process for validating the decision to use point average for the initial cutting point:

1. Trace the cardboard with a septum normal trace and a nasal bridge trace to get the cardboard normal and a convex hull.
2. Place the pointer on the target initial cutting point and do point a trace.
3. Visualize the resulting point cloud

Every circle in this figure represents an individual point, and x symbol represents the center of the point cloud. Matlab is used for visualizing the point cloud result.

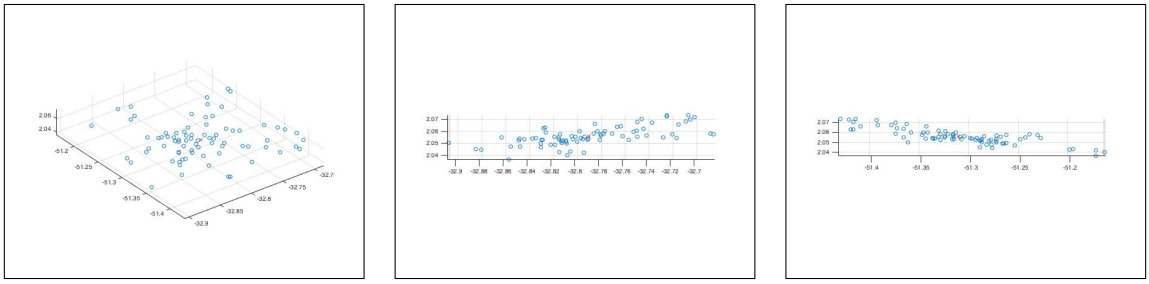


Figure 9: Initial Cutting Point Estimation

From the figure above, we conclude that using averaging is a reasonable decision to approximate the initial cutting point.

4.2.2 Estimating line of cut vectors from point cloud data

To get a good scissor model calibration, it is important for the line of cut vector to be stable with repetition and independent of the cardboard pose relative to the world coordinate system.

The process for validating the line vector from point cloud data is described in the following list:

1. Trace the cardboard with the septum normal trace and the nasal bridge trace to get the cardboard normal and a convex hull.
2. Trace the target line with the pointer tool.
3. Run CalculateVectorFromPointCloud to calculate the vector direction.
4. Visualize the resulting point cloud.
5. Rotate the cardboard and repeat all steps.

The following figure shows the centered point cloud with the line fitting result. Matlab is used for visualizing the point cloud.

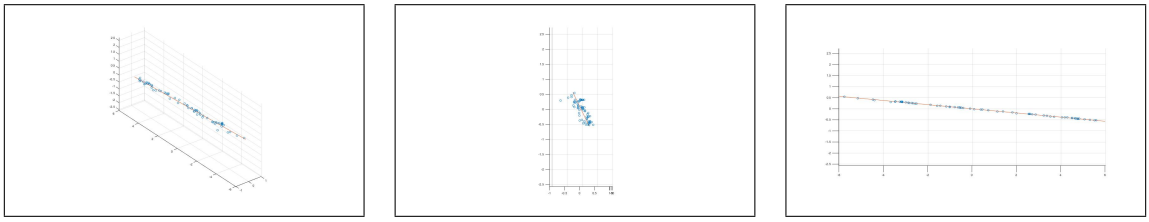


Figure 10: Line Fitting Test

The vector generated from this line fitting is (0.0364642, 0.994949, -0.0935225).

The visualized line seems to match with the actual pointer movement. To prove that the line vector is invariant to the cardboard pose, the figure below shows the visualized line after rotating the cardboard.

The vector generated from this line fitting is (0.0384819, 0.994891, -0.0933362). Given that there is almost no difference in the line fitting result after rotating the cardboard, it is proved that the line vector is relative to the cardboard frame.

4.2.3 Estimating Scissor Pinch Point

One of the ways to validate the accuracy of the scissor pinch point position is by checking the distribution of points in the concatenated point clouds from multiple points.

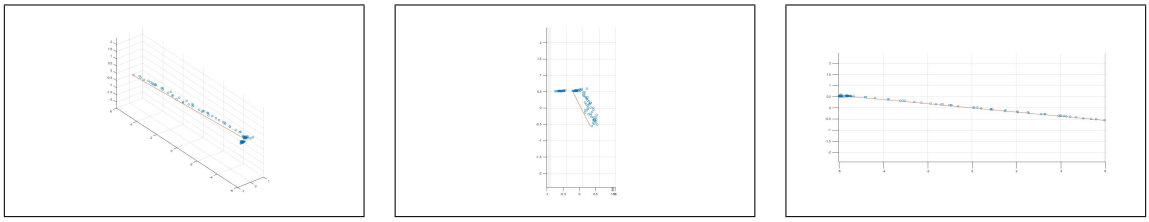


Figure 11: Rotated Cardboard Line Fitting Test

Another way to validate the pinch point is by doing an actual cut based on line of cut prediction, which will be described further in the Line Of Cut documentation.

This following figure is the visualized result of pinch point estimation from the data collected by following the data collection subsection.

