COMPUTER INTEGRATED SURGERY I
Programming Assignment 2 Report

Iou-Sheng (Danny) Chang
ichang9@jhu.edu

Ching-Yang (Austin) Huang
chuan120@jhu.edu

Prerequisites

In this programming assignment, we build upon the concepts introduced in Programming Assignment 1 and extend our focus to the calibration and correction of distortions within the stereotactic navigation system. The main objective is to obtain the tip location with respect to the CT frame $\vec{V}_{tip}^{CT}$, as depicted in Fig. 1 as $\vec{V}_i$.

Please refer to our submission for Programming Assignment 1 for a comprehensive explanation of the formulation and mathematical approaches, and algorithmic approaches related to the following tasks: *"Cartesian 3D Points, Rotations, and Frame Transformations"*, *"3D Point Cloud Registration"*, *"Pivot Calibration"*, *"Determination of* $\vec{C}_i^{(expected)}$*"*, *"EM Tracking-Based Pivot Calibration and Dimple Position Determination* $\vec{p}_{EM}^{(dimple)}$*"*, and *"Optical Tracking-Based Pivot Calibration and Dimple Position Determination* $\vec{p}_{OT}^{(dimple)}$*"*, which are also utilized in this programming assignment.

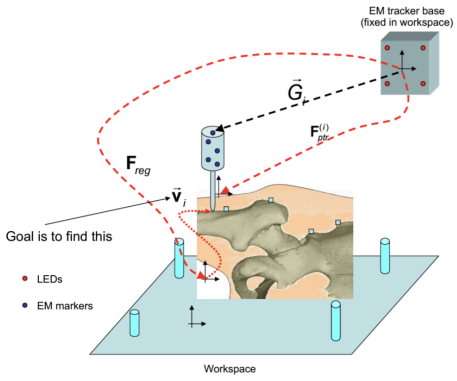

Figure 1. The stereotactic navigation system: finding $\vec{V}_{tip}^{CT}$. Figure is obtained and adapted from [12].

1 Formulation and Mathematical Approach

This section describes the formulation and mathematical approaches employed to complete the assignment.

1.1 Interpolation

The formulation approaches in this subsection closely adhere to the methods detailed in the lecture note titled "Interpolation Review", [10], and the document "Formulation and Python Implementation of Bézier and B-Spline Geometry" authored by Chad B. Hovey [2].

We will begin by delving into 1D interpolation, Bézier curves, and Bernstein polynomials, building up gradually through 2D interpolation and Bézier surfaces, and ultimately into the realm of 3D interpolation and Bézier volumes. These concepts will be the foundation for tackling a main task of our assignment - the 3D calibration and correction of distortion, as outlined in [12].

1.1.1 1D Interpolation, Bézier Curves, and Bernstein Polynomials

The Bézier curve, denoted as $\mathbb{C}$, is a parametric curve defined by control points, referred to as $\boldsymbol{P}_i$. Within the realm of our problem scenario in 3D Cartesian space, each control point $\boldsymbol{P}_i$ has three coordinates, *viz.* $\boldsymbol{P}_i(x, y, z) \in \mathbb{R}^3 \; \forall \; i \in 0 \cdots n$.

The parameter used for describing the curve is typically denoted as t, which can be thought of as a *pseudo-time* that flows within the parameterization from the beginning to the end of the t bounds.

In Bézier geometry, the parameter space t for curves is a real number between zero and unity, inclusive. I.e.,:

$$t \in \mathbb{R} \subset [0.0, 1.0] \tag{1}$$

This range is suitable because the Bernstein polynomial is designed to work effectively in the domain $0 \leq t \leq 1$.

The general form of a Bézier curve $\mathbb{C}^n(t)$, having degree n and $n+1$ control points, is given by:

$$\mathbb{C}^n(t) \triangleq \sum_{i=0}^{n} B_i^n(t) \boldsymbol{P}_i \tag{2}$$

where $B_i^n(t)$ is a Bernstein polynomial, defined as:

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i} \tag{3}$$

and where $\binom{n}{i}$ is the binomial coefficients:

$$\binom{n}{i} = \frac{n!}{i!(n-i)!} \tag{4}$$

1.1.2 2D Interpolation, and Bézier Surfaces

The Bézier surface, denoted as $\mathbb{S}$, is a parametric surface defined by control point net/grid, referred to as $\mathcal{N}$. The control point net/grid is a collection of control points, $\boldsymbol{P}_{i,j}(x,y,z) \in \mathbb{R}^3 \ \forall \ i \in 0 \cdots n, \ j \in 0 \cdots m$, which are arranged in a non-decreasing sequence within two dimensions. It is worth noting that Bézier surfaces are an extension of Bézier curves, as introduced and given in Sec. 1.1.1 and Eq. (2) respectively.

In Bézier geometry, the parameter space (t,u) for surfaces is a real number between zero and unity, inclusive. I.e.,

$$t, u \in \mathbb{R} \subset [0.0, 1.0] \tag{5}$$

The general form of a Bézier surface $\mathbb{S}^{n,m}(t,u)$ of

> degree n and $n+1$ control points for the t parameter, and
> degree m and $m+1$ control points for the u parameter,

is defined as follows:

$$\mathbb{S}^{n,m}(t,u) \triangleq \sum_{i=0}^{n} \sum_{j=0}^{m} B_i^n(t) B_j^m(u) \boldsymbol{P}_{i,j} \tag{6}$$

where the Bézier basis functions are defined as the outer product of two Bernstein polynomials,

$$B_{i,j}^{n,m}(t,u) \triangleq B_i^n(t) \otimes B_j^m(u) \tag{7}$$

In practice, it is common to have the same number of control points for both the t and u parameters, *viz.* $(n+1) = (m+1)$. In this scenario, Eq. (7) simplifies to:

$$B_{i,j}^{n}(t,u) \triangleq B_i^n(t) \otimes B_j^n(u) \tag{8}$$

1.1.3 3D Interpolation, and Bézier Volumes

The Bézier volume, denoted as $\mathbb{V}$, is a parametric volume defined by control point lattice, referred to as $\mathcal{L}$. The control point lattice is a collection of control points, $\boldsymbol{P}_{i,j,k}(x,y,z) \in \mathbb{R}^3 \ \forall \ i \in 0 \cdots n, \ j \in 0 \cdots m, \ k \in 0 \cdots l$, which are arranged in a non-decreasing sequence within three dimensions. It is worth noting that Bézier volumes derive as a natural dimensional extension of Bézier surfaces, as introduced and given in Sec. 1.1.2 and Eq. (6) respectively.

