CIS I - Programming Assignment 2 Report

Shreya Terala & Yvonne Zhang

January 16, 2026

1 Introduction

This report will focus on the tasks presented in Assignment 2, including the development of mathematical tools and subroutines for the calibration, registration, and tracking of a stereotactic navigation system using an electromagnetic position tracking device [1]. All programs were written in MATLAB for this project and will be used for all subsequent programming assignments.

2 Summary of Problems To Be Solved

Problems to be solved are as follows: the registration of the stereotactic navigation system (which comprises an optical tracker and an EM tracker), calibration of the optical and EM pointer probes, distortion correction for the EM tracking system, and registration between the EM tracking system and the CT images.

In summary, solving these problems will enable an accurate calibration between the EM tracking system used in the workspace and the CT images that are being referenced. More details on the variables and terminology used in the stereotactic navigation system problem scenario are covered below [1]. Diagrams were recreated for the team's comprehension.

2.1 Stereotactic Navigation System Registration

The setup includes a calibration object moving in the workspace, an optical tracker and the EM tracker base. The following variables define the stereotactic navigation system and the workspace.

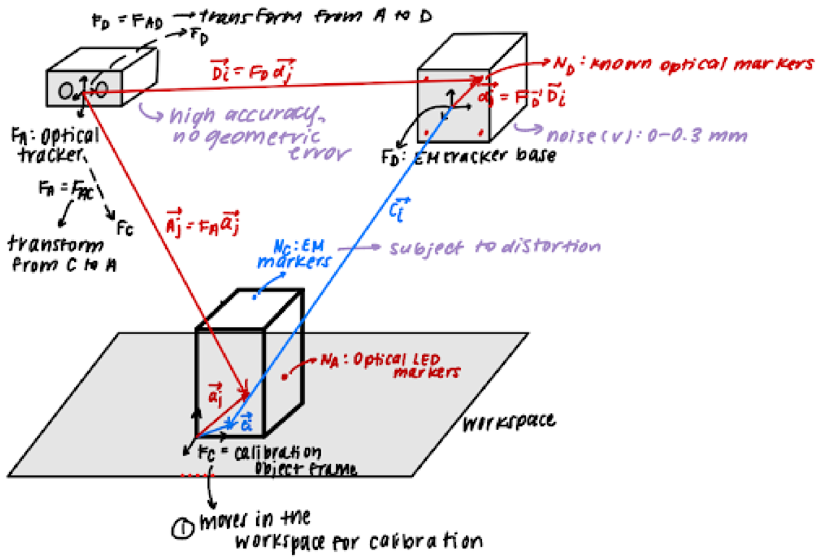
$F_D = F_{AD}$ → transform from A to D
F_D
$\vec{D_i} = F_D \vec{d_j}$
N_D : known optical markers
$\vec{d_j} = F_D^{-1} \vec{D_i}$
high accuracy, no geometric error
noise (v): 0–0.3 mm
F_A: optical tracker
F_D: EM tracker base
$F_A = F_{AC}$
F_C
transform from C to A
$\vec{A_j} = F_A \vec{a_j}$
$\vec{C_i}$
N_C: EM markers
subject to distortion
N_A: Optical LED markers
$\vec{a_j}$
$\vec{c_i}$
workspace
F_C = calibration object frame
① moves in the workspace for calibration

Figure 1: Stereotactic Registration System Setup

Error v = 0 to 0.3 mm
N_C - Calibration object EM markers
N_A - Calibration object optical LED markers
N_D - Optical markers in the EM tracking system

F_C - Frame calibration object coordinate system
F_A - Frame transformation from F_C to frame A (optical tracker)
F_D - Frame transformation from frame A (optical tracker) to frame D (EM tracker base)

C_i - Positions from frame D (EM tracker base) to EM markers on the calibration object
c_i - Known positions relative to F_C for i = $1 \dots N_C$
A_j - Positions from frame A (optical tracker) to optical LED markers on the calibration object
a_j - Position of optical markers relative to F_C frame on calibration object
D_j - Positions from frame A (optical tracker) to optical markers on EM base markers (N_D)
d_j - Positions of optical markers on the EM tracker base relative to frame D

Calibration data sample frame - position of the calibration object in the workspace, optical markers, and EM markers $[D_1, \dots, D_{N_D}, A_1, \dots, A_{N_A}, C_1, \dots, C_{N_C}]$

The end result of this registration process will be the C_i values read by the EM tracker and the ground truth $C_{expected}$ values calculated using the optical tracker's readings.

See Section 3.1 for the mathematical approach.

2.2 Pivot Calibration

Two pointer probes are used, one with LED markers and one with electromagnetic (EM) markers with unknown positions and orientations with respect to the EM tracker system. The pivot calibrations of each probe are given using the dimpled posts in the workspace. The following variables define the calibration probe systems.

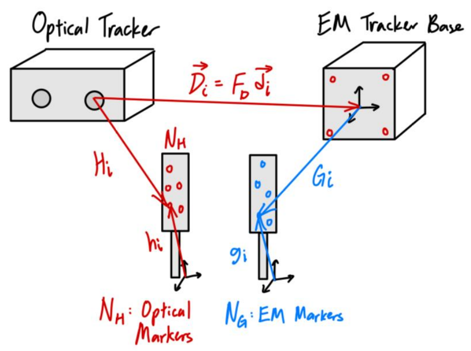

Figure 2: Pivot Calibration System Setup

N_H - LED markers unknown with fixed position h_i
N_G - EM markers unknown with fixed position g_i

The pivot calibration data at each post are given as:
Optical probe calibration: $[D_1, \dots, D_{N_D}, H_1, \dots, H_{N_H}]$
EM probe calibration: $[G_1, \dots, G_{N_G}]$

Following the pivot calibration process, the frame transformations relating the pointers to their respective trackers are calculated and the estimated position of the pivot calibration posts.

See Section 3.2 for the mathematical approach.

2.3 Distortion Correction

The distortion correction process creates a nonlinear mapping between the distorted EM tracker measurements and their corresponding ground-truth coordinates obtained from the optical tracker. This mapping is modeled using a fifth-order, three-dimensional Bernstein polynomial, which provides a smooth and flexible basis for capturing nonlinear spatial distortions across the measurement volume.

The bounding box around the normalized EM coordinates extends 5% beyond the minimum and maximum bounds of the measured data in each dimension to ensure numerical stability and prevent boundary effects. The coefficients of each coordinate axis (x, y, z) are determined using **lsqminnorm**, minimizing residual error between the measured and ground truth points.

See Section 3.3 for the mathematical approach.

2.4 CT Image Registration

CT images taken prior need to be calibrated to the EM tracker being used to navigate around the workspace. The position of the EM probe on each fiducial is captured and the transformation mapping between the EM tracker readings and the CT image needs to be calculated. The following variables describe the position of the fiducials with respect to the CT image and the EM tracker.

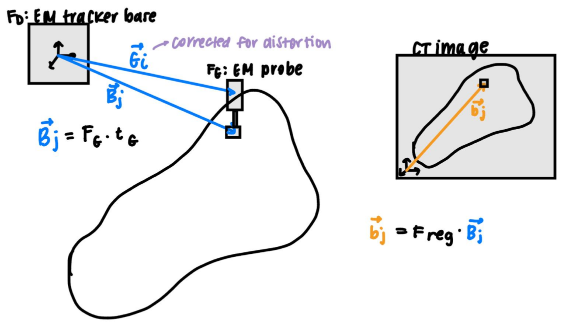

Figure 3: CT Registration Setup

b_j - fiducial landmarks in CT scan coordinate system
B_j - position of the pointer tip relative to the EM base coordinate system
N_B - number of fiducial landmarks
F_{reg} - transformation from Bj to bj (from EM frame to CT frame)
N_{frames} - number of frames of data

With the CT image registration process, a mapping between the undistorted EM tracker readings and the CT images will be generated.

See Section 3.4 for the mathematical approach.

3 Mathematical Approach

3.1 3D Point Set Registration Algorithm

A primary function, "**point_cloud_registration.m**", was developed to register the correspondence between the two 3D point sets. This function accepts two sets of corresponding points and returns the aligning transformation matrix using Arun's method [2]. The output is the rotation matrix and the translation vector that form the frame transformation matrix.

The following will use the lower case x_i for the calibration markers vector and X_i for the positions of the markers. This can correspond to all variables A, C, D, G, and H used in this assignment. Note that a similar algorithmic approach was taken for the "pivot_calibration.m" function to determine the transformations of each frame [k] of F_G.