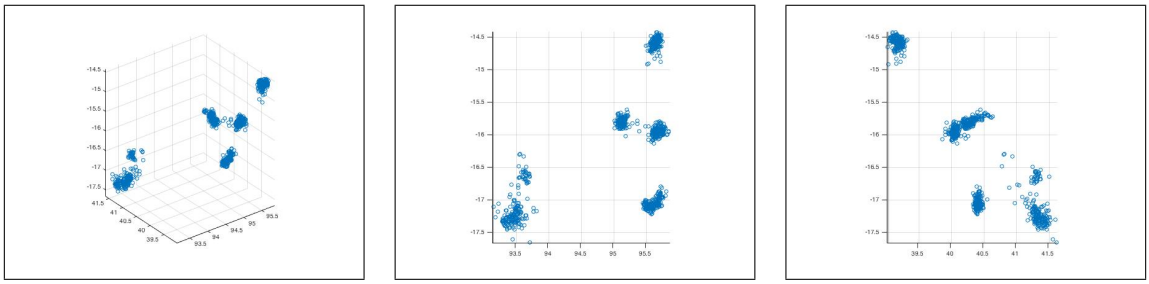


Figure 12: Visualized Pinch Point Training Result

The variation is about 2mm in every axis, which is reasonable considering the sensor error spread is about 1mm, and we use 3 sensors to estimate scissor pinch point position.

4.2.4 Estimating Scissor Plane Normal

To validate the accuracy of the scissor normal, we calculate the angle differences between the estimated line of cut vector direction from pointer tracing and the predicted line of cut vector from cross product of phantom normal and the scissor plane normal. The angular error for each of six training line's is (in degrees): 4.05656 2.89201 4.38761 3.44492 4.60916, with average 3.87805 degrees.

We will also do the actual cut validation in the following section.

4.3 Line of Cut Prediction Evaluation

In this section we evaluate the accuracy of the line of cut.
The procedure to do the evaluation is as follows:

1. Load the current scissor parameters.
2. Collect initial cutting points and line directions (see scissor training documentation for more details)
3. Run the Tool Cut Validation Mode
4. Align the predicted line of cut (red line) as close as possible to the training line (purple line) on the visualization.
5. Cut the phantom.

This following figure represents the cardboard before the evaluation process:

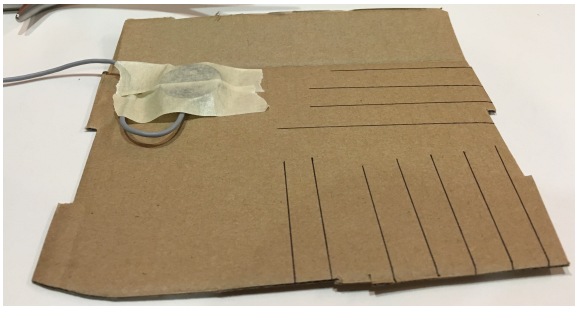


Figure 13: Original Cardboard

This following figure represents the cardboard after the evaluation process, where we performed cuts on each line:

The result of the evaluation is represented below:

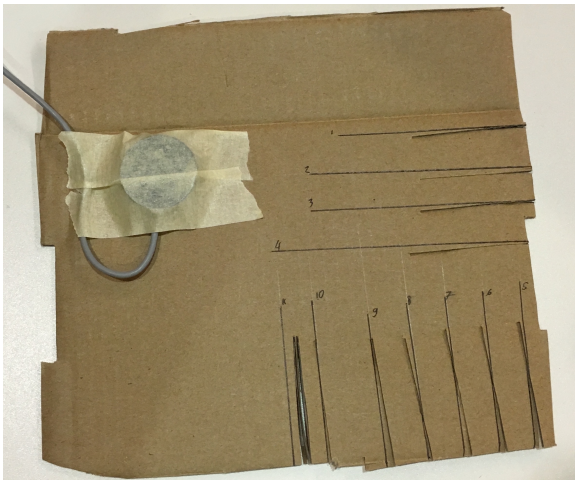


Figure 14: Line of Cut Evaluation Result

Line Num	Translation Error(mm)	Angular Error(degrees)
1	< 0.5	2.4
2	1.78	1.4
3	< 0.5	2.6
4	0.93	3.7
5	1.75	4.5
6	1.61	4.3
7	1.97	4.5
8	1.21	4.3
9	1.90	2.9
10	1.02	2.3
11	2.35	0.8

The translation error is measured from the distance between the actual initial cut point and the target initial cut point. The angular error is measured from the angle differences between the actual cut vector and the target cut vector. The following graphs are representation the translation and angular error for each line:

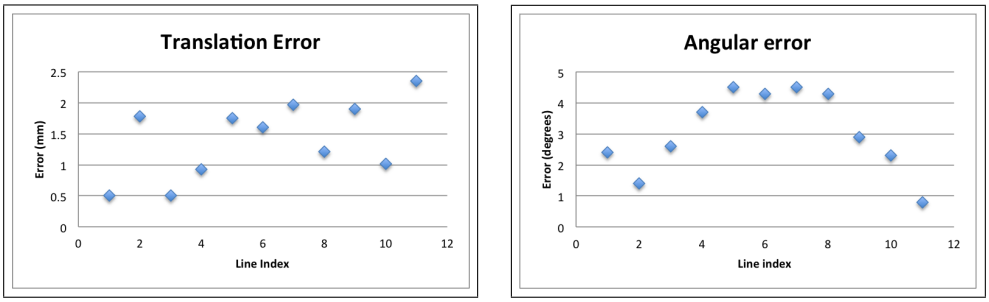


Figure 15: Translation and Angular Error Chart

5 User's Guide

How to Run

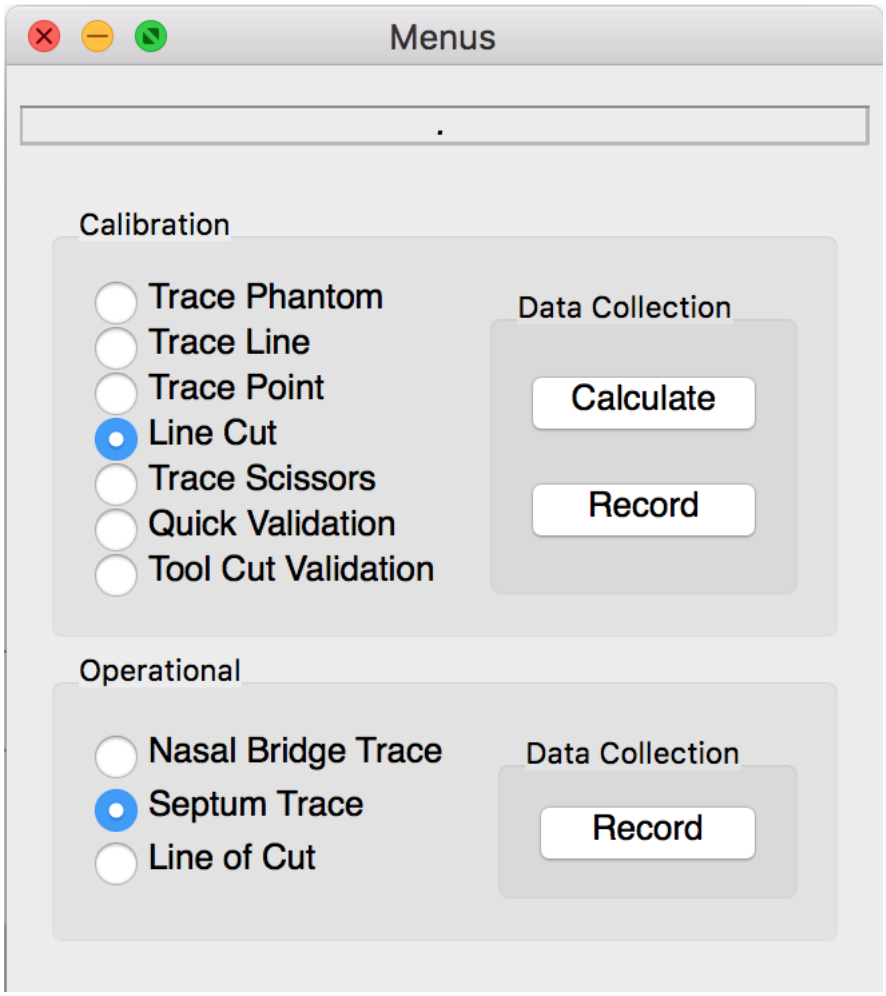
We created a convenience script "run.bash" which starts the SeptoServer and starts our application.

There's a CMake file in the root directory which compiles our C++ code and produces an executable. The software dependencies are Qt 4.8.7_1, boost 1.5.8, PCL 1.7.2, Eigen 3.2.6.

The python script to run the visualizer (which uses VTK 6.3.0 and NumPy 1.9.2) is in the vis folder.

Overview of Data Collection Modes

The workflow for our application is divided into two different categories: Calibration and Operation. Calibration tasks are used to calculate our scissor model parameters or validate our algorithm, and operation tasks are tracing the septum and visualizing the line-of-cut.



The ModeHandler object is a bridge between the GUI and the Platform object that allows us to change the current mode.

5.1 Calibration Modes

Trace Phantom

Trace Phantom mode takes data from the patient's forehead sensor and EM pointer. When the Record button is pressed it will collect the points traced on the phantom. When the Record button is unpressed, the data will be used to fit a plane to the phantom surface. This plane will be used to create the scissor model.

Trace Line

For Trace Line mode, straight lines should be drawn on the phantom. These will be traced with the EM pointer from the edge of the phantom to the end of the line.

Trace Line mode will save a single line traced on the phantom surface. To trace a line, press the Record button, then move the pointer along the line. This mode creates a point cloud, so it's okay to trace over the line multiple times, although once should be fine. If the phantom that is being used doesn't have a ridge for the line, then a flat surface like a credit card should be used to ensure that the line is traced as straight as possible.

For each line that is traced, the Record button needs to be pressed while tracing, then unpressed when finished. Each Traced Line will correspond to a Traced Point, and they both need to be completed in the same order.

Trace Point

For Trace Point mode, the start of the line from the Trace Line will be touched with the EM pointer. This is the same location that the scissors will be placed for the Line Cut calibration mode.