In Bézier geometry, the parameter space (t,u,v) for volumes is a real number between zero and unity, inclusive. I.e.,

$$t, u, v \in \mathbb{R} \subset [0.0, 1.0] \tag{9}$$

The general form of a Bézier volume $\mathbb{V}^{n,m,l}(t,u,v)$ of

> degree n and $n+1$ control points for the t parameter,
> degree m and $m+1$ control points for the u parameter, and
> degree l and $l+1$ control points for the v parameter,

is defined as follows:

$$\mathbb{V}^{n,m,l}(t,u,v) \triangleq \sum_{i=0}^{n} \sum_{j=0}^{m} \sum_{k=0}^{l} B_i^n(t) B_j^m(u) B_k^l(l) \boldsymbol{P}_{i,j,k} \tag{10}$$

where the Bézier basis functions are defined as the outer product of three Bernstein polynomials,

$$B_{i,j,k}^{n,m,l}(t,u,v) \triangleq B_i^n(t) \otimes B_j^m(u) \otimes B_k^l(v) \tag{11}$$

In practice, it is common to have the same number of control points for both the t,u and v parameters, *viz.* $(n+1) = (m+1) = (l+1)$. In this scenario, Eq. (11) simplifies to:

$$B_{i,j,k}^{n}(t,u,v) \triangleq B_i^n(t) \otimes B_j^n(u) \otimes B_k^n(v) \tag{12}$$

1.1.4 3D Calibration and Correction of Distortion

Suppose that we are provided/have obtained the 3D ground truth data $\vec{p}_i$, the distorted data (e.g. readings from a navigational sensor) $\vec{q}_i$, and the sample data $\vec{q}_s$. Our objective is to compute a distortion correction function for the distorted sensor. We have opted to employ a tensor form interpolation polynomial using 5^{th} degree Bernstein polynomials for the implementation.

As previously discussed in Secs. 1.1.1 to 1.1.3, Bernstein polynomial is designed to work effectively in the domain $0 \leq t \leq 1$. Hence in Bézier geometry, the parameter space t is a real number between zero and unity, inclusive. I.e.,:

$$t \in \mathbb{R} \subset [0.0, 1.0] \tag{13}$$

Consequently, we first establish a bounding box for scaling the distorted $\vec{q}_i$ values. In other words, we determine the lower and upper limits represented as $\vec{q}^{min}$ and $\vec{q}^{max}$, then compute $\vec{u}_s$ as follows:

$$\vec{u}_s = ScaleToBezierParameterSpace\left(\vec{q}_s, \vec{q}^{min}, \vec{q}^{max}\right) \tag{14}$$

where

$$ScaleToBezierParameterSpace(x, x^{min}, x^{max}) = \frac{x - x^{min}}{x^{max} - x^{min}} \tag{15}$$

We then construct the tensor-form interpolation polynomial using a 5^{th} degree Bernstein polynomials as shown in Eq. (16). It is worth noting that the choice of the 5^{th} degree Bernstein polynomials for constructing the tensor-form interpolation polynomial is based on its ability to offer a balance between accuracy and computational efficiency.

$$B^5_{i,j,k}(u_x, u_y, u_z) \triangleq B^5_i(u_x) \otimes B^5_j(u_y) \otimes B^5_k(u_z) \tag{16}$$

Finally, we set up and solve the least squares problem as shown in Eq. (17), to obtain the matrix of $c_{i,j,k}$ values, which represents the $coefficients$ that approximate the distortion correction polynomial utilized in our application.

$$\begin{bmatrix} & \vdots & \\ B_{0,0,0}(\vec{u}_s) & \cdots & B_{5,5,5}(\vec{u}_s) \\ & \vdots & \end{bmatrix} \cdot \begin{bmatrix} c^x_{0,0,0} & c^y_{0,0,0} & c^z_{0,0,0} \\ \vdots & \vdots & \vdots \\ c^x_{5,5,5} & c^y_{5,5,5} & c^z_{5,5,5} \end{bmatrix} \approx \begin{bmatrix} & \vdots & \\ p^x_s & p^y_s & p^z_s \\ & \vdots & \end{bmatrix} \tag{17}$$

1.2 Distortion Calibration and Determination of Tip Location with Respect to the CT Frame

Our goal is to utilize either the provided or acquired data to fit polynomial models that describe distortion. These polynomial models are then applied to correct the distortion in the EM tracker space. This dewarped EM tracker space is subsequently utilized to reiterate the pivot calibration process. With this calibration in place, we can establish the registration to the CT coordinate system. Ultimately, we determine the location of the pointer tip with respect to the CT coordinate system. A detailed illustration of the scenario is shown in Fig. 1.

1.2.1 Determination of $C_{expected}$

We commence the process by determining the values of $\vec{C}_i^{(expected)}$ through the methods and functions that we elaborated and developed in *Programming Assignment 1*'s corresponding section. In brief, we compute the transformation matrices F_D and F_A using the function *Rigid Registration* with the parameters $\vec{d}$, $\vec{D}[i]$, and $\vec{a}$, $\vec{A}[i]$, respectively. Subsequently, $\vec{C}_i^{(expected)}$ is derived by applying the transformation $\vec{C}_i^{(expected)} = F_D^{-1} F_a \cdot \vec{c}_i$ using the function *Point Transformation*.

$$\begin{aligned} F_D &= RigidRegistration(\vec{d}, \vec{D}_i) \\ F_A &= RigidRegistration(\vec{a}, \vec{A}_i) \\ \vec{C}_i^{(expected)} &= F_D^{-1} F_A \cdot \vec{c}_i \end{aligned} \tag{18}$$

After computing the $\vec{C}_i^{(expected)}$ values for each frame, we combine them to obtain $C_{expected}$.

1.2.2 Produce a Suitable Distortion Correction Function

Having computed the ground truth data $C_{expected}$, and provided with the distorted navigational sensor readings C, we proceed to follow the methods discussed in Sec. 1.1.4 to fit a 5^{th} degree Bernstein polynomials, as shown in Eq. (16), to serve as our distortion correction function named *Bézier Polynomial Fit*.

$$coefficients \leftarrow BezierPolynomialFit(C_{expected}, C) \tag{19}$$

1.2.3 EM Tracking-Based Pivot Calibration and Tip Position Determination Using the Distortion Correction Function

We employ the obtained distortion correction function to dewarp and correct the distorted sensor data G following the methods detailed in Sec. 1.1.4. This function is named *Correction of Distortion*.

$$\begin{aligned} U_G &= ScaleToBezierParameterSpace(G, G^{min}, G^{max}) \\ B^5_{G,(i,j,k)} &\triangleq B^5_{G,i}(U_{G,x}) \otimes B^5_{G,j}(U_{G,y}) \otimes B^5_{G,k}(U_{G,z}) \\ G_{corrected} &= B^5_{G,(i,j,k)} \cdot coefficients \end{aligned} \tag{20}$$

Using this corrected sensor data $G_{corrected}$, we proceed to repeat the pivot calibration for the EM probe. This process involves applying the methods and functions that we elaborated and developed in *Programming Assignment 1*'s corresponding section to obtain the desired improved tip position $\vec{p}^{EM}_{tip}$.

$$\begin{aligned} \vec{G}_0 &= \frac{1}{N_G} \sum G_{corrected} \\ g_j &= G_{corrected} - \vec{G}_0 \\ F_G = [R_G, p_G] &= RigidRegistration(g_j, G_{corrected}) \\ [\vec{p}^{EM}_{tip}, \vec{p}^{EM}_{dimple}] &= PivotCalibration(R_G, p_G) \end{aligned} \tag{21}$$

1.2.4 Determination of the Fiducial Points' Locations with Respect to the EM Tracker Frame

Following the same procedure outlined in Eq. (20), we obtain the corrected $G^{fiducials}_{EM,corrected}$. This corrected data is then used in conjunction with the previously computed g_j values in Eq. (21) to determine the transformation matrix F_{EM}. Applying F_{EM} to the improved pivot value $\vec{p}^{EM}_{tip}$, we determine the locations B_j of the fiducials points with respect to the EM tracker base coordinate system as follows.