First, the inputs from the data reading function is allocated to variables x_i and X_i for calibration markers and calibration frames readings respectively. The size of both markers and frames are also stored. Arun's method is used to find the optimal frame transformation in closed form to estimate the frame transformation that best aligns the set of known x_i object points with corresponding 3D points X_i for each frame. For each frame, the following are solved to form the $SE(3)$ frame transformation [2]:

$$X_i \approx Rx_i + t, \quad R \in SO(3), \quad t \in R^3 \tag{1}$$

for $i = 1, ..., N$ in each frame.

Next, the translation is removed for centroid alignment to isolate the rotation from the translation effects of the transformation. The centroids of both point sets are calculated:

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i, \quad \bar{X} = \frac{1}{N}\sum_{i=1}^{N} X_i \tag{2}$$

The centroids are subtracted from each data point to obtain the centered coordinates:

$$x_i' = x_i - \bar{x}, \quad X_i' = X_i - \bar{X} \tag{3}$$

The optimal rotation is determined by minimizing the following equation with translation effects removed [2]:

$$\min_{R\in \mathrm{SO}(3)} \sum_{i=1}^{N} \|X_i' - Rx_i'\|^2 \tag{4}$$

By expanding the squared norm:

$$\|X_i' - Rx_i'\|^2 = (X_i' - Rx_i')^T (X_i' - Rx_i') = \|X_i'\|^2 + \|Rx_i'\|^2 - 2(X_i')^T Rx_i' \tag{5}$$

where rotation has no effect inside a norm for $||Rx_i'||^2 = ||x_i'||^2$, and $||X_i'||^2$ and $||x_i'||^2$ are independent of rotation, then:

$$\min_R \sum_i \|X_i' - Rx_i'\|^2 \longleftrightarrow \max_R \sum_i (X_i')^T Rx_i' \tag{6}$$

which identifies the covariance matrix that is defined as,

$$H = \sum_{i=1}^{N} (x_i')(X_i')^\top = (x_i')\left(X_i'^\top\right) \tag{7}$$

The covariance matrix H is the correlation between the two centered point sets by encapsulating how the two sets are aligned in orientation. Its singular value decomposition can be found using the svd function for the following equation:

$$H = USV^\top \tag{8}$$

The optimal rotation is solved by utilizing the diagonal matrix for the weight of the diagonal entries of the covariance matrix H, therefore the weights of the mapping from x_i to X_i. The diagonal is therefore the trace of (7) and by equating to (8) [3],

$$tr((X')^T Rx') = tr(RH) = tr(RUSV^\top) = tr(SV^\top RU) \tag{9}$$

Let $V^\top RU = M$, where M is an orthogonal matrix for orthogonal $V^\top, R$ and U, and $tr(SM)$ is maximized if m-components equal to 1. Therefore, M is the identity matrix and the optimal rotation is given by the svd orthogonal matrices:

$$M = I = V^\top RU \to V = RU \to R = VU^\top \tag{10}$$

Finally, the translation is determined by aligning the centroids such that:

$$t = \bar{X} - R\bar{x} \tag{11}$$

The rotations and translations of each frame is stored in "rotationMatrix_all" and "translationVector_all" respectively. The function outputs "rotationMatrix" and "translationVector" for each X dataset for A, C, D, G, and H.

3.1.1 Zero Singular Value Case

In practice, there are degenerate cases when the covariance matrix H is rank-deficient ($det(R) \approx 0$)). This occurs if the points are collinear or coplanar, therefore reducing the dataset dimension. In these cases, H carries at least one singular value of zero or numerically very small; therefore, the corresponding rotation is under-determined.

The algorithm detects the singular values from the SVD as follows:

```
if min(diag(S)) < 1e-12
    warning('Frame %d: Covariance matrix near-singular / zero
        singular value detected, therefore best-fit rotation will
        be used.', f);
    % R=V*U' gives best-fit rotation even with zero singular value
end
```

Even though the rotation is not fully determined along the degenerate axes, the least-squares best-fit rotation is still found.

3.1.2 Reflection Case

In addition, there are reflection cases ($det(R) < 0$) handled by using the following:

```
if det(R) < 0
    V(:,3) = -V(:,3); % flip last column of V to correct
        reflection
    R = V * U'; % resolve for R
    warning('Frame %d: Reflection detected and corrected.', f);
end
```

SVD orders the singular values along its diagonal from largest to smallest. Therefore, the last column is often the least constrained by the data and has minimal effects on the axes' alignment. Flipping the last column of V therefore corrects the determinant to positive while preserving the main alignment.

These two pieces of code are used in the "point_cloud_registration.m" function to mediate corner cases.

3.2 Pivot Calibration Algorithm

The following steps follow after finding the frame transformation of $F_G[k]$ using the EM tracking data for the EM probe as described in Section 3.1.

The function "**pivot_calibration.m**" is implemented to determine the positive relative to the EM tracker base coordinate system to the dimple in the calibration post. Some new variables need to be defined for all frame [k] in F_G [1]:

P_{dimple} - Fixed pivot point in EM tracker coordinates and is constant for every frame in F_G,
t_G - Tool tip location in EM probe coordinates.

Therefore, the following setup of transformation is setup to solve the least squares problem [1]:

$$\begin{aligned} P_{dimple} &= F_G \cdot t_G \\ P_{dimple} &= R \cdot t_G + t \\ R \cdot t_G - I \cdot P_{dimple} &= -t \\ [R - I][t_G; pivot\_point] &= -t \end{aligned} \tag{12}$$

This forms a least square problem in the form of $Ax = b$,
where $A = [R - I]$, $x = [t_G; pivot\_point]$ and $b = -t$.

The "**pivot_calibrations.m**" function uses the lsqr function to solve x, where the bottom 3 values give the coordinates of the pivot point to solve this problem.

3.3 Distortion Correction Algorithm

The distortion correction function, "**distortion_correction.m**", generates an estimation model of a sensor's measurement error at a given position and orientation in the workspace. The calibration coefficient is based on the body calibration between C_i and $C_{expected}$ in each frame of $[k]$ data [1]. The goal is to calibrate the measured, distorted points of C_i to the optical, ground truth $C_{expected}$. This forms a smooth mapping as a linear combination of Bernstein basis functions for the corrected points [4].

$$f_{ijk}(u,v,w) = \sum_{i=0}^{n}\sum_{j=0}^{n}\sum_{k=0}^{n} c_{1,j,k} B_{5,i}(u)\, B_{5,j}(v)\, B_{5,k}(w) \tag{13}$$

The raw data with distortion (C_i) is scaled to a box to form the correction function in the bounding cube $[0,1]^3$ [5].

$$u = \frac{x - x_{\min}}{x_{\max} - x_{\min}}, \quad v = \frac{y - y_{\min}}{y_{\max} - y_{\min}}, \quad w = \frac{z - z_{\min}}{z_{\max} - z_{\min}}. \tag{14}$$

Bernstein polynomials are used to model measurement error in the 3D measurement space to the 5th order; therefore, six error values are interpolated for each measurement and $6^3 = 216$ basis terms using the following equation.

$$B_i^n = \binom{n}{i} x^i (1-x)^{n-i}, \quad \text{where} \quad \binom{n}{i} = \frac{n!}{i!(n-i)!}. \tag{15}$$

This forms a matrix F of Bernstein basis functions evaluated for N measured points, therefore forms a $[Nx216]$:

$$F_{ijk}(u,v,w) = B_{5,i}(u)\, B_{5,j}(v)\, B_{5,k}(w) \tag{16}$$

$$F = \begin{bmatrix} B_1(u_1,v_1,w_1) & B_2(u_1,v_1,w_1) & \dots & B_{216}(u_1,v_1,w_1) \\ B_1(u_2,v_2,w_2) & B_2(u_2,v_2,w_2) & \dots & B_{216}(u_2,v_2,w_2) \\ \vdots & \vdots & \ddots & \vdots \\ B_1(u_N,v_N,w_N) & B_2(u_N,v_N,w_N) & \dots & B_{216}(u_N,v_N,w_N) \end{bmatrix} \tag{17}$$

To solve the unknown coefficient for the polynomial, the following least squares problem is solved for x, y, z with known coordinates p_r ($C_{expected}$) and (u,v,w) are the normalized coordinates of C_i [5]:

$$\begin{bmatrix} F(u,v,w) \end{bmatrix} \begin{bmatrix} \mathbf{c} \end{bmatrix} = \mathbf{p}_s, \quad \text{or component-wise:} \quad \begin{bmatrix} F(u) \\ F(v) \\ F(w) \end{bmatrix} \begin{bmatrix} \mathbf{c}_x \\ \mathbf{c}_y \\ \mathbf{c}_z \end{bmatrix} = \begin{bmatrix} x_r \\ y_r \\ z_r \end{bmatrix}. \tag{18}$$

By solving the coefficients of the least squares problem, corrected points are found using equation 13 to evaluate the Bernstein polynomial at the normalized measured position.