Trace Point mode will save a single point from the EM pointer. The point should be the start of the traced line. This is where we will place the scissors for the Line Cut calibration mode. It will average the point cloud collected from the EM pointer while the Record button is pressed.

For each point that is collected, the Record button needs to be pressed while tracing, then unpressed when finished. Each Traced Point will correspond to a Traced Line, and they both need to be completed in the same order.

Line Cut

For Line Cut mode, the pinch point of the scissors will be placed at the same point from the Trace Point mode. This is the start of a Traced Line. The scissor blades should be placed on top of the Traced Line.

Line Cut mode will collect a point cloud of the scissor sensor's pose when the Record button is pressed. When the Record button is unpressed, the point cloud will be averaged, and the center of the point cloud will be treated as the pinch point.

Trace Scissors

In Trace Scissors mode, the EM pointer will be traced along the surface of one of the scissor blades. When the Record button is pressed, the EM pointer and scissor sensor data will be cached, and when Record is unpressed a plane is fit to the blade of the scissor.

Quick Validation

Quick Validation prints out the distance between the calculated pinch point of the scissors and the EM pointer in the scissor coordinate system. It also prints out the angle between the scissor blade plane and the phantom plane in the scissor coordinate frame.

The pointer should be placed on the scissor pinch point to see the accuracy of the prediction of the scissor pinch point in millimeters.

The scissors should be placed on top of the phantom to see the angle between the scissor blade plane and the phantom plane.

Tool Cut Validation

This runs the line of cut visualization (it's similar to the Line of Cut Operational mode) and it also prints out additional debug data like the distance between the initial cutting point to the trace point and the angle between a training line and the estimated line of cut.

Pressing Record will toggle the training line index. If you are currently validating the accuracy of line 0 and want to see the debug information for line 1, then toggle the Record button.

5.2 Operational Modes

Nasal Bridge Trace

When the Record button is pressed, this mode takes the EM pointer and patient's forehead EM sensor data and stores it until the Record button is unpressed. When the recording ends, the SeptumSurfaceGenerator will store the nasal bridge points. They are used to constrain the convex hull of the septum surface and make a boundary with the outside of the nose.

If the Nasal Bridge points and Septum points have both been generated when the Record button is unpressed, then the Septum Surface Generator will calculate the Septum

Surface and send the septum surface packet to the visualization module.

Septum Trace

When the Record button is pressed, this mode takes the EM pointer and patient's forehead EM sensor data and stores it until the Record button is unpressed. When the recording ends, the SeptumSurfaceGenerator will store the septum points.

If the Nasal Bridge points and Septum points have both been generated when the Record button is unpressed, then the Septum Surface Generator will calculate the Septum Surface and send the septum surface packet to the visualization module.

Line of Cut

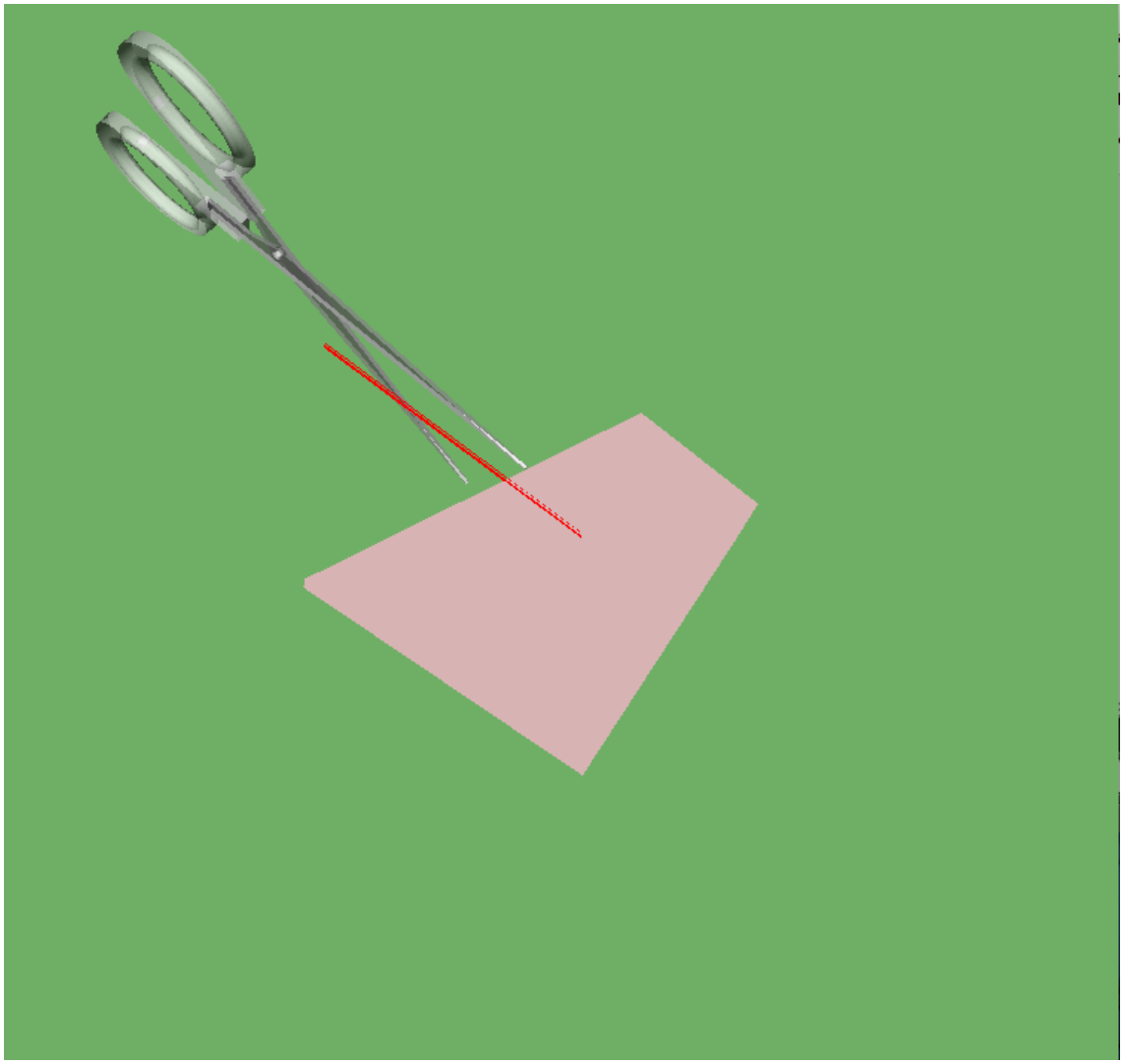
Line of Cut mode is a streaming mode. When the Record button is pressed, it will continuously calculate a line-of-cut for each incoming packet of sensor data, and will send them to the visualization module.