$$\begin{aligned} F_{EM} &= RigidRegistration(g_j, G^{fiducials}_{EM,corrected}) \\ B_j &= F_{EM} \cdot \vec{p}^{EM}_{tip} \end{aligned} \tag{22}$$

1.2.5 Computation of the Registration F_{reg}

Having computed the locations B_j of the fiducials points with respect to the EM tracker frame, and provided with their coordinates in the CT frame $b^{fiducials}_{CT}$, we can find the registration F_{reg} between them using the methods and functions that we elaborated and developed in Programming Assignment 1's corresponding section.

$$F_{reg} = RigidRegistration(b^{fiducials}_{CT}, B_j) \tag{23}$$

1.2.6 Computation of the Tip Locations with Respect to the CT Frame

Following the same procedure outlined in Eq. (20), we obtain the corrected $G^{NAV}_{EM,corrected}$. This corrected data is then used in conjunction with the previously computed g_j values in Eq. (21) to determine the transformation matrix F_{NAV}. Applying F_{NAV} to the improved pivot value $\vec{p}^{EM}_{tip}$, we obtain the pointer tip coordinates with respect to the EM tracker base V^{EM}_{tip}. Similarly, by applying the previously computed registration F_{reg}, we obtain the desired pointer tip coordinates with respect to the CT frame V^{CT}_{tip}.

$$\begin{aligned} F_{NAV} &= RigidRegistration(g_j, G^{NAV}_{EM,corrected}) \\ V^{EM}_{tip} &= F_{NAV} \cdot \vec{p}^{EM}_{tip} \\ V^{CT}_{tip} &= F^{-1}_{reg} \cdot V^{EM}_{tip} \end{aligned} \tag{24}$$

2 Algorithmic Approach

This section describes the algorithmic approaches employed to complete the assignment.

2.1 Interpolation

The algorithmic approaches in this subsection closely follow the formulation approaches outlined in Sec. 1.1, and adhere to the methods detailed in the lecture note titled *Interpolation Review* [10], and the document *Formulation and Python Implementation of Bézier and B-Spline Geometry* authored by Chad B. Hovey [2].

To construct the tensor-form interpolation polynomial $B^5_{i,j,k}(u_x, u_y, u_z) \triangleq B^5_i(u_x) \otimes B^5_j(u_y) \otimes B^5_k(u_z)$, nested for loops are employed to iteratively compute the individual elements $B_{i,j,k}$, which are then combined to assemble the tensor-form interpolation polynomial B. This matrix B is structured such that each row corresponds to one of the N_{data} sets of normalized 3D navigational sensor readings, and within each row, it stores the $[B_{0,0,0} \dots B5, 5, 5]$ values for that particular set of readings.

This function is aptly named `bezier_volume_basis_function`, reflecting its use in the context of 3D sensor data processing. The pseudocode that encapsulates this approach is presented in Algorithm 1.

Algorithm 1 `bezier_volume_basis_function`

Require: u: normalized sensor readings in Bézier parameter space
  Initialize B
  for $row = 1$ to N_{data} **do**
    $u_x \leftarrow u[row][0]$, $u_y \leftarrow u[row][1]$, $u_z \leftarrow u[row][2]$
    $col \leftarrow 0$
    for $i = 0$ to $polynomial\_degree$ **do**
      $B_x =$ `_berstein_polynomial`(i, u_x)
      for $j = 0$ to $polynomial\_degree$ **do**
        $B_y =$ `_berstein_polynomial`(j, u_y)
        for $k = 0$ to $polynomial\_degree$ **do**
          $B_z =$ `_berstein_polynomial`(k, u_z)
          $B[row][col] \leftarrow B_x B_y B_z$
          $col \leftarrow col + 1$
        end for
      end for
    end for
  end for
Return: B

For the task of polynomial interpolation, our goal is to fit the 5^{th} degree Bernstein polynomials to the provided 3D ground truth data p and the data obtained from navigational sensor readings q. This interpolation method is employed to generate the distortion correction function $coefficients$. The interpolation process begins by finding the minimum and maximum values of the sensor readings, represented as q_{min} and q_{max} respectively. Subsequently, we scale the sensor readings to the Bernstein polynomial parameter space u using `scale_to_bezier_parameter_space`. Following this, the tensor-form interpolation polynomial B is obtained using `bezier_volume_basis_function`. To complete the process, the resulting distortion correction function coefficients are obtained through solving the least-squares problem shown in Eq. (17).

The pseudocode that encapsulates this approach, named

`bezier_polynomial_fit` is presented in Algorithm 2:

Algorithm 2 `bezier_polynomial_fit`

Require: p: ground truth, q: navigational sensor readings
$q_{max} \leftarrow \max(q)$, $q_{min} \leftarrow \min(q)$ ▷ For normalization
$u \leftarrow$ `scale_to_bezier_parameter_space`(q)
$B \leftarrow$ `bezier_volume_basis_function`(u)
$coefficients = SolveLeastSquaresProblem(B, p)$
Return: $coefficients$

With the distortion correction polynomial and the $coefficients$ obtained from `bezier_polynomial_fit`, we can then utilize them to perform calibration and correction for the remaining distored data. The process begins with scaling the distorted sensor readings q to the Bernstein polynomial parameter space u using `scale_to_bezier_parameter_space`. Following this, the tensor-form interpolation polynomial B is obtained using `bezier_volume_basis_function`. B is then processed using the previously obtained distortion correction $coefficients$, resulting in the corrected data $q_{corrected} = B \cdot coefficients$.

The pseudocode that encapsulates this approach, named `correction_of_distortion` is presented in Algorithm 3:

Algorithm 3 `correction_of_distortion`

Require: q: distorted sensor readings
$u \leftarrow$ `scale_to_bezier_parameter_space`(q)
$B \leftarrow$ `bezier_volume_basis_function`(u)
$q_{corrected} \leftarrow B \cdot coefficients$
Return: $q_{corrected}$

2.2 Distortion Calibration and Determination of Tip Location with Respect to the CT Frame

The algorithmic approaches in this subsection closely follow the formulation approaches outlined in Sec. 1.2, and adhere to the programming assignment instructions titled *Programming Assignment 1 and 2* [12].

To begin, we acquire the required data from the provided datasets utilizing the data loader, a process detailed in Sec. 3.3.

We proceed by computing the transformation matrices F_D and F_A using the function `rigid_registration`. These matrices are then employed to calculate $C_{expected}$ with the function `point_transformation`. This process is repeated for each frame of data based on the calibration object coordinates c.

Next, we produce the distortion correction function and its corresponding $coefficients$ by fitting a 5^{th} degree Bernstein polynomials to the computed ground truth data $C_{expected}$ and distorted sensor readings C using the function `bezier_polynomial_fit` as discussed in Sec. 2.1 and Algorithm 2.

Following this, we acquire the distortion-corrected sensor data $G_{corrected}$ by employing the function `correction_of_distortion`. This is utilized to obtain the transformation F_G with the function `rigid_registration`. F_G is then applied to perform pivot calibration for the EM probe using `pivot_calibration` to derive the desired improved tip position $\vec{p}_{tip}^{EM}$.