3.4 CT Image Registration Algorithm

The CT image registration algorithm computes the rigid transformation between the EM tracker coordinate frame and the CT image coordinate frame:

$$F_{\text{reg}} = \begin{bmatrix} R_{\text{reg}} & t_{\text{reg}} \\ 0 & 1 \end{bmatrix}. \tag{19}$$

Each fiducial has a corresponding location in CT coordinates and the marker locations on the EM probe with respect to the EM tracking system. For each of the N_B fiducials, the measured marker positions in the EM tracker frame are collected into G_j. To correct for EM field distortion, the polynomial correction approach from Section 3.3 is applied to G_j prior to any calculations.

Let $G_{\text{ref}} \in R^{N_G \times 3}$ denote the probe marker coordinates in the EM frame that were used as reference during the previous pivot calibration procedure. The corrected marker positions are related by a rigid transformation:

$$G_{\text{j}}^{(i)} = R^{(i)} G_{\text{ref}} + t^{(i)}, \quad R^{(i)} \in SO(3),\ t^{(i)} \in R^3, \tag{20}$$

where $(R^{(i)}, t^{(i)})$ is solved using the point set registration method from Section 3.1 (equations (7)–(8)). Given the calibrated probe tip t_G in the probe coordinates from pivot calibration (Section 3.2, equation (12)), the tip position in the EM frame is

$$B_j^{(i)} = R^{(i)} t_G + t^{(i)}. \tag{21}$$

The corresponding CT-space fiducial points are given by b_j which are used to construct the rigid transformation aligning EM-space fiducials to CT-space is obtained using the same least-squares registration framework as Section 3.1 (equations (7)–(8)):

$$b_j^{(i)} \approx R_{\text{reg}} B_j^{(i)} + t_{\text{reg}}, \quad \forall i \in \{1, \ldots, N_B\}. \tag{22}$$

Following the point cloud registration procedure in Section 3.1, the solution is computed, giving the resulting transformation maps any point from EM tracker coordinates to CT coordinates:

$$p_{\text{CT}} = R_{\text{reg}} p_{\text{EM}} + t_{\text{reg}}. \tag{23}$$

For any general point v_i being located in the CT image with the EM probe, the same process above would be repeated to produce the corresponding EM probe tip coordinates V_i. Then the following equation, based on the previously calculated registration transformation, would be used to determine the coordinated in the CT image:

$$v_{\text{i}} = R_{\text{reg}} V_{\text{i}} + t_{\text{reg}}. \tag{24}$$

4 Algorithmic Approach

4.1 Cartesian Math Package

MATLAB's Symbolic Math Toolbox was used as the Cartesian math package for 3D points, rotations, and frame transformations [6].

Basic mathematical operations for the matrices were used, such as addition, matrix multiplication, and matrix transpose using single quotes ('). These operations were used throughout the assignment.

From Matlab's core numerical linear algebra libraries, the **svd** function performs a singular value decomposition of matrix H, such that [U,S,V] = svd(H) is used to solve $H = USV^T$ [7]. Singular value decomposition is used to solve for the covariance matrix in its matrix-vector product components. This includes the two orthogonal matrices U and V with columns formed by the left and right singular vectors of H respectively. S is a diagonal matrix containing the singular values of H [8]. This decomposition of the covariance will be utilized to determine the rotation matrix.

The **lsqr** function is used to solve the system of linear equations using the iterative least squares method. It solves the system by approximating $Ax \approx b$ by minimizing the Euclidean norm of the residual of [2]:

$$x = min_x ||b - Ax||^2 \tag{25}$$

The least squares algorithm is an adaptation of the conjugate gradient method for rectangular matrices, where $Ax = b$ produces the same residuals as the conjugate gradient for the normal equation $A^T Ax = A^T b$, without forming the explicit $A^T A$ that results in superior numerical stability [9]. Note, the lsqr function will use a tolerance of $1e - 6$, which is sufficiently stable for our use case [9].

For Programming Assignment 2, in the case of an under-determined (more columns than rows) solution for solving the Bernstein basis coefficients, the function **lsqrminnorm** is used to solve 25 while also minimizing $||x||^2$ from the possible solutions [10]. This returns a unique, smallest-norm solution even if the system is under-determined or ill-conditioned for a more numerically stable polynomial fitting.

4.2 Reading Data Files

Algorithms were developed to read the provided data files by extracting the locations of the markers and their corresponding locations in each frame. The calibration data for frames D, A and C, is extracted using the function "**read_cal_data.m**". The pivot calibration data for the EM probe with respect to the EM tracker base is extracted using the function "**read_em_pivot_data.m**". The pivot calibration data for the optical probe with respect to the optical tracker base utilizing data from H and D is extracted using the function "**read_optical_pivot_data.m**".

The input is the string of the reading filepath.
For "**read_em_pivot_data.m**" and "**read_optical_pivot_data.m**" functions, the string for empivot and optpivot is called upon to parse the uppercase X_i values. For "read_cal_data.m", there are two sets of data: calbody for the lowercase x_i markers and calreadings for the uppercase X_i coordinates.

The output for all files is the parsing of each file's header to use for all functions and algorithms of this project. For X_i, it is returned in the form of $N_{markers} \times 3 \times N_{frames}$ for each respective frame F_X. Notes that for calibration, x_i sets return $N_{markers} \times 3$ values.

The data reading algorithm first reads the data file using the function readmatrix. The first row of each file is split for "," to parse the number of each marker as according to the "Data file formats" section in the assignment [1].

The number of rows for each frame is partitioned from the heading and each frame's data block is extracted for each marker group in 3D arrays. Each frame's corresponding transformations (D, A, C, G and H) can be individually accessed for the project algorithms.

Algorithm 1 Parse Calibration Files

```
1: **Input:** Calibration body file, Calibration readings file
2: **Output:** Marker positions $(d_i, a_i, c_i)$ and frame data $(F_D, F_A, F_C)$
3:
4: Read first line of readings file to get $N_D, N_A, N_C, N_{\text{frames}}$
5: Read calibration body data and extract $d_i, a_i, c_i$ marker positions
6:
7: **for** each frame $i = 1$ to $N_{\text{frames}}$ **do**
8:     Extract frame block of $(N_D + N_A + N_C)$ rows
9:     Split into $F_D(:,:,i), F_A(:,:,i), F_C(:,:,i)$
10: **end for**
11:
12: **return** $d_i, a_i, c_i, F_D, F_A, F_C$
```

4.3 3D Point Set Registration Algorithm

The "**point_cloud_registration.m**" function computes the rigid-body transformation to align two corresponding sets of 3D points based on Arun's method as described mathematically in Section 3.1.

The **svd** function is used to solve for the rotation matrix. The **mean** function is used throughout this algorithm to find the centroids of the markers.

Algorithm 2 Point Cloud Registration Using Arun's Method

```
**Input:** $calmarkers$ (Nx3), $calframereadings$ (Nx3xF)
**Output:** $rotationMatrix$ (3x3xF), $translationVector$ (3xF)

$N_{\text{markers}}$ = number of rows in $calmarkers$
$N_{\text{frames}}$ = number of frames in $calframereadings$

$centroid_{x_i}$ = mean of $calmarkers$ along rows
Initialize $rotationMatrix\_all = zeros(3, 3, N_{\text{frames}})$
Initialize $translationVector\_all = zeros(3, N_{\text{frames}})$

**for** $f = 1$ to $N_{\text{frames}}$ **do**
    $X_{\text{frame}} = calframereadings(:, :, f)$                ▷ Current frame positions
    $centroid_X$ = mean of $X_{\text{frame}}$ along rows

    $x' = calmarkers - centroid_{x_i}$
    $X' = X_{\text{frame}} - centroid_X$

    $H = (x')^T \cdot X'$                ▷ Covariance matrix
    $[U, S, V] = \text{svd}(H)$
    $R = V \cdot U^T$                ▷ Compute rotation

    **if** $\det(R) < 0$ **then**
        $V(:, 3) = -V(:, 3)$                ▷ Correct reflection
        $R = V \cdot U^T$
    **end if**

    $t = centroid_X^T - R \cdot centroid_{x_i}^T$                ▷ Compute translation

    $rotationMatrix\_all(:, :, f) = R$
    $translationVector\_all(:, f) = t$
**end for**

**return** $rotationMatrix\_all$, $translationVector\_all$
```

4.3.1 Computing $C_{expected}$

The expected values for the C_i were found in Programming Assignment 1 using the following files all under the programs/problems directory:

"**pa1_problem4a.m**",
"**pa1_problem4b.m**", and
"**pa1_problem4d.m**".