When the Record button is pressed, the Line of Cut packet is sent to the visualization module. A valid scissor model needs to be loaded and the septum surface needs to have been drawn for the line-of-cut to be calculated.

This mode requires the patient's forehead EM sensor and the scissor EM sensor to both be in view of the device for it to make a single line-of-cut calculation.

5.3 Line of Cut Visualization

When the application begins, the VIOLINS window will contain a pair of surgical scissors. When the septum surface is generated from the Septum Trace and Nasal Bridge Trace modes, it will be added to the model. When Line of Cut mode is enabled and a scissor model has been loaded and the septum surface has been generated (and thus is visible in the VIOLINS window), Line of Cut packets will be sent to the visualization which will update the scissor position and render a line of cut.



5.4 Packet Formats

This document describes the packet formats for the software modules that we have developed. These packets are used for communicating data to the Python visualization module from Platform.cpp

SeptoServer Packet

SeptoServer (code that was provided to us by Narges) transmits packets of data to the code that we have written. Originally, SeptoServer wrote these packets into a file. We added an http server and packet queue in SeptoServer that serves the most recent packets to The data values in the packets are defined below:

```
toolname: string
framenum: string
timestamp: string of a long, of a unix timestamp, seconds since Unix epoch
portHandleStatus: string of the status,\\ portHandleStatus be OK if the tool is
x: string of the x component translation of the transformation
y: string of the y component translation of the transformation
z: string of the z component translation of the transformation
r00: string of double, element 00 of the rotation matrix
r01: string of double, element 01 of the rotation matrix
r02: string of double, element 02 of the rotation matrix
r10: string of double, element 10 of the rotation matrix
r11: string of double, element 11 of the rotation matrix
r12: string of double, element 12 of the rotation matrix
r20: string of double, element 20 of the rotation matrix
r21: string of double, element 21 of the rotation matrix
r22: string of double, element 22 of the rotation matrix
stat: Root Mean Square error of the reading
```

Table 1. SeptoServer packet data description.

Each packet is a single ascii-encoded string. Each value in the packet is separated by a tab character. If there is one sensor, then these 17 values will appear in the order listed above, separated by tabs, and will terminate with a newline character. If there are n sensors, then these 17 values will appear in the order listed above, with n*17 values in the packet.

If there are 3 sensors (which our project requires), then each packet will contain 51 data elements, each separated by tabs, and will terminate with a newline character. Entries $1 - 17$ come from sensor 1, $18 - 34$ come from packet 2, etc.

SeptoServer Packet Reception

The code written in Platform.cpp processes the packets from the SeptoServer. We require that sensor 1 is the sensor placed on the patient's forehead, sensor 2 is the sensor attached

to the surgical scissors, and sensor 3 is the pointer.

Platform.cpp processes the packets in three modes.

5.5 Input Modes

5.5.1 pipe

When the pipe flag is passed to the executable created from Platform.cpp, packets are read from standard input.

5.5.2 http

When the http flag is passed to the executable, packets are read from a tcp client.

5.5.3 file

When neither of these flags are specified, the command line argument is assumed to be the name of a file of a recorded session which is read line-by-line.

5.6 Processing Modes

There are two algorithms which are run on the EM sensor pose data. The Processing Mode determines where the data packets from the SeptoServer are sent by the Platform object.

A GUI will allow the user to choose the current mode. One button will say "Septum Tracing Mode," and the other button will say "Line of Cut Mode." Only one mode can be selected at a time. The selected mode should have visual indication that it has been selected.