Similarly, we acquire the distortion-corrected sensor data $G_{EM,corrected}^{fiducials}$ by employing the function `correction_of_distortion`. This is utilized to obtain the transformation F_{EM} with the function `rigid_registration`. The locations B_j of the fiducials points with respect to the EM tracker base coordinate system is then computed using `point_transformation`.

This information, in combination with their provided coordinates in the CT frame $b_{CT}^{fiducials}$, allows us to determine the registration frame F_{reg} using the function `rigid_registration`.

To complete the process, we again acquire the distortion-corrected sensor data $G_{EM,corrected}^{NAV}$ by employing the function `correction_of_distortion`. This is utilized to obtain the transformation F_{NAV} with the function `rigid_registration`. Applying F_{NAV} to $\vec{p}_{tip}^{EM}$ using `point_transformation`, we obtain V_{tip}^{EM}. Similarly, by applying the previously computed F_{reg}^{-1} to V_{tip}^{EM} using `point_transformation`, we obtain V_{tip}^{CT}.

The pseudocode that encapsulates this approach, named `main_task_7` is presented in Algorithm 4:

3 Overview of Program

This section offers a concise overview of the program, covering the high-level code structures, and the descriptions of the functions, including their input arguments (Args), outputs (Returns), and intended purposes (Desc).

3.1 PA2_main.py

`PA2_main.py` serves as the driver file responsible for executing the primary tasks of the assignment, conducting validation procedures for the developed mathematical packages, examining the output results, and generating visual plots. This program is implemented in *Python*, utilizing essential libraries such as *NumPy*, *SciPy*, *Pandas*, *Plotly*, and relying on *Python*'s *pathlib*, *collections* and *random* standard libraries [1, 3, 5, 7, 13]. Fig. 4 shows the high-level

Algorithm 4 `main_task_7`

Require: d, a, c: from `calbody` dataset
Require: D, A, C: from `calreadings` dataset
Require: G_{EM}^{pivot}: from `empivot` dataset
Require: $b_{CT}^{fiducials}$: from `ct-fiducials` dataset
Require: $G_{EM}^{fiducials}$: from `em-fiducials` dataset
Require: G_{EM}^{nav}: from `em-nav` dataset

▷ Determine $C_{expected}$ (Sec. 1.2.1)
for $i = 1$ to $N_{frames}^{calreadings}$ **do**
  $F_D \leftarrow$ `rigid_registration`$(d, D[i])$
  $F_A \leftarrow$ `rigid_registration`$(a, A[i])$
  $C_{expected}[i] =$ `point_transformation`$(c, F_D^{-1} F_A)$
end for

▷ Produce a suitable distortion correction function (Sec. 1.2.2)
$ip \leftarrow Interpolation(deg = 5, dim = 3)$
ip.`bezier_polynomial_fit`$(C_{expected}, C)$

▷ Compute $\vec{p}_{tip}^{EM}$ (Sec. 1.2.3)
$G_{corrected} \leftarrow ip$.`correction_of_distortion`$(G_{EM}^{pivot})$
$\bar{G}_0 \leftarrow Mean(G_{corrected})$
$g_j \leftarrow G_{corrected} - \bar{G}_0$
for $i = 1$ to $N_{frames}^{empivot}$ **do**
  $F_G[i] =$ `rigid_registration`$(g_j[0], G_{corrected}[i])$
end for
$[R_G, p_G] \leftarrow F_G$
$\vec{p}_{tip}^{EM}, \vec{p}_{dimple}^{EM} \leftarrow$ `pivot_calibration`(R_G, p_G)

▷ Determine B_j (Sec. 1.2.4)
$G_{EM,corrected}^{fiducials} \leftarrow ip$.`correction_of_distortion`$(G_{EM}^{fiducials})$
for $i = 1$ to $N_B^{em-fiducials}$ **do**
  $F_{EM}[i] \leftarrow$ `rigid_registration`$(g_j[i], G_{EM,corrected}^{fiducials}[i])$
  $B_j[i] \leftarrow$ `point_transformation`$(\vec{p}_{tip}^{EM}, F_{EM}[i])$
end for

▷ Compute F_{reg} (Sec. 1.2.5)
$F_{reg} \leftarrow$ `rigid_registration`$(b_{CT}^{fiducials}, B_j)$

▷ Obtain V_{tip}^{CT} (Sec. 1.2.6)
$G_{EM,corrected}^{NAV} \leftarrow ip$.`correction_of_distortion`$(G_{EM}^{NAV})$
for $i = 1$ to N_{frames}^{em-nav} **do**
  $F_{NAV}[i] \leftarrow$ `rigid_registration`$(g_j[i], G_{EM,corrected}^{NAV}[i])$
  $V_{tip}^{EM}[i] \leftarrow$ `point_transformation`$(\vec{p}_{tip}^{EM}, F_{NAV}[i])$
end for
$V_{tip}^{CT} \leftarrow$ `point_transformation`$(V_{tip}^{EM}, F_{reg}^{-1})$

Return: V_{tip}^{CT}

program structure, along with descriptions of the functions, including their inputs, outputs, and intended purposes.

The main driver file can be executed via a command-line interface that we designed using the *Click* library [6]. For detailed information about the positional and optional arguments available through the command-line interface, please refer to Fig. 2.

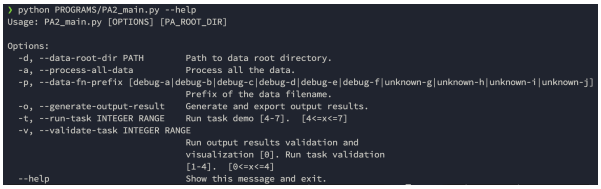

Figure 2. Command-Line Interface for *PA2_main.py*.

3.2 interpolation.py

The mathematical package `interpolation.py` is implemented in *Python*, utilizing *NumPy*, and *Python*'s standard *math* libraries [1, 7]. Fig. 5 shows the high-level program structure, along with descriptions of the functions, including their inputs, outputs, and intended purposes.

3.3 load_data.py

`load_data.py` serves as the data loader responsible for loading and formatting the provided *PA2 Student Data* files. This loader leverages the *Pandas* library for efficient handling of data from the provided *.txt* files, and relies on *Python*'s *pathlib* standard library for efficient filesystem path handling [5, 7]. Fig. 6 shows the high-level program structure, along with descriptions of the functions, including their inputs, outputs, and intended purposes.

The main driver file can be executed via a command-line interface that we designed using the *Click* library [6]. For detailed information about the positional and optional arguments available through the command-line interface, please refer to Fig. 3.

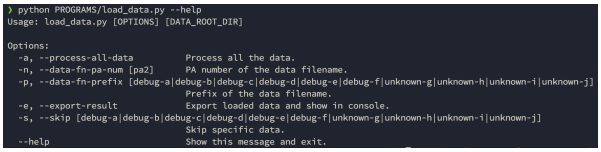

Figure 3. Command-Line Interface for *load_data.py*.

3.4 Other Source Code Files

For an overview of the mathematical packages, *transform.py* and *pivot_calibration.py*, which are also utilized in this programming assignment, please refer to our submission for *Programming Assignment 1* for comprehensive information about these programs.