In summary, the transformation F_D is found between the optical tracker and EM tracker coordinates (part a), and similar steps were taken for finding F_A (part b). The 3D point set registration algorithm described in Section 3.1 is implemented using the function "**point_cloud_registration.m**" in the previous Section 4.3 for parts a and b.

The following shows the algorithm for part a to solve for F_D, and part b uses the exact same algorithm for subscripts of F_A instead.

Algorithm 3 Compute Frame Transformations F_D and F_A Using Point Cloud Registration

```
1: Input:
2:     $calbody\_file$: file containing calibration marker positions
3:     $calreadings\_file$: file containing marker readings for multiple frames
4: Output:
5:     $rotationMatrix_D, translationVector_D$: Frame D transformations (3x3xF, 3xF)
6:
7: Read calibration data:
8:     $d_i, FD = \texttt{read\_cal\_data}(\text{calbody\_file, calreadings\_file})$
9:
10: Compute Frame D transformation:
11:     $rotationMatrix_D, translationVector_D = \texttt{point\_cloud\_registration}(d_i, FD)$
12: return $rotationMatrix_D, translationVector_D$
```

Finally, "**pa1_problem4d.m**" calculates the expected positions of EM markers on the calibration body by applying the known transformations for Frame D and Frame A from parts a and b. This is found using the following equation:

$$C_i^{expected} = F_D^{-1} \cdot F_A \cdot c_i \tag{26}$$

for each marker c_i in each frame.

The **rigidtform3d** function is used to form the rigid transformation matrices for D (tformD) and A (tformA). The **transformPointsInverse** function solves equation 26 for variable transformed_ci.

Algorithm 4 Compute Expected EM Marker Positions C_{expected}

```
1: Input: $calbody\_file$, $calreadings\_file$, $rot_D$, $trans_D$, $rot_A$, $trans_A$
2: Output: $C_{\text{expected}}$, $N_{\text{markers}}$, $N_{\text{frames}}$
3:
4: Read calibration marker data: $c_i = \texttt{read\_cal\_data}(calbody\_file, calreadings\_file)$
5: $N_{\text{markers}}$ = number of rows in $c_i$
6: $N_{\text{frames}}$ = number of frames (third dimension of $rot_D$)
7:
8: Initialize $C_{\text{expected}}$ as empty array of size $(N_{\text{markers}} \cdot N_{\text{frames}}) \times 3$
9:
10: for each frame $i$ do
11:     $tform_D$ = rigid transform for Frame D          ▷ Form 4x4 transformation matrices
12:     $tform_A$ = rigid transform for Frame A
13:     for each marker $j$ do
14:         $transformed\_ci = F_D^{-1} \cdot F_A \cdot c_i(j)$          ▷ Solve equation 26
15:         Round $transformed\_ci$ and store in $C_{\text{expected}}$
16:     end for
17: end for
18:
19: return $C_{\text{expected}}, N_{\text{markers}}, N_{\text{frames}}$
```

4.4 Distortion Correction Algorithm

The function "**distortion_correction.m**" corrects 3D distortions in measured point clouds by fitting a 5th-order 3D Bernstein polynomial to map the distorted points to ground truth positions as described mathematically in Section 3.3. This fitting process creates the corrected points and the polynomial coefficients, along with a bounding box expanding by a 10% margin to account for spatial variability.

Outputs for problem 2 in PA2 were found in "**pa2_problem2.m**". The **coeffs** output contains the least-squares polynomial weights that model the systematic distortion. The **Cmin_margin** and

Cmax_margin represent the lower and upper boundaries of the normalized calibration volume, extending ±5%. These parameters are stored to preserve consistent normalization when applying the same distortion correction model to a new set of data, ensuring compatibility with the previously solved coefficient mapping.

Algorithm 5 Distortion Correction using 3D Bernstein Polynomial

```
 1: Input:
 2:     $C_{measured}$: $N \times 3$ array of distorted coordinates
 3:     $C_{ground}$: $N \times 3$ array of ground truth coordinates
 4: Output:
 5:     $C_{corrected}$, $coeffs$, $C_{min\_margin}$, $C_{max\_margin}$
 6:
 7: $n \leftarrow 5$                                          ▷ Order of Bernstein polynomial
 8: Compute $C_{min} \leftarrow \min(C_{measured})$, $C_{max} \leftarrow \max(C_{measured})$
 9: $range \leftarrow C_{max} - C_{min}$
10: $margin \leftarrow 0.05 \times range$
11: $C_{min\_margin} \leftarrow C_{min} - margin$
12: $C_{max\_margin} \leftarrow C_{max} + margin$
13:
14: Normalize input points:                                    ▷ Form equation 14
        $$C_{norm} = \frac{C_{measured} - C_{min\_margin}}{C_{max\_margin} - C_{min\_margin}}$$
15:
16: Extract coordinates $(u, v, w)$ from $C_{norm}$
17:
18: Build Bernstein basis matrix $F$ of size $N \times (n+1)^3$:    ▷ Solve equation 15
19: for $i = 0$ to $n$ do
20:     for $j = 0$ to $n$ do
21:         for $k = 0$ to $n$ do
22:             $F(:, col) \leftarrow B_i(u) \cdot B_j(v) \cdot B_k(w)$
23:             $col \leftarrow col + 1$
24:         end for
25:     end for
26: end for
27:
28: Solve least-squares mapping for each coordinate:
        $$coeffs = \text{lsqminnorm}(F, C_{ground})$$
29: Compute corrected points:
        $$C_{corrected} = F \cdot coeffs$$
30:
31: return $C_{corrected}$, $coeffs$, $C_{min\_margin}$, $C_{max\_margin}$
```

4.5 Pivot Calibration Algorithm

The function "**pivot_calibration.m**" estimates the fixed pivot location (dimple point) of the probe and the position of the probe tip in its local coordinate frame using the least-squares calibration approach described in Section 3.2.

The input F_G represents the measured marker positions of the probe across multiple frames. For each frame, the rotation and translation relative to the first frame are computed using the **svd** method described by Arun's method [2].

Algorithm 6 Compute Pivot Point and Tip Position via Least-Squares Calibration

```
1: **Input:** $F_G$ $(N \times 3 \times N_{\text{frames}})$ array of probe marker readings
2: **Output:** $\mathbf{t}_G$, $\mathbf{P}_{\text{dimple}}$
3:
4: Set reference frame $G_{\text{ref}} = F_G(:,:,1)$
5: Compute centroid $\bar{g}_{\text{ref}}$ and centered points $g_{\text{ref}} = G_{\text{ref}} - \bar{g}_{\text{ref}}$
6:
7: **for** $k = 1$ to $N_{\text{frames}}$ **do**
8:     Extract frame $G_k = F_G(:,:,k)$
9:     Compute centroid $\bar{g}_k$ and centered points $g_k = G_k - \bar{g}_k$
10:
11:     Compute cross-covariance $H = g_k^T g_{\text{ref}}$
12:     Compute $[U, \Sigma, V] = \text{svd}(H)$                          ▷ Equation 8
13:
14:     Compute rotation $R_k = VU^T$
15:     Compute translation $t_k = \bar{g}_{\text{ref}} - R_k \bar{g}_k$
16:     Append $[R_k \;\; -I]$ to $A$ and $-t_k$ to $b$                  ▷ Equation 12
17: **end for**
18:
19: Solve $Ax = b$ using lsqr
20: Extract $\mathbf{t}_G = x(1:3)$ and $\mathbf{P}_{\text{dimple}} = x(4:6)$
21: **return** $\mathbf{t}_G$, $\mathbf{P}_{\text{dimple}}$
```

4.6 Computing EM Markers Probe Post Coordinates with Distortion Correction

The function "**pa2_problem3.m**" performs pivot calibration using the EM tracker data after distortion correction. The input file contains the EM marker readings, and each frame of measured EM data is corrected using the Bernstein polynomial coefficients obtained from the distortion correction step in Section 4.4. The function "**apply_correction.m**" is a modified version of the "**distortion_correction.m**" function by entering the calculated coefficients, C_{min} and C_{max}, to apply to a new set of data and only outputting the corrected points.

The corrected dataset is then passed to the "**pivot_calibration.m**" to determine the fixed pivot point and probe tip position as described in Section 4.5.