Septum Tracing Mode

This mode will generate the Septum Surface Model, which is used in the Line of Cut Mode. The Line of Cut Mode cannot run until the Septum Surface Model has been created.

During the Septum Tracing Mode, the packet data is turned into transformComponent objects. The packet from sensor 1 (the patient's forehead sensor) and the packet from sensor 3 (the pointer) are placed into containers. The packet from sensor 2 is dropped.

When Septum Tracing Mode ends, a Surface model will be generated and sent to the visualization module using the Septum Surface Generation algorithm defined in the Septum Tracing Procedure.

Septum Tracing Mode produces packets described in the next section.

Line of Cut Mode

When the Septum Tracing Mode has generated a Septum Surface Model and transmitted it to the visualization module, the Line of Cut Mode can begin.

Line of Cut Mode will take data from sensor 1 (the patient's forehead sensor) and sensor 2 (the scissor sensor) and perform the Line of Cut Prediction algorithm. The data from sensor 1 and sensor 2 in each packet will be fed to the line of cut prediction algorithm and will be written to standard output. The visualization module's standard input is piped from the output of the Platform module, so the visualization module is able to receive the Line of Cut Packets.

Packets for Visualization

When the Septum Tracing Mode has completed, a Septum Surface packet needs to be sent to the visualization module. These packets are transmitted by printing to standard output. The visualization module is connected via a unix pipe to the standard output of the Platform class that controls the line of cut creation and septum surface creation.

When the Line of Cut Mode is active, a Line of Cut packet is sent to the visualization module.

We define a standard packet form in the first section, then outline the Septum Surface packets and Line of Cut packet formats.

Standard Packet Form

Each data packet will be an ascii-encoded string with fields separated by tabs, and each packet will terminate in newline characters. Multiple packets can be joined together through string concatenation provided each packet ends in a newline character.

Each packet begins with an integer specifying the packet type. The packet types are:

Integer Prefix	Packet Type
1	Septum Plane
2	Septum Surface Reset Packet
3	Septum Surface Points
4	Septum Surface Vertex Points
5	Line of Cut
6	Septum Convex Hull
7	Training Line Packet
8	Remove Training Line
9	Plane of Cut Packet
A	Training Plane Packet

Septum Plane Packet

The first iteration of our project assumes the Nasal Septum is a mathematical plane. The Septum Plane packet will contain three points, p1, p2, p3. The format of this packet is:

packet prefix	x1	y1	z1	x2	y2	z2	x3	y3	z3
1	x1	y1	z1	x2	y2	z2	x3	y3	z3

Septum Plane Packet

Septum Surface Reset Packet

When the Septum Tracing Mode ends, the Platform will send the visualization module a Septum Surface Reset packet. This packet signifies that the old Septum Surface Model should be deleted.

This packet is the literal string:

2\tERASE\n

Literal String of Septum Surface Reset Packet

Note: This packet is not implemented. Any old septum surface data is removed when new septum surface data is received.

packet prefix	message
2	ERASE

Tabular form of Septum Surface Reset Packet

The only valid message is ERASE (which deletes the old septum surface model). Any other message is currently undefined.

In the future, this system could be extended to support a mode which stores old models in the visualization module and allow the user to toggle between surface models.

Septum Surface Point Packet

The Septum Surface Point packet will contain points on the surface of the septum mesh. This is a packet that begins with the integer 3, and contains n points collected by the EM pointer.

The packet will contain the integer 3, and interleaved points (x1, y1, z1, x2, y2, z2, ..., xn, yn, zn), which are string representations of double values. All values are separated by tabs, and the whole packet ends in a newline character.

3\tx1\ty1\tz1\tx2\ty2\tz2\t...xn\tyn\tzn\n

Septum Surface Vertex Points Packet

A Vertex Point Packet contains triplets of the points that constitute a single triangle in the Septum Surface mesh. The format of the packet contains the packet prefix, followed by tab-spaced triplets of string representations of integers.

All values in the packet are separated by tabs, and terminate in a newline character.

packet prefix	int	int	int	...	int	int	int	newline
4	p1	p2	p3	...	p_{n-2}	p_{n-1}	p_n	newline

Line of Cut Packet

The Line of Cut Packet contains the packet prefix 5, the scissor pose, the start of the line of cut, the end of the line of cut, a timestamp, and a flag for whether the line of cut is valid (whether it intersects the septum plane). All values are tab spaced, and the packet ends in a newline.

The scissor rotation is given in three Euler angles, it's the scissor rotation relative to the septum plane, the scissor translation is three doubles, the line of cut start is a vector with three doubles, the line of cut end is a vector with three doubles, the timestamp is a long since unix epoch, and valid is a flag that is 0 if the line of cut is not in the septum

plane, and 1 otherwise.

The packet starts with the packet prefix, and contains all fields in the order given, they are tab-spaced, and it ends in a newline.

Septum Convex Hull Packet

The Septum Convex Hull packet begins with a 6, and contains a list of 3 or more (x,y,z) points on the convex hull. All values are tab spaced and the packet ends in a newline character.