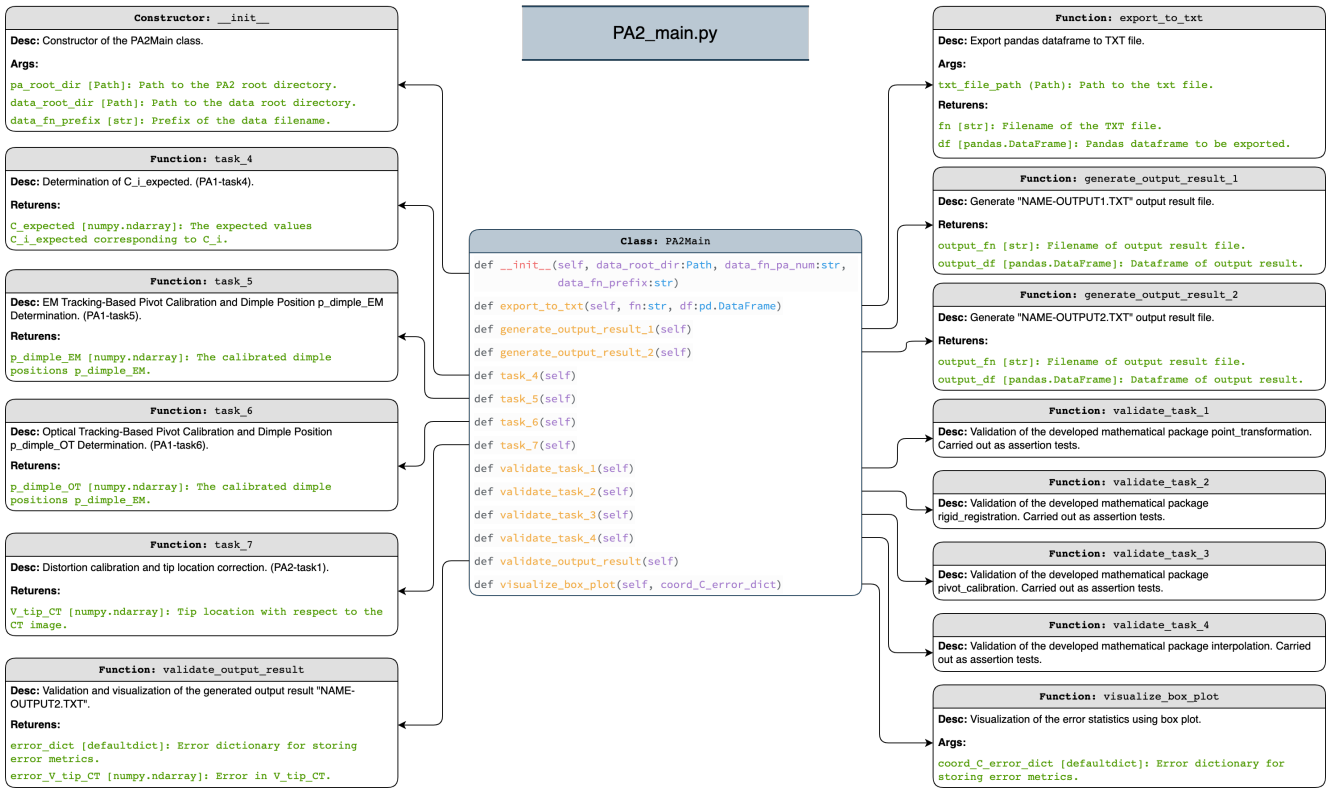

Figure 4. High-level *PA2_main.py* class overview with descriptions of functions, including input arguments, outputs, and intended purpose.

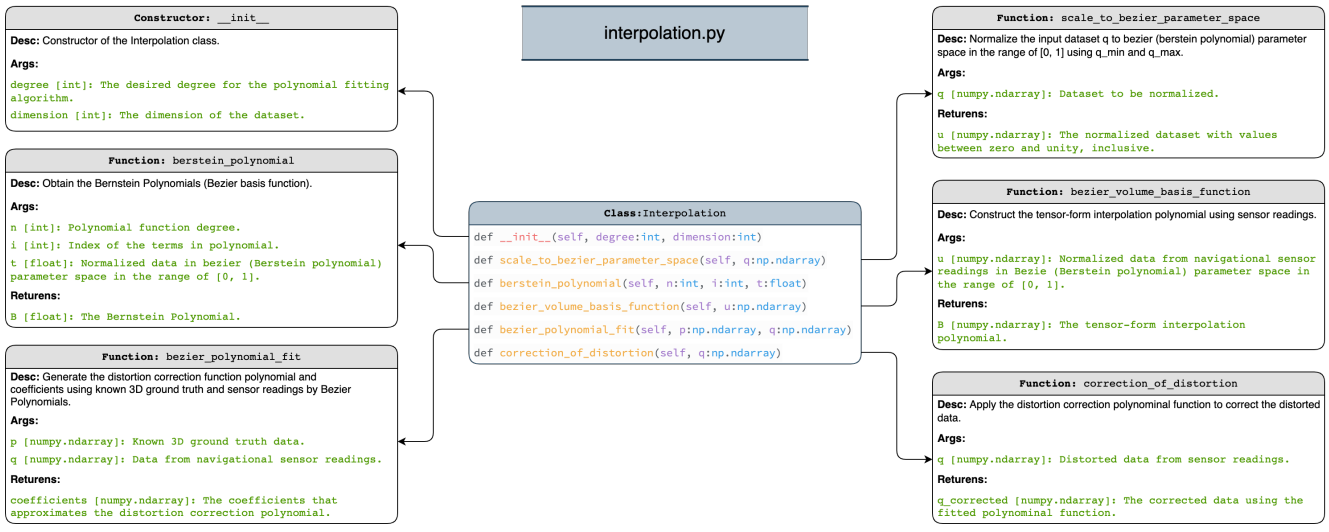

Figure 5. High-level *interpolation.py* class overview with descriptions of functions, including input arguments, outputs, and intended purpose.