Algorithm 7 EM Pivot Calibration After Distortion Correction

```
1: Input:
2:     $empivot\_file$: path to EM pivot data file
3:     $coeffs$: distortion correction coefficients
4:     $C_{min}, C_{max}$: calibration cube bounds for normalization
5: Output:
6:     $p_{em}$: $(1 \times 3)$ array of EM post pivot coordinates
7:     $t_G$: $(3 \times 1)$ probe tip position in probe frame
8:
9: Parse EM pivot data $F_G = \texttt{read\_em\_pivot\_data}(empivot\_file)$
10: Initialize $F_G^{corrected}$ of size $(N_{markers}, 3, N_{frames})$
11:
12: for each frame $f = 1$ to $N_{frames}$ do
13:     Extract marker set $points = F_G(:, :, f)$
14:     Apply distortion correction:
        $F_G^{corrected}(:, :, f) = \texttt{apply\_distortion\_correction}(points, coeffs, C_{min}, C_{max})$
15: end for
16:
17: Compute pivot calibration:
        $[p_{em}, t_G] = \texttt{pivot\_calibration}(F_G^{corrected})$
18: Format $p_{em}$ into cell array of $(x, y, z)$ strings
19: return $p_{em}, t_G$
```

4.7 Computing Locations in CT Coordinates with Distortion Correction

The functions "**pa2_problem4.m**", "**pa2_problem5.m**", and "**pa2_problem6.m**" compute the registration transformation between the CT and EM space using readings at each fiducial and then use that transformation to calculate any location of the EM pointer in CT coordinates.

4.7.1 Computing the CT Image Registration

The functions "**pa2_problem4.m**" and "**pa2_problem5.m**" calculate the transformation between the CT Image and EM tracker space using the readings from each of the fiducials. After applying distortion correction to the EM tracker data, the probe tip location is computed in the EM frame using the approach described in 3.4. With both the EM probe tip and CT image location, point cloud registration is used to determine the homogenous transformation between the two.

Algorithm 8 Registration Between EM Tracker and CT Image

```
1: **Input:** $emfiducial$, $ctfiducial$, $G_{ref}$, $coeffs$, $C_{min}, C_{max}$
2: **Output:** $rotationMatrix$ (3x3), $translationVector$ (3x1)
3:
4: **Extract EM marker readings for each fiducial:**
5: $(G_j, N_G, N_B) \leftarrow \textsc{ReadEMFiducialData}(emfiducial\_file)$
6: Initialize $G_{j,flat} \in R^{N_G N_B \times 3}$
7: **for** $i = 1$ to $N_B$ **do**
8:     $G_{j,flat}[(i-1)N_G + 1 : iN_G, :] \leftarrow G_j[:, :, i]$
9: **end for**
10:
11: **Apply distortion correction to EM readings:**
12: $G_{j,corrected} \leftarrow \textsc{ApplyDistortionCorrection}(G_{j,flat}, coeffs, C_{min}, C_{max})$
13:
14: **Compute probe tip positions in EM frame:**
15: Initialize $B_j \in R^{N_B \times 3}$
16: **for** $i = 1$ to $N_B$ **do**
17:     $G_{frame} \leftarrow G_{j,corrected}[(i-1)N_G + 1 : iN_G, :]$
18:     $(R, t) \leftarrow \textsc{PointCloudRegistration}(G_{ref}, G_{frame})$
19:     $tform_G \leftarrow \textsc{RigidTform3D}(R, t)$
20:     $B_j[i, :] \leftarrow \textsc{TransformPointsForward}(tform_G, t'_G)$
21: **end for**
22:
23: **Read fiducial positions in CT frame:**
24: $b_j \leftarrow \textsc{ReadCTFiducialData}(ctfiducial\_file)$
25:
26: **Compute registration between EM tracker and CT image:**
27: $(R_{reg}, t_{reg}) \leftarrow \textsc{PointCloudRegistration}(B_j, b_j)$
28:
29: **return** $(R_{reg}, t_{reg})$
```

4.7.2 Computing the Locations in CT Coordinates

Once the registration transformation is complete, it can be used to compute the location of the EM probe at any point in the CT image. Generalizing this process, the data from the EM probe is corrected for distortion and the location of the probe tip in the EM space is calculated. Then the registration transformation is used to determine the corresponding location coordinates in the CT image.

Algorithm 9 EM Probe Tip Location in CT Frame

```
1: **Input:** $emnav$, $coeffs$, $C_{min}$, $C_{max}$, $R_{reg}$, $t_{reg}$
2: **Output:** $v_i$, $V_i$, $N_{frames}$
3:
4: **Extract EM marker readings for each frame:**
5: $(G_j, N_G, N_{frames}) \leftarrow$ ReadEMFiducialData($emnav\_file$)
6: Initialize $G_{j,flat} \in R^{N_G N_{frames} \times 3}$
7: **for** $i = 1$ to $N_{frames}$ **do**
8:     $G_{j,flat}[(i-1)N_G + 1 : iN_G, :] \leftarrow G_j[:, :, i]$
9: **end for**
10:
11: **Apply distortion correction:**
12: $G_{j,corrected} \leftarrow$ ApplyDistortionCorrection($G_{j,flat}, coeffs, C_{min}, C_{max}$)
13:
14: **Initialize outputs:**
15: $V_i, v_i \leftarrow$ empty arrays of size $N_{frames} \times 3$
16: $tform_{reg} \leftarrow$ RigidTform3D($R_{reg}, t_{reg}$)
17:
18: **Compute probe tip positions:**
19: **for** $i = 1$ to $N_{frames}$ **do**
20:     $G_{frame} \leftarrow G_{j,corrected}[(i-1)N_G + 1 : iN_G, :]$
21:     $(R, t) \leftarrow$ PointCloudRegistration($G_{ref}, G_{frame}$)
22:     $tform_G \leftarrow$ RigidTform3D($R, t$)
23:     $V \leftarrow$ TransformPointsForward($tform_G, t'_G$)                    ▷ Tip in EM frame
24:     $V_i[i, :] \leftarrow V$
25:     $v_i[i, :] \leftarrow$ Round(TransformPointsForward($tform_{reg}, V$), 2)    ▷ Tip in CT frame
26: **end for**
27:
28: **return** $(v_i, V_i, N_{frames})$
```

5 Overview of Program Structure

The algorithms described in Section 4 are summarized below. A complete overview of the program structure is provided to illustrate the organization of all scripts, helper functions, and unit tests within the project directory. The corresponding folder hierarchy and locations of the `.m` files are shown in Figure 4.

Each function's inputs and outputs have been explicitly defined and commented within the MATLAB source code, and are also summarized in the pseudocode provided in Section 4. This section therefore focuses on the structural relationships between scripts and how the functions, data processing routines, and test classes interact within the overall workflow.