Training Line Packet

The Training Line Packet begins with a 7, and contains two (x,y,z) triplet. The first (x,y,z) triplet is the start of the line, and the second one is the end of the line. The packet's format is 7, tab, x, tab, y, tab, z, tab, x, tab, y, tab, z, end

Remove Training Line Packet

The Remove Training Line Packet removes the training lines. It begins with the packet header 8, and ends with a newline character.

Plane of Cut Packet

The plane of cut packet begins with the packet header 9, the scissor pose, and contains four triplets of x, y, z points in the septum plane coordinate system, given in clockwise order. The packet ends in a newline
, a timestamp, and a flag for whether the line of cut is valid (whether it intersects the septum plane). All values are tab spaced, and the packet ends in a newline.

The scissor rotation is given in three Euler angles, it's the scissor rotation relative to the septum plane, the scissor translation is three doubles, the line of cut start is a vector with three doubles, the line of cut end is a vector with three doubles, the timestamp is a long since unix epoch, and valid is a flag that is 0 if the line of cut is not in the septum plane, and 1 otherwise.

The packet starts with the packet prefix, and contains all fields in the order given, they are tab-spaced, and it ends in a newline.

Training Plane Packet

The training plane packet begins with the packet header A, and contains four triplets of x, y, z points in the septum plane coordinate system, given in clockwise order. The packet ends in a newline

Properties XML format

properties.xml contains deployment-specific properties for the VIOLINS system. Sensor name and serial numbers, pivot calibration data, the location of the scissor model file, and the location of the training data file should all be specified in properties.xml

```
<properties>
 <sensors>
  <sensor>
   <name>scissor</name>
   <serial>38CC9C00</serial>
  </sensor>
  <sensor>
   <name>phantom</name>
   <serial>38DD7400</serial>
  </sensor>
  <sensor>
   <name>pointer</name>
   <serial>385D8400</serial>
   <x_offset>0.07087782770395279</x_offset>
   <y_offset>-0.12134015560150146</y_offset>
   <z_offset>0.7610346674919128</z_offset>
  </sensor>
 </sensors>
 <scissormodelfile>scissormodel.xml</scissormodelfile>
 <trainingdatafile>trainingdata.xml</trainingdatafile>
</properties>
```

Example properties.xml file.

Scissor Model XML format

scissormodel.xml contains the blade length, scissormodelid, number of blade points, the normal of the scissor blades, and the translation from the sensor to the scissor pinch point. An example scissor model file is below:

```
<scissormodel>
  <bladelength>40</bladelength>
  <scissormodelid>40</scissormodelid>
  <nbladepoints>1</nbladepoints>
  <normalvector>
    <x>0.32372914039218242</x>
    <y>-0.0014404408107645014</y>
    <z>0.94614870331846324</z>
  </normalvector>
  <sensortranslationtopinchpoint>
    <x>97.485531132318968</x>
    <y>45.832089081243019</y>
    <z>-16.19994109359023</z>
  </sensortranslationtopinchpoint>
</scissormodel>
```

Example scissormodel.xml

Training Data XML format

When lines have been traced on the phantom, it may be useful to save this file for viewing later. trainingdata.xml allows traced lines and points to be saved, as well as the normal of the phantom plane.

```
<scissorTrainingData>
  <numberOfLineTrace>2</numberOfLineTrace>
  <numberOfPointTrace>2</numberOfPointTrace>
  <lineTraces>
    <lineTrace>
      <x>1.5</x>
      <y>1.2</y>
      <z>1.1</z>
    </lineTrace>
    <lineTrace>
      <x>-1.5</x>
      <y>1.6</y>
```

```
      <z>1.3</z>
    </lineTrace>
  </lineTraces>
  <pointTraces>
    <pointTrace>
      <x>0.1</x>
      <y>0.1</y>
      <z>0.1</z>
    </pointTrace>
    <pointTrace>
      <x>1.1</x>
      <y>0.3</y>
      <z>0.2</z>
    </pointTrace>
  </pointTraces>
  <planeNormal>
    <x>0</x>
    <y>0</y>
    <z>0</z>
  </planeNormal>
</scissorTrainingData>
```

Example trainingdata.xml

6 Management

We ended up meeting together to develop at least twice a week and we also met at least twice a week with our mentor on Mondays and Thursdays.

6.1 Deliverables

======Deliverables======

6.1.1 Minimum Deliverables - All Completed

1. Training procedure for any model of scissors. Completed.
2. Line of Cut prediction for scissors. Completed.
3. Visualizing line of cut prediction and septum surface / phantom. Completed.

4. Septum surface reconstruction by tracing the actual septum. Completed.

5. Documentation for all software and mechanical designs (see pseudocode for software design, and for implementation see the software documentation document with 50 pages of software documentation). Completed.

6.1.2 Expected Deliverables - Partially Completed

1. Realtime visualization of line of cut prediction on septum surface (¿ 15 Hz refresh rate). Completed

2. Validation and accuracy measurement for all components. Completed

3. Mesh reconstruction from point cloud data. Not Completed.

   Pseudocode was developed, including PCL calls that need to be made with data that exists in our pipeline

   Packet format for transmitting data to the visualization module was developed

   Proof of concept was developed to make a mesh from a point cloud but we did not have enough time to integrate it into our pipeline or validate the results.