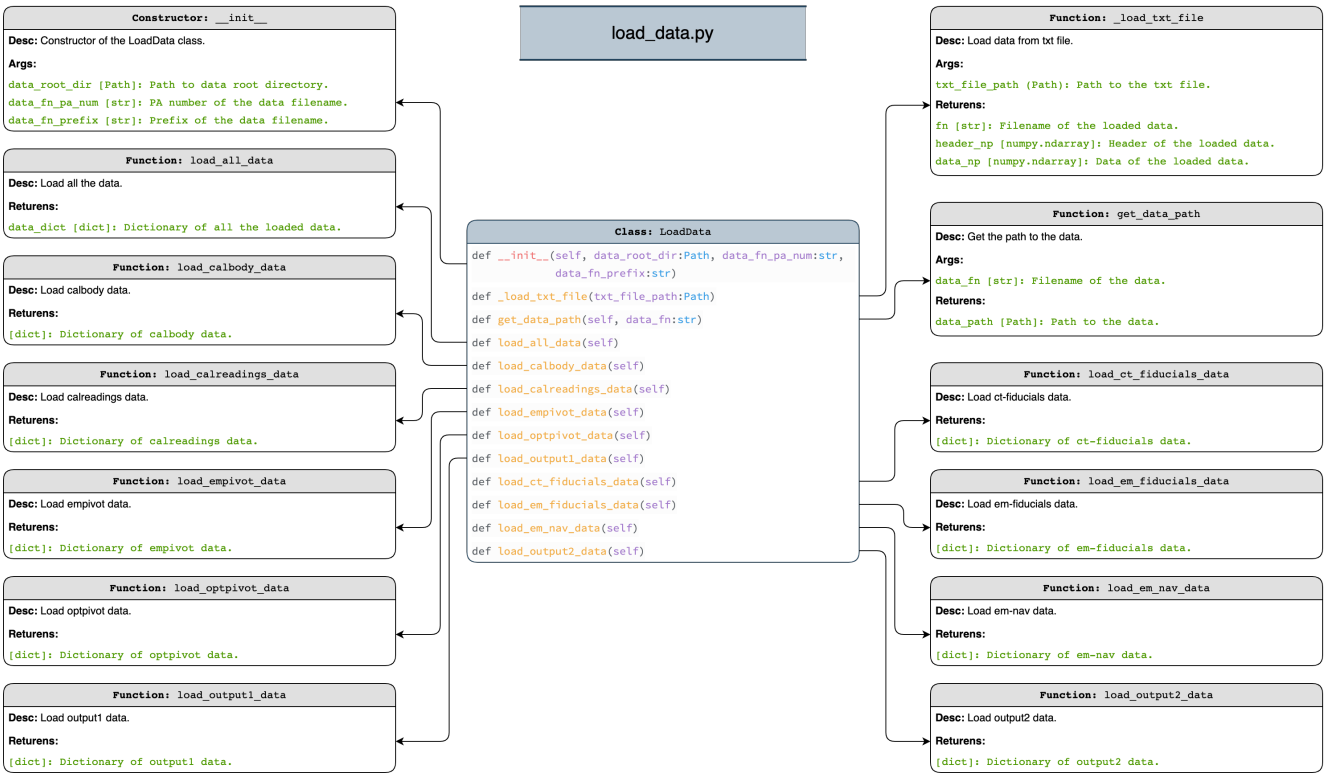

Figure 6. High-level *load_data.py* class overview with descriptions of functions, including input arguments, outputs, and intended purpose.

4 Discussion

This section provides an overview of the validation approaches and results.

4.1 Discussion of Validation Approaches

4.1.1 Validation of Interpolation

The validation approach for the developed mathematical package, *interpolation.py*, discussed in Sec. 1.1.4, is demonstrated through the implementation of the `validate_task_4` function in *PA2_main.py* utilizing the *NumPy* library [1].

We commence by defining the degree and dimension parameters for our Bézier polynomial, denoted as deg and dim, respectively. Our choice for the polynomial degree is set to 5, while the dimension is configured as 3, reflecting our dataset characteristics. Next, we proceed to generate random distortion correction function coefficients $coefficients_{rand}$ and random distorted sensor readings data q_{rand}. Following the formulation methods outlined in Sec. 1.1.4, we calculate the distortion-free ground truth data as $p = B \cdot coefficients_{rand}$. Subsequently, we generate the distortion correction function polynomial and coefficients $coefficients_{computed}$ using the distortion-free ground truth data p and the distorted data q_{rand}. With these computed coefficients, we can derive the distortion-free computed data $q_{corrected}$ using the `correction_of_distortion` function.

To validate our class, we assess the alignment between the computed values $coefficients_{computed}$ and $q_{corrected}$ and their respective ground truth counterparts, $coefficients_{rand}$ and p. We perform assertion tests, employing *NumPy*'s `allclose` function to check for element-wise equivalence within a specified tolerance [1].

The pseudocode that encapsulates this approach, named `validate_task_4` is presented in Algorithm 5:

Algorithm 5 `validate_task_4`

$deg \leftarrow 5,\ dim \leftarrow 3$
$coefficients_{rand} \leftarrow Random((deg+1)^{dim}, dim)$
$q_{rand} \leftarrow Random((deg+1)^{dim}, dim)$

▷ Obtain distortion-free ground truth data
$ip \leftarrow Interpolation(deg, dim)$
$u \leftarrow ip.\texttt{scale\_to\_bezier\_parameter\_space}(q_{rand})$
$F \leftarrow ip.\texttt{bezier\_volume\_basis\_function}(u)$
$p \leftarrow B \cdot coefficients_{rand}$

▷ Obtain distortion-free computed data
$ip.coefficients \leftarrow ip.\texttt{bezier\_polynomial\_fit}(p, q_{rand})$
$coefficients_{computed} \leftarrow ip.coefficients$
$q_{corrected} \leftarrow ip.\texttt{correction\_of\_distortion}(q_{rand})$

▷ Assertion Tests for validating the Interpolation package
Assert: $AllClose(p, q_{corrected})$
Assert: $AllClose(coefficients_{rand}, coefficients_{computed})$

4.2 Discussion of Results

4.2.1 Validation and Visualization of the Output Results

The generation of the output result file, named `NAME-OUTPUT2.TXT`, adheres to the prescribed file format outlined in the programming assignment instructions titled *Programming Assignment 1 and 2* [12]. This file generation process is executed by the `generate_output_result` function in *PA2_main.py*, making use of the *NumPy* and *Pandas* libraries [1, 5].

The validation approach for the output result file is demonstrated through the implementation of the `validate_output_result` function in *PA2_main.py* utilizing the *NumPy* and *Plotly* library [1,3]. The procedure commences with the retrieval of computed data and provided ground truth data from their respective paths, namely `OUTPUT` and `PA2 Student Data`, using the data loader *load_data.py*. Subsequently, we compute the L2 norm (Euclidean distance-based) error metrics between the computed values and the given ground truth values. Following this, we construct error metrics for $\vec{V}^{CT}_{tip,error}$, including measures such as the mean, standard deviation, median, upper and lower quartiles, maximum, and minimum. These error measurements and statistics are then stored in a dictionary. For visualization, we generate 3D visualization plots that depict the computed data alongside the provided ground truth data.

4.2.2 Tabular Summary of the Output Results

Tabs. 1 and 2 provide the tabular summaries of the results contained in `NAME(DEBUG)-OUTPUT2.TXT` and `NAME(UNKNOWN)-OUTPUT2.TXT` respectively. Each row in the table corresponds to the coordinates of $\vec{V}^{CT}_{tip,i}$ [mm], signifying the probe tip position in CT coordinates of the i^{th} data frame in `NAME-EM-NAV.TXT`.

4.2.3 Analysis of Output Results

Following the approach described in Sec. 4.2.1, we have compiled all the error measurements and statistics for the `DEBUG` datasets into a tabular summary, shown in Tab. 3.

Similarly, we have created 3D visualization plots that present the computed data alongside the provided ground truth data, as depicted in Fig. 7. In this visualization, the given and computed $\vec{V}^{CT}_{tip}$ values are shown as solid and open circles, respectively.