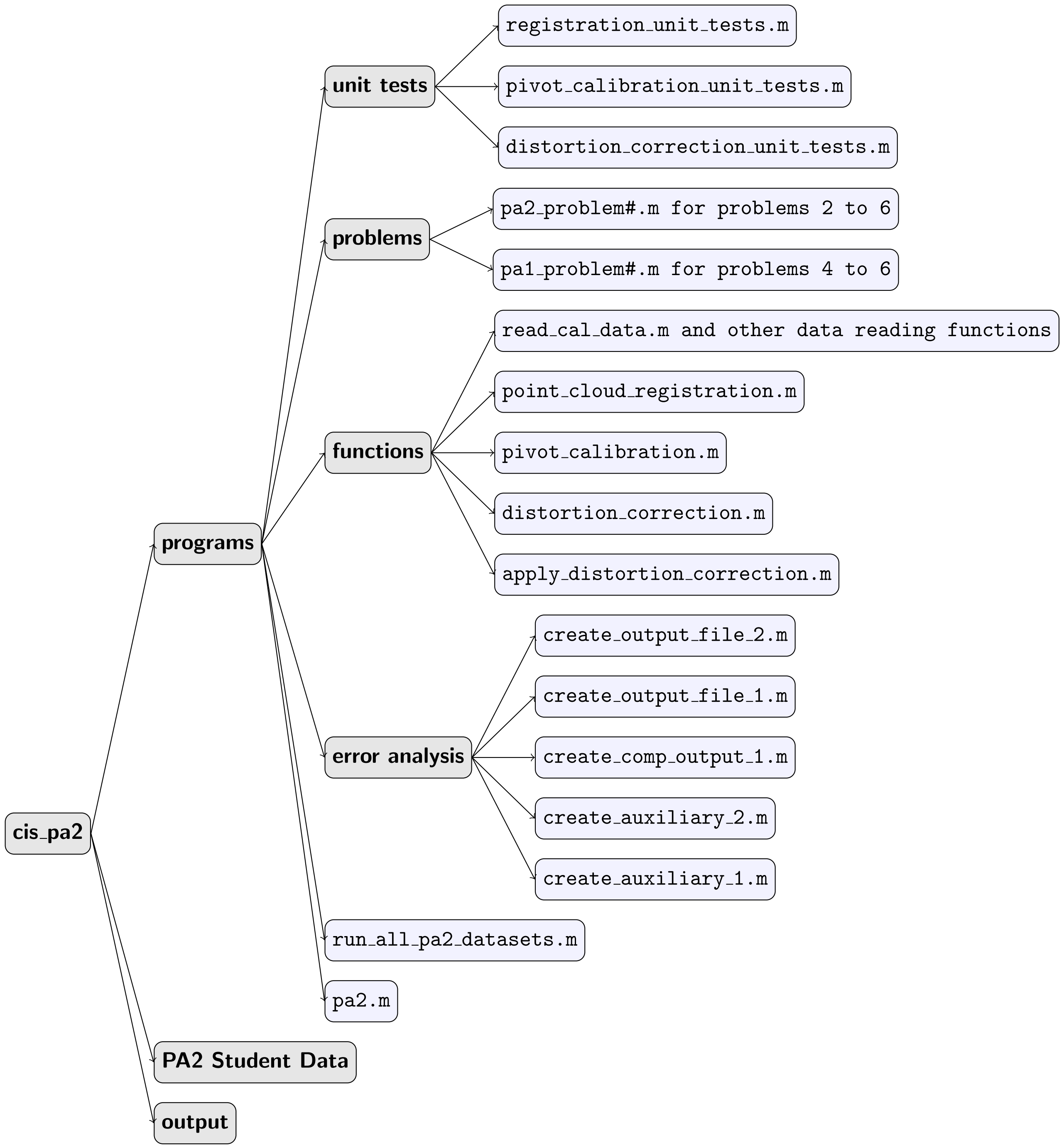
cis_pa2
programs
unit tests
registration_unit_tests.m
pivot_calibration_unit_tests.m
distortion_correction_unit_tests.m
problems
pa2_problem#.m for problems 2 to 6
pa1_problem#.m for problems 4 to 6
functions
read_cal_data.m and other data reading functions
point_cloud_registration.m
pivot_calibration.m
distortion_correction.m
apply_distortion_correction.m
error analysis
create_output_file_2.m
create_output_file_1.m
create_comp_output_1.m
create_auxiliary_2.m
create_auxiliary_1.m
run_all_pa2_datasets.m
pa2.m
PA2 Student Data
output

Figure 4: Program hierarchy showing folders, functions, and scripts in `cis_pa2`.

As described in the **README.md** file, navigate to the `programs` folder and run the following code for each dataset in PA2:

```
pa2({type},{letter});
```

Replace {`type`} with `debug` or `unknown` based on the type of data and {`letter`} with the letter of the dataset (e.g., `pa2('debug','a');`).

To run all the code for the given datasets at once, use the following code. The `run_all_pa2_datasets.m` file executes PA2 for all debug and unknown datasets.

```
cd programs/unit tests/
run_all_pa2_datasets.m;
```

Each function's inputs and outputs have been explicitly defined and commented within the MATLAB source code, and are also summarized in the pseudocode provided in Section 4.

Programs Order:

1. Corrected 3D points after distortion correction
2. Point cloud registration outputs aligning EM and CT frames
3. Estimated pivot calibration results for each dataset
4. Error analysis plots summarizing deviations

All outputs are saved to the `output/` folder for further review.

Figure 5 illustrates the main program workflow, showing how the primary scripts call supporting functions and how data flows through the PA2 pipeline. This flow chart complements the hierarchy chart in Figure 4 by making the functional relationships between scripts, helper functions, and analysis routines explicit. This is a simplified flowchart to show the general flow of our program. In practice, it is more circular in calling our three main functions to solve the problems and may not simply go directly into the outputs, depending on the problem step.

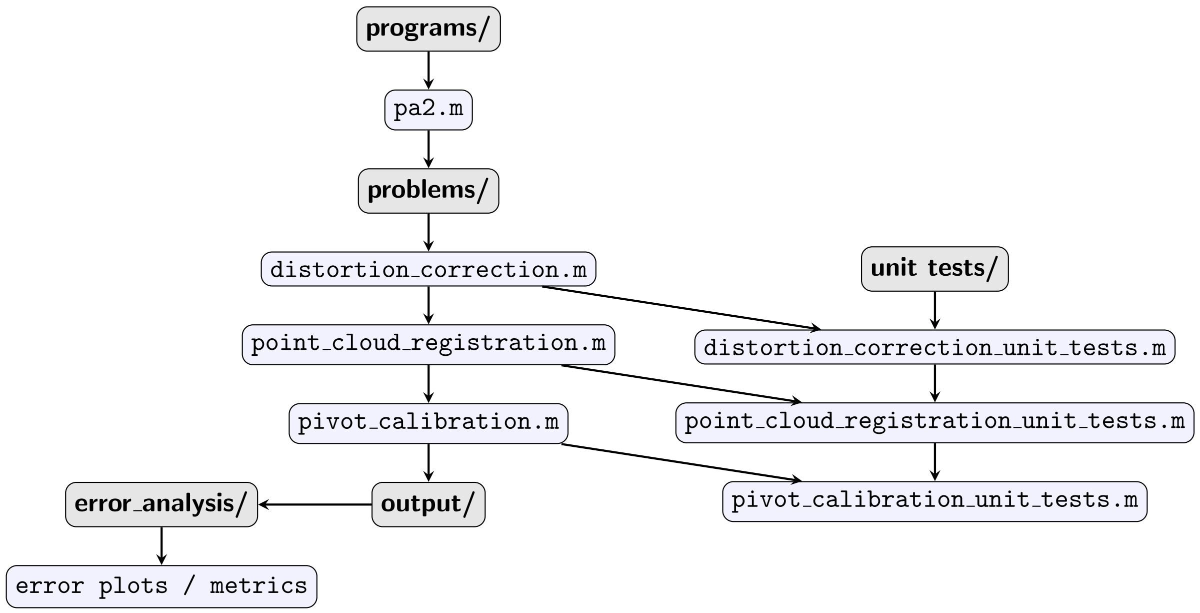
programs/
pa2.m
problems/
distortion_correction.m
point_cloud_registration.m
pivot_calibration.m
output/
error_analysis/
error plots / metrics
unit tests/
distortion_correction_unit_tests.m
point_cloud_registration_unit_tests.m
pivot_calibration_unit_tests.m

Figure 5: PA2 program flow showing how `pa2.m` calls the problems folder and functions, with unit tests and outputs connected.

6 Validation Approach

Three main unit tests were created for testing and validating the point cloud registration, point set registration, and distortion correction algorithms.

6.1 Point Cloud Registration Unit Tests

The unit tests used to verify the functionality of the point cloud registration algorithm described in Section 3.1 is called **"registration_unit_tests.m"**, located in the unit tests folder of the program files.

This unit test class evaluates the functionality and accuracy of the point cloud registration function. It does so by defining a ground truth dataset consisting of a set of marker locations in the calibration object frame and applying a random transformation to get the marker locations in the tracker object frame. The unit test generates a synthetic dataset that has multiple frames of markers locations in the tracker object frame, each with their own associated random transformation.

The subclass of "**matlab.unittest.TestCase**" implements the test class to generate a synthetic, noise-free test dataset. A set of transformation matrices, using roll pitch and yaw for rotations and randomized translation, is stored as the known ground truth values. The test class then creates an initial set of 3D marker coordinates (x_i) within the calibration frame and applies the known transformation matrix to produce the corresponding transformed coordinates (X_i) in the tracker frame.

The same test case data for x_i and X_i is fed into the "**point_cloud_registration**" function for comparison to the known ground truth transformations. The difference of the the estimated rotations and translations from the "**point_cloud_registration**" to the known test case rotations and translations is calculated and the norm is found using the norm function. Note that using the norm function utilizes the Euclidean norm [11]. The unit test will also detect errors in the transformation matrix outside a tolerance of 1×10^{-6}.

This unit test class tests that the "**point_cloud_registration**" function returns accurate transformations for both a single frame and multiple frames passed into the function. It also tests for the minimum number of markers required to calculate an accurate transformation between the marker and tracker data. Running this test yielded that a minimum of 3 markers are required to calculate a transformation matrix within the defined accuracy bounds.

6.2 Pivot Calibration Unit Tests

To validate the accuracy and robustness of the implemented pivot calibration algorithm, a unit test class was developed, **"pivot_calibration_unit_tests.m"**. This test ensures that the pivot calibration algorithm can accurately compute the pivot point in the tracker coordinate system, given multiple frames of data of the tool rotated around a fixed pivot. The estimated pivot must coincide with the known ground truth pivot within a numerical tolerance.