6.1.3 Maximum Deliverables - Partially Completed

1. Improvements in accuracy. Completed

   applying pivot calibration to pointer tool

   doing calibration of scissor model in a low-noise environment

   redoing scissor calibration multiple times to ensure consistent results.

2. Septum surface reconstruction by randomized septum surface touching. Not completed.

3. Image projection of anatomy onto surface mesh. Not completed.

   Our mentor specified that this is not important.

6.2 Discussion

We did not complete all of our deliverables in order, and we made partial progress on multiple deliverables at the same time, which made it hard to track our progress. For future projects, it would be good to have broken up the individual deliverables into smaller subtasks so we can track the progress of them.

We plan on continuing working on this project, since we have been discussing studying how this tool can improve surgeon's skills through providing appropriate understanding of

the underlying anatomy.

The next deliverable that we would work on after completing this course is calculating a septum mesh. It's not techincally difficult, and we wrote the pseudocode for the algorithm that includes the PCL function that we need to call. We need to make a decision for how to exactly compute the mesh-to-scissor-plane intersection. One suggestion is to assume the septum surface is somewhat planar, keep the existing plane estimation algorithms, render a surface mesh, and let the visualization library handle the rendering of the intersection between the plane-of-cut and the septum mesh. Another possible implementation would have the line of cut generator project the line of cut onto the septum mesh.

One of our maximum expected deliverables, the septum surface reconstruction by randomized septum surface touching, was a stretch goal for us, and it is a bigger task than we had initially anticipated. We would need to try multiple different approaches for this algorithm and validate them, which may take up to a month to perfect.

Another one of the maximum expected deliverables, the image projection of the anatomy onto the surface projection is something that our mentor mentioned as a "nice to have," but isn't crucial for this system. When we showed our project to a surgeon, Dr. Ishii, this wasn't something that he thought was necessary for this system to be useable. It's important as engineers that we can separate what the "nice to have" deliverables are, and what is actually needed to have a working prototype, and to mainly focus on the mission critical aspects of a system.

7 Future Work

One interesting application of this project that emerged from discussion with a surgeon is that this visual system can be used to show a surgeon what cuts should be made next. The first cut in septoplasty should be about 1 cm from the nasal bridge, so that cut could be overlayed to the surgeon to ensure that the first cut is along that line.

We are in the process of setting up a study to train sugeon skill while using this tool. An interesting study that could be done would be to measure how surgeon's skill improves while using this tool, and comparing it to surgeons who do not use this tool. The data collected from the sensor on the scissors is valuable in itself since it provides motion data that can be used to assess surgeon skill using papers like [4].

Another application that could come from [4] is using the motion data from the Aurora sensor to do gesture classification, which could be fed into the visualization to provide bet-

ter visual feedback, or could be fed into prediction modules like our line of cut generator. The line of cut could be predicted farther into the future than a single frame by making assumptions on what gesture the user will perform next, based on a past history.

Another extension of this project is to implement the mesh algorithms that we discussed in the septum surface generation section. This consists of performing a line-to-mesh intersection and projecting the line of cut on the septum mesh. We do intend to implement this soon, since it makes the system more visually accurate. It's especially important since deviated septums are not perfect planes.

The nasal bridge could be visualized with a different color from the rest of the septum. This would give a user a quick way to identify the structure while viewing the visualization. Another alternative option for improvements to the visualization would be using deformable registration to project the septum onto a model of the human nose, which would provide a view of the nose that would be closer to reality than seeing just a slice of the anatomy.

One of our maximum deliverables was implementing the septum surface reconstruction by randomized surface touching. Our Septum tracing algorithms use RANSAC, and a filter could be developed and applied to the points to the septum surface generator which could generate a mesh model of the nose while not requiring the surgeon to drag the pointer along the surface, but to merely touch various points on the surface.

References

[1] Neil Bhattacharyya MD *Ambulatory sinus and nasal surgery in the United States: Demographics and perioperative outcomes* The Laryngoscope Volume 120, Issue 3, March 2010 pp. 635?638,

[2] Bayiz, nal; Dursun, Engin; ?slam, Ahmet; Korkmaz, Hakan; Arslan, Necmi; Ceylan, Kr?at; Samim, Erdal, *Is Septoplasty Alone Adequate for the Treatment of Chronic Rhinosinusitis with Septal Deviation?*, American Journal of Rhinology, Volume 19, Number 6, November-December 2005, pp. 612-616(5)

[3] Russell, M.D. and Kangelaris, G.T., *Comparison of L-strut preservation in endonasal and endoscopic septoplasty: a cadaveric study*, International forum of allergy and rhinology, Volume 4, No. 2, February 2014, pp. 147-150

[4] Lingling Tao, Ehsan Elhamifar, Sanjeev Khudanpur, Gregory D. Hager, Ren Vidal *Sparse Hidden Markov Models for Surgical Gesture Classification and Skill Evaluation*, Third International Conference, IPCAI 2012, Pisa, Italy, June 27, 2012., Proceedings

Information Processing in Computer-Assisted Interventions, Volume 7330 of the series Lecture Notes in Computer Science pp 167-177, 2011.