It is to be noted that the degree of similarity between the open shapes and solid shapes in the data points serves as a

PA2	debug-a	debug-b	debug-c
$\vec{V}^{CT}_{tip,0}$	$[79.23, 88.90, 70.69]$	$[125.97, 125.24, 158.02]$	$[40.86, 97.35, 85.80]$
$\vec{V}^{CT}_{tip,1}$	$[65.56, 105.70, 106.85]$	$[110.94, 75.88, 161.45]$	$[95.93, 63.21, 108.57]$
$\vec{V}^{CT}_{tip,2}$	$[67.09, 152.55, 131.77]$	$[131.76, 46.74, 34.62]$	$[48.32, 150.62, 105.34]$
$\vec{V}^{CT}_{tip,3}$	$[100.78, 139.66, 160.31]$	$[75.56, 61.41, 133.99]$	$[154.66, 135.35, 52.93]$
PA2	debug-d	debug-e	debug-f
$\vec{V}^{CT}_{tip,0}$	$[165.54, 94.25, 158.83]$	$[109.55, 169.31, 140.71]$	$[46.49, 52.39, 59.30]$
$\vec{V}^{CT}_{tip,1}$	$[106.22, 59.09, 158.42]$	$[85.63, 120.18, 139.92]$	$[54.83, 85.58, 72.01]$
$\vec{V}^{CT}_{tip,2}$	$[71.86, 94.91, 61.70]$	$[43.00, 129.16, 92.00]$	$[60.82, 125.74, 155.60]$
$\vec{V}^{CT}_{tip,3}$	$[158.91, 105.44, 158.65]$	$[43.34, 152.84, 54.47]$	$[113.06, 40.26, 157.17]$

Table 1. Tabular Summary of `NAME(DEBUG)-OUTPUT2.TXT` Results.

PA2	unknown-g	unknown-h	unknown-i	unknown-j
$\vec{V}^{CT}_{tip,0}$	$[82.70, 163.27, 169.82]$	$[55.49, 61.64, 78.37]$	$[155.50, 70.19, 47.63]$	$[72.15, 82.94, 72.26]$
$\vec{V}^{CT}_{tip,1}$	$[93.35, 136.46, 140.79]$	$[94.90, 147.76, 88.57]$	$[39.86, 121.97, 171.54]$	$[65.97, 43.23, 73.27]$
$\vec{V}^{CT}_{tip,2}$	$[49.13, 47.21, 76.66]$	$[145.17, 40.38, 168.41]$	$[60.86, 91.55, 153.28]$	$[163.39, 44.77, 125.75]$
$\vec{V}^{CT}_{tip,3}$	$[71.17, 133.32, 161.19]$	$[132.80, 117.44, 123.80]$	$[109.68, 67.20, 27.82]$	$[162.67, 59.47, 152.39]$

Table 2. Tabular Summary of `NAME(UNKNOWN)-OUTPUT2.TXT` Results.

PA2 Student Data	debug-a	debug-b	debug-c	debug-d	debug-e	debug-f
mean	0.0085	0.2999	0.0430	0.0121	0.1734	0.7249
std	0.0052	0.0816	0.0157	0.0021	0.0541	0.1916
Q1 (lower quartile)	0.0075	0.2517	0.0283	0.0100	0.1323	0.6959
Q2 (median)	0.0100	0.2962	0.0391	0.0121	0.1603	0.8242
Q3 (upper quartile)	0.0110	0.3443	0.0539	0.0141	0.2014	0.8532
max	0.0141	0.4163	0.0656	0.0141	0.2573	0.8557
min	0.0000	0.1908	0.0283	0.0100	0.1158	0.3956

Table 3. Errors Statistics of $\|\vec{V}^{CT}_{tip,computed} - \vec{V}^{CT}_{tip,given}\|_2$ [mm].

visual indicator of the dataset's noise level.

Upon evaluating the tabular table Tab. 3, and the figures for the `DEBUG` datasets as depicted in Fig. 7, we can establish a ranking of the noise levels in the following order:

$$\begin{aligned} &debug-a < debug-d < debug-c \\ &< debug-e < debug-b < debug-f \end{aligned} \tag{25}$$

Since the errors $\|\vec{V}^{CT}_{tip,computed} - \vec{V}^{CT}_{tip,given}\|_2$ for the `DEBUG` datasets are all less than 1.0 [mm], we can confidently assert that our implementation of the assignment tasks is successful.

4.2.4 Validation and Visualization of Error Statistics

Having compiled and stored the error statistics for $\vec{V}^{CT}_{tip,error}$, we take the next step and visualize these statistics using a box plot. This visualization is carried out by the `visualize_box_plot` function in *PA2_main.py*, utilizing the *Plotly* library [3].

4.2.5 Analysis of Error statistics

Following the approach described in Sec. 4.2.4, we can now visualize and analyze the error statistics using the box plot of $\vec{V}^{CT}_{tip,error}$, as illustrated in Fig. 8. The box plot comprises the following components:

(a) *debug-a* (b) *debug-b*

(c) *debug-c* (d) *debug-d*

(e) *debug-e* (f) *debug-f*

Figure 7. 3D visualization of the given and computed $\vec{V}_{tip}^{CT}$.

- *Maximum*: The maximum error value within the dataset.
- *Upper Fence*: The upper threshold that helps identify potential outliers in the data.
- *Q3 (Third Quartile)*: The 75th percentile value of the error data, representing the upper edge of the box.
- *Median*: The middle value of the data, indicated by the line inside the box.
- *Q1 (First Quartile)*: The 25th percentile value of the error data, representing the lower edge of the box.
- *Lower Fence*: The lower threshold that helps identify potential outliers in the data.
- *Minimum*: The minimum error value within the dataset.

For a more detailed view, the underlying error data points are also plotted alongside the box plots.

In the context of box plot analysis, a broader range between the maximum and minimum values signifies greater variability and potential noise in the data, while a larger standard deviation also indicates increased variability and potential noise. As a result, it is evident from the statistics in Tab. 3 and visualization in Fig. 8 that the error rank aligns with the analysis conducted in Sec. 4.2.3. I.e.,

$$\begin{aligned} debug-a < debug-d < debug-c \\ < debug-e < debug-b < debug-f \end{aligned} \tag{26}$$