Synthetic data is created by forming six randomly positioned markers to define the tool geometry in the tool frame. The centroid of the markers is shifted to the origin so that the pivot lies at [0, 0, 0] in the tool frame. A random pivot point is assigned in the tracker frame. The tool is then rotated through multiple poses using small random roll, pitch and yaw angles. For each pose, the rotation is applied to the markers, and a translation equal to the pivot point ensures that the tip remains fixed. Therefore, the marker positions in the tracker frame are as follows:

$$X_{ij} = R_i x_j + t_i \tag{27}$$

The testing procedure is as follows:

Algorithm 10 Pivot Calibration Unit Test

```
1: for trial = 1 to $N_{\text{trials}}$ do
2:     Generate $N_{\text{markers}}$ random points $x_i$ in the tool frame
3:     Shift $x_i$ so that pivot = $[0, 0, 0]$ in the tool frame
4:     Assign random pivot $p_{\text{tip}}$ in the tracker frame
5:     for frame = 1 to $N_{\text{frames}}$ do
6:         Generate small random rotation $R_i$
7:         Set translation $t_i = p_{\text{tip}}$
8:         Compute tracker frame positions: $X_i = R_i \cdot x_i + t_i$
9:     end for
10:    Call pivot_calibration($X_i$) $\rightarrow p_{\text{est}}$
11:    Compute error: error = $\|p_{\text{est}} - p_{\text{tip}}\|_2$
12:    if error $\geq 10^{-6}$ then
13:        Flag test as failed
14:    else
15:        Mark test as passed
16:    end if
17: end for
```

The Euclidean norm is found between the estimated and ground truth pivot tip to quantify the accuracy. As summarized below, the algorithm is numerically precise and very stable, where all five random trials demonstrated reproduction of the correct pivot point with very small errors.

Table 1: Pivot Calibration Unit Test Results

Trial	**Ground Truth Pivot (mm)**			**Estimated Pivot (mm)**			**Norm Error**
1	303.4329	144.7169	198.4900	303.4329	144.7169	198.4900	1.083e-12
2	196.7164	256.3732	162.0065	196.7164	256.3732	162.0065	1.627e-12
3	117.1845	220.9087	296.0509	117.1845	220.9087	296.0509	2.242e-12
4	7.5356	105.3807	399.4359	7.5356	105.3807	399.4359	1.545e-12
5	54.3939	156.2039	183.0222	54.3939	156.2039	183.0222	2.296e-13

6.3 Distortion Correction Unit Tests

To validate the accuracy and robustness of the implemented distortion correction algorithm, a unit test class was developed, **"distortion_correction_unit_tests.m"**. The test ensures accurate mapping of distorted datasets to their ground truth positions and that the correction is numerically stable under small perturbations or edge-case scenarios.

Synthetic data was used to create randomized ground truth values for the 3D point cloud of 50 points within a bounding box. Systematic distortions was simulated by applying a combination of nonlinear transformations, random scaling and Gaussian noise as defined below:

```
% create synthetic, systematic EM distortion
% (nonlinear on each coordinate point)
scale = 1 + 0.05*randn(size(testCase.C_ground)); % random scaling
noise = 0.02*randn(size(testCase.C_ground));       % measurement noise
nonlinear = @(x) x + 0.1 * (sin(x/10) + cos(x/20)); % nonlinear EM
    distortion
```

The testing procedure is as follows:

Algorithm 11 Distortion Correction Unit Tests

```
1: for trial = 1 to $N_{\text{trials}}$ do
2:     Generate ground truth points
3:     Generate $N_{\text{points}}$ random 3D points $C_{\text{ground}}$ in bounding box
4:
5:     Apply synthetic EM distortions
6:     Create $C_{\text{measured}}$ by adding:
7:         nonlinear distortion, small random scaling, and Gaussian noise
8:
9:     Apply distortion correction
10:    $[C_{\text{corrected}}, \text{coeffs}, \_, \_] = \text{distortion\_correction}(C_{\text{measured}}, C_{\text{ground}})$
11:
12:    Verify correction accuracy
13:    $\text{maxError} = \max(\text{abs}(C_{\text{corrected}} - C_{\text{ground}}))$
14:    Ensure $\text{maxError} < 10^{-3}$
15:
16:    Verify polynomial coefficient size
17:    $\text{expected\_size} = [(n+1)^3, 3]$    ▷ n = polynomial order (5)
18:    Ensure size(coeffs) == expected_size
19:
20:    Test numerical stability under small perturbations
21:    $\text{perturbation} = 10^{-6} \cdot$ random noise    ▷ Define perturbation added to measurements
22:    $C_{\text{perturbed}} = C_{\text{measured}} +$ perturbation
23:    $C_{\text{corrected\_perturbed}} = \text{distortion\_correction}(C_{\text{perturbed}}, C_{\text{ground}})$    ▷ Resolve with perturbations
24:    $\text{diffError} = \max(\text{abs}(C_{\text{corrected}} - C_{\text{corrected\_perturbed}}))$
25:    Ensure $\text{diffError} < 10^{-6}$
26:
27:    Test correction of bounding box edge points
28:    Generate $C_{\text{ground\_edge}}$ using min and max coordinates
29:    Apply synthetic distortion $\rightarrow C_{\text{measured\_edge}}$
30:    $C_{\text{corrected\_edge}} = \text{distortion\_correction}(C_{\text{measured\_edge}}, C_{\text{ground\_edge}})$    ▷ Resolve with edge cases
31:    $\text{edgeDiffError} = \max(\text{abs}(C_{\text{corrected\_edge}} - C_{\text{ground\_edge}}))$
32:    Ensure $\text{edgeDiffError} < 10^{-3}$
33: end for
```

Passing all tests demonstrates that the distortion correction algorithm accurately recovers ground truth points from distorted measurements, is robust to small perturbations, and handles edge points of the bounding box correctly. In the terminal, all unit tests were passed with "Success" for all 10 iterations that we looped through when running this code.

7 Results

7.1 Debug Dataset

The following table shows the resultant estimated positions of the probe top with respect to the EM and CT coordinates.

Table 2: Estimated Positions (EM and CT) for Datasets a–f

Dataset	Nav Point	Est EM (X,Y,Z)			Est CT (X,Y,Z)		
a	1	409.85	406.95	313.95	104.85	107.53	57.87
	2	404.23	427.11	377.26	99.23	128.33	120.99
	3	392.84	371.87	415.40	87.84	73.47	159.67
	4	438.10	375.57	293.61	133.11	75.95	37.85
b	1	374.33	386.08	376.93	74.32	86.88	76.03
	2	397.98	378.00	345.96	97.93	78.41	45.13
	3	362.11	461.40	437.65	62.33	162.86	135.96
	4	335.63	442.62	454.38	35.81	144.33	152.90
c	1	399.60	304.29	348.52	74.66	27.81	49.25
	2	351.68	406.14	432.94	27.96	130.60	133.19
	3	349.55	363.69	389.49	25.41	88.20	89.73
	4	485.04	385.76	424.79	160.96	107.56	126.36
d	1	473.05	496.88	468.49	172.06	97.86	49.99
	2	356.53	473.36	485.17	55.22	74.65	64.80
	3	470.01	558.73	535.22	167.70	161.03	115.40
	4	456.72	466.89	497.11	155.15	68.44	78.87
e	1	366.44	379.52	464.51	118.63	76.05	114.60
	2	309.80	386.00	449.68	62.24	84.32	99.71
	3	330.48	401.14	454.83	83.39	98.80	104.87
	4	406.67	422.53	499.31	160.17	117.79	149.41
f	1	500.81	449.62	363.34	150.23	151.46	60.39
	2	503.22	361.35	390.77	152.28	64.07	90.53
	3	375.77	411.23	340.41	25.46	112.39	37.17
	4	436.79	425.65	417.09	85.57	129.14	114.09

7.2 Unknown Dataset

The output files for unknown data h, i, j, and k are summarized in the table below. All data was extracted from output files name ”pa2-unknown-x-auxiliary-2.txt”.

Note that as according to the PA2 document, database h and i featured all three types of noise, including EM distortion, EM noise and OT jiggle [1]. The effects will be discussed in Section 7.3.