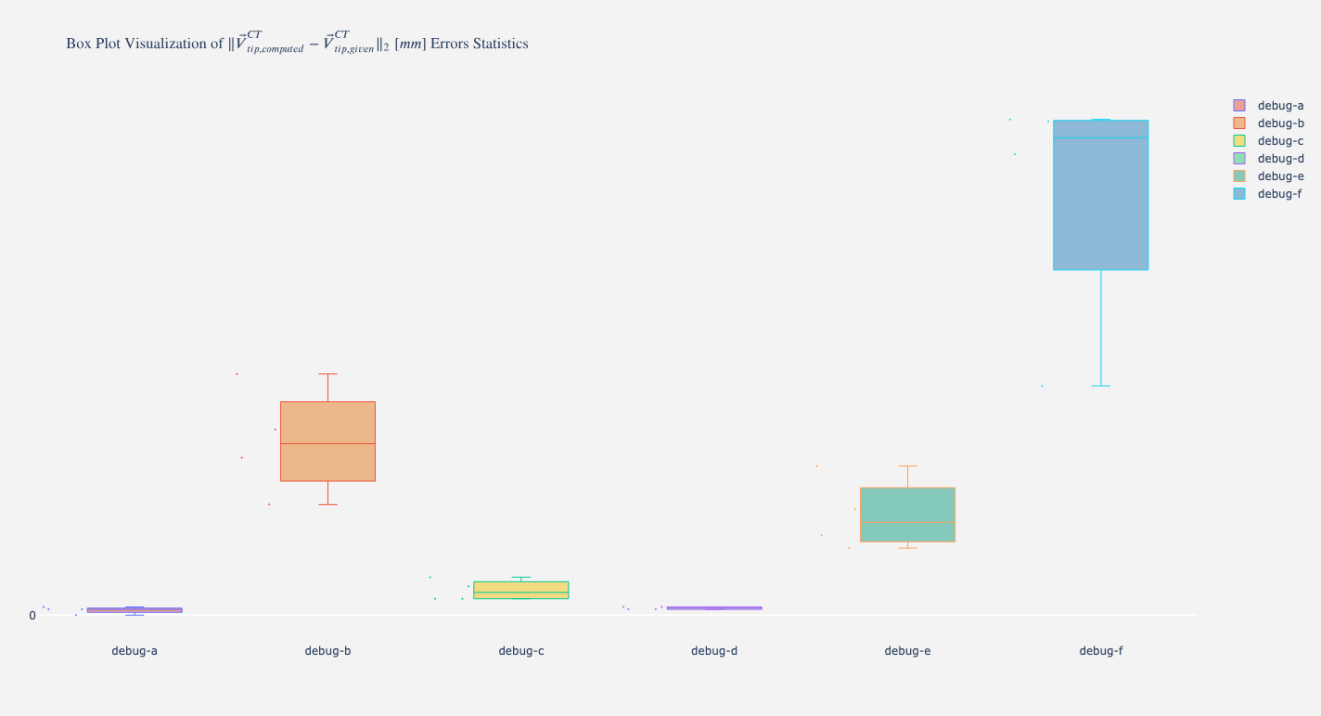

Figure 8. Box Plot Visualization of $\|\vec{V}^{CT}_{tip,computed} - \vec{V}^{CT}_{tip,given}\|_2$ [mm] Error Statistics.

Moreover, according to the programming assignment instructions titled *Programming Assignment 1 and 2* [12], the datasets were generated with different amounts of simulated sources of noise, including *EM Distortion*, *EM noise*, and *OT Jiggle*. Tab. 4 indicates which data sets have non-zero values for the different quantities.

PA2 Student Data	EM Distortion	EM Noise	OT Jiggle
debug-a	0	0	0
debug-b	0	✓	0
debug-c	✓	0	0
debug-d	0	0	✓
debug-e	✓	✓	✓
debug-f	✓	✓	✓

Table 4. Sources of Noise in PA2 Student Data.

As demonstrated in Fig. 1 and Sec. 1.2, the process of determining the tip location with respect to the CT frame $\vec{V}^{CT}_{tip}$ does not necessitate the use of the OT frame. Consequently, our primary focus lies on the *EM Distortion* and *EM Noise* columns in Tab. 4. Our observations Eq. (26) closely align with the presence or absence of these noise sources.

For datasets *debug-a* and *debug-d*, none of these noise sources are present, resulting in errors close to zero. In the case of *debug-c* where *EM Distortion* is introduced, the application of our distortion correction procedures and the dewarping of the EM tracker space ensure that the error remains only slightly larger than zero, akin to *debug-a* and *debug-d*. On the other hand, the remaining datasets, namely *debug-b*, *debug-e* and *debug-f*, feature *EM Noise*, resulting in larger errors compared to the other datasets.

Overall, we can confidently affirm that the implementation of the mathematical packages, as well as the efforts to correct the distortion and dewarp the EM space, have been a success.

4.2.6 Complete Listings of `OUTPUT.TXT` Files

Please refer to the *OUTPUT* subfolder for the complete `NAME-OUTPUT-1.TXT` and `NAME-OUTPUT-2.TXT` results. Since the primary objective of *Programming Assignment 2* is to compute $\vec{V}^{CT}_{tip}$ and generate the corresponding `NAME-OUTPUT-2.TXT` results, we have included tabular summaries of them in the preceding subsections.

5 Statement

5.1 Contribution

Iou-Sheng (Danny) Chang was responsible for drafting the data loader named *load_data.py*, creating the structure of *PA2_main.py* and the command-line interfaces. Ching-Yang

(Austin) Huang, on the other hand, worked on drafting the mathematical package *interpolation.py* and the validation for it.

During the debugging and finalization phases of the program, the authors collaborated closely through pair programming. Additionally, the report was a joint effort, with both authors contributing equally to its completion.

5.2 Alignment with Course Materials

The formulation and algorithmic approaches of the assignment are in close alignment with the methods presented in several key course materials, including lecture notes such as *Cartesian Coordinates, Points, and Transformations*, *Point Cloud to Point Cloud Rigid Transformations*, *Calibration*, and *Interpolation Review* [8–11]. Additionally, the methods outlined in the programming assignment instructions *Programming Assignment 1 and 2* [12] have also played a significant role in shaping our approach.

5.3 External Libraries Utilized

The program submitted for this assignment is implemented in *Python* [7], incorporating a selection of external libraries, including [1, 3–6, 13].

References

[1] Charles R Harris, K Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, 2020. 5, 6, 9, 13

[2] Chad B. Hovey. Formulation and python implementation of bézier and b-spline geometry. https://hovey.github.io/docs/Hovey_2022_Bezier_B-Spline_SAND2022-7702_C.pdf, 2022. SAND2022-7702 C, National Technology and Engineering Solutions of Sandia, LLC (NTESS). 1, 4

[3] Plotly Technologies Inc. *Collaborative Data Science*. Montreal, QC, 2015. 5, 9, 10, 13

[4] Will McGugan. *Rich API*, 2020. 13

[5] Wes McKinney et al. Data structures for statistical computing in python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, volume 445, pages 51 – 56. Austin, TX, 2010. 5, 6, 9, 13

[6] Pallets. *Click*, 2014. 6, 13

[7] Python Core Team. *Python: A dynamic, open source programming language*. Python Software Foundation, 2019. 5, 6, 13

[8] Russell H. Taylor. Calibration. https://ciis.lcsr.jhu.edu/lib/exe/fetch.php?media=courses:455-655:lectures:calibration.pdf, 2023. Supplied as course notes in Computer Integrated Surgery I, Johns Hopkins University. 13

[9] Russell H. Taylor. Cartesian coordinates, points, and transformations. https://ciis.lcsr.jhu.edu/lib/exe/fetch.php?media=courses:455-655:lectures:frames.pdf, 2023. Supplied as lecture notes in Computer Integrated Surgery I, Johns Hopkins University. 13

[10] Russell H. Taylor. Interpolation review. https://ciis.lcsr.jhu.edu/lib/exe/fetch.php?media=courses:455-655:lectures:interpolationreview.pdf, 2023. Supplied as course notes in Computer Integrated Surgery I, Johns Hopkins University. 1, 4, 13

[11] Russell H. Taylor. Point cloud to point cloud rigid transformations. https://ciis.lcsr.jhu.edu/lib/exe/fetch.php?media=courses:455-655:lectures:rigid3d3dcalculations.pdf, 2023. Supplied as lecture notes in Computer Integrated Surgery I, Johns Hopkins University. 13

[12] Russell H. Taylor. Programming assignments 1 and 2. https://ciis.lcsr.jhu.edu/lib/exe/fetch.php?media=courses:455-655:2023:programmingassignments1and2.pdf, 2023. Supplied as course assignment instructions in Computer Integrated Surgery I, Johns Hopkins University. 1, 5, 9, 12, 13

[13] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. 5, 13