Table 3: Estimated Positions (EM and CT) for Datasets g–j

Dataset	**Nav Point**	**Est EM (mm)**			**Est CT (mm)**		
		X	Y	Z	X	Y	Z
g	1	446.07	434.83	398.28	90.41	83.02	40.19
	2	400.59	426.74	431.99	44.96	74.83	73.92
	3	407.97	470.12	468.97	52.77	117.66	111.45
	4	390.26	416.80	423.43	34.53	65.09	65.25
h	1	490.95	387.69	486.07	145.17	39.15	134.73
	2	414.99	400.52	508.62	69.92	53.60	158.65
	3	395.05	492.85	461.73	50.88	146.04	111.62
	4	439.40	389.39	501.97	93.97	41.95	151.60
i	1	399.07	477.70	448.31	46.46	124.57	90.15
	2	451.94	478.59	391.01	98.75	125.05	32.32
	3	461.49	497.97	406.37	108.44	144.57	47.42
	4	462.40	422.99	509.63	110.47	70.43	151.27
j	1	414.04	410.06	440.54	64.07	53.73	88.92
	2	417.42	524.23	501.76	67.84	168.50	148.97
	3	503.32	461.48	452.23	153.41	105.33	99.39
	4	513.09	507.97	499.87	163.51	152.31	146.49

7.3 Error Analysis

Auxiliary data files are created for each debug dataset as part of the output folder, named "pa2-debug-x-auxiliary1.txt" and "pa2-debug-x-auxiliary2.txt" for x files a to g, and "pa2-unknown-x-auxiliary1.txt" and "pa2-unknown-x-auxiliary2.txt" for x files h to k. The function "create_auxiliary_1.m", located in the error analysis folder, compares the calculated EM pivot post coordinates, optical pivot post coordinates, and $C_{expected}$ values to the actual dataset values. The function "create_auxiliary_2.m", also located in the error analysis folder, compares the navigation points in the EM-space and CT-space coordinates to the values in the provided output. The output files are formatted similarly to the ones provided in the program template.

7.3.1 Errors in C_i from Debug Data

This section will focus on debug files b, c and d. As summarized in the PA document, dataset b has EM noise, database c has EM distortion, and database d has OT jiggle. These datasets will allow isolated analysis of the effects of each type of error.

EM Noise represents random fluctuations present along each individual measurement along each coordinate axis. EM distortion refers to a systematic error that varies with position relative to the EM tracking device. The OT jiggle corresponds to small physical movements or vibrations of the optical tracking base unit that occur between successive data frames. The table below summarizes the differences due to different types of error.

Table 4: Summary of database b, c and d error effects on $C_{corrected}$

Dataset	**Mean Difference**	**Norm Mean Difference**	**Norm Max Difference**
A - None	0, 0, 0	0	0.02
B - EM Noise	0, 0, 0	0.46	0.85
C - EM Distortion	0, 0, 0	0.03	0.11
D - OT Jiggle	0, 0, 0	0.01	0.03
E - All	0, 0, 0	0.09	0.48
F - All	0, 0, 0	0.20	0.55
G - Unknown	0, 0, 0	0.05	0.24
H - Unknown	0, 0, 0	0.20	0.52
I - Unknown	0, 0, 0	0.19	0.51
J - Unknown	0, 0, 0	0.20	0.49

By analyzing datasets b, c and d, the team is able to isolate and analyze each type of error and its effects on deviating the expected and actual values of C. Especially the EM noise on dataset b created significant random deviations as visualized below.

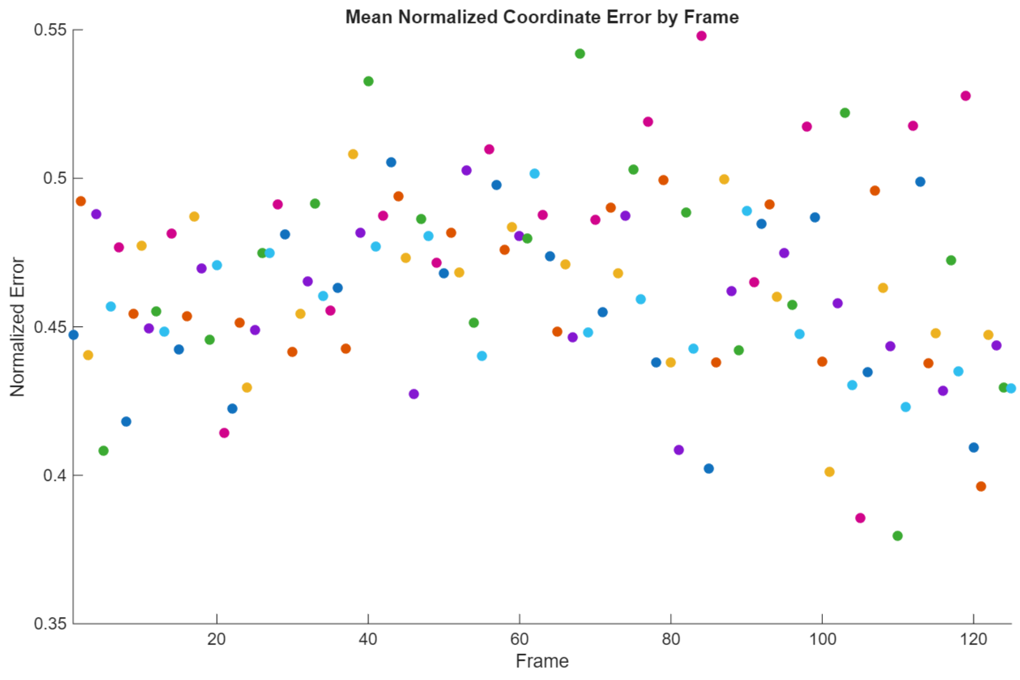

Figure 6: Normalized Error by Marker and Frame for Dataset B

Since Table 4 evaluates $C_{corrected}$, the mean difference for each coordinate is zero. This indicates that the distortion correction procedure accurately corrected the EM tracker readings based on the ground truth readings from the optical tracker. The largest norm difference occurs in dataset b, which has EM noise. This indicates that the distortion correction procedure works well for a set skewing of the data from systematic errors, but has difficulties with just random noise. Future steps to increase the accuracy of the distortion correction function could involve some type of filtering of the data to remove specific types of noises for different distortion correction sources.

7.3.2 Errors in v_i from Debug Data

The following table shows the errors between the calculated v_i values and the provided debug files. With the addition of all three types of errors in dataset e and f, larger skews in the final results are observed. Future steps would require closer distortion correction for random noise errors when accumulated on top of the systematic errors.

Table 5: CT-Space Errors for Datasets a–f

Dataset	**Mean Error (X,Y,Z) [mm]**			**Mean Norm Error [mm]**	**Max Norm Error [mm]**
a	-0.00	0.00	0.00	0.01	0.02
b	0.03	0.03	0.07	0.12	0.16
c	0.50	-0.13	-0.05	0.56	0.71
d	0.01	0.01	0.00	0.01	0.02
e	-1.17	-1.41	-0.30	1.94	2.08
f	1.20	0.24	-0.86	1.63	2.11

8 Work Distribution

The distribution of the work was shared between both team members. Shreya and Yvonne worked together on the mathematical approach and understanding the setup.

Shreya focused on functions used to read data files, the point cloud registration function, unit testing files, and computing the CT image registration frame. Yvonne focused on the mathematical approaches, the distortion correction algorithm, the point set registration algorithm, the pivot calibration algorithm, and translating the team's work into the report.

References

[1] R. Taylor, “Programming assignment 1 and 2,” 2025. Accessed: 2025-09-18.

[2] J. Shi, “Arun’s method for 3d registration,” 2018. Accessed: 2025-09-21.

[3] O. Sorkine-Hornung and M. Rabinovich, “Least-squares rigid motion using svd.” Technical Report, ETH Zurich, 2017. Accessed: 2025-09-21.

[4] G. Fischer and R. Taylor, “Electromagnetic tracker measurement error simulation and tool design,” 02 2005.

[5] R. H. Taylor, “Interpolation and deformations: A short cookbook,” Tech. Rep. 600.445—Fall 2000, Engineering Research Center for Computer Integrated Surgical Systems and Technology, Johns Hopkins University, 2000. Updated: 17 September 2025.

[6] The MathWorks, Inc., “Linear algebra - symbolic math toolbox documentation,” 2025. Accessed: 2025-10-03.

[7] The MathWorks, Inc., “svd - singular value decomposition,” 2025. Accessed: 2025-10-03.

[8] G. Gundersen, “A non-technical introduction to svd,” 2018. Accessed: 2025-10-03.

[9] The MathWorks, Inc., “lsqr - solve system of linear equations: least-squares method,” 2025. Accessed: 2025-10-01.

[10] The MathWorks, Inc., *lsqminnorm — Minimum norm least-squares solution to linear equation*. The MathWorks, Inc., 2025. Online; accessed 1 November 2025.

[11] The MathWorks, Inc., “norm - vector and matrix norms,” 2025. Accessed: 2025-10-06.