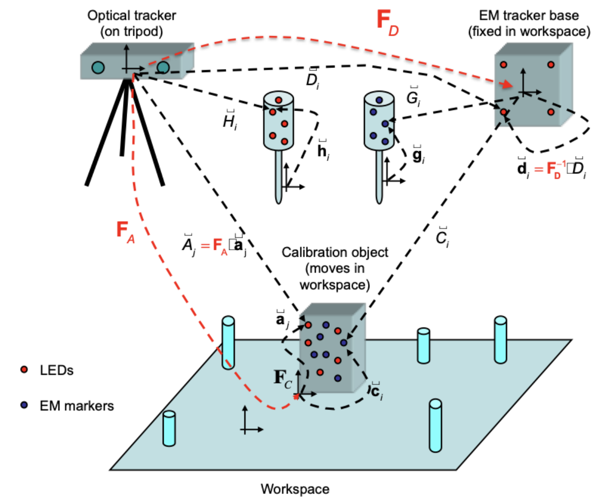
Overview:

Description of the Problem Scenario

This problem concerns calibration, simple registration, and tracking for a stereotactic navigation system that uses an electromagnetic positional tracking device.


Optical tracker (on tripod)
$\mathbf{F}_D$
EM tracker base (fixed in workspace)
$\vec{D}_i$
$\vec{G}_i$
$\vec{H}_i$
$\vec{h}_i$
$\vec{g}_i$
$\vec{d}_i = \mathbf{F}_D^{-1} \cdot \vec{D}_i$
$\mathbf{F}_A$
$\vec{A}_j = \mathbf{F}_A \cdot \vec{a}_j$
$\vec{C}_i$
Calibration object (moves in workspace)
$\vec{a}_j$
$\mathbf{F}_C$
$\vec{c}_i$
LEDs
EM markers
Workspace

Problem Scenario (what & why): We are building a stereotactic navigation pipeline that reports the 3D position of an EM-tracked probe tip in CT coordinates during surgery. The optical tracking provides accurate marker positions (assumed geometry-error free) while the EM tracker suffers repeatable, systematic distortion. Therefore, we are trying to learn and correct the EM distortion so EM measurements can be used for accurate navigation. The calibration object, instrumented with known EM and optical markers, lets us relate frames and generate the undistorted "expected" positions.

Methods in context: For each calibration frame k, we estimate the transforms, FD (k) (optical to the EM base) and FA(k) (cal-object to optical) to predict undistorted EM positions of the object's EM markers. Paired with measured EM data, these train a 3D polynomial distortion model. We then recompute the EM probe pivot calibration (tip in probe frame) using corrected EM data. Touching image fiducials gives us EM space tip positions Bj which we register to CT fiducials bj to obtain F reg (EM to CT). During navigation, we unwarp the raw EM probe markers, recover the tip in EM space, and apply F reg to report the tip in CT space.

Mathematical approach:

3D Point Set to Point Set Registration

This procedure finds the rigid transform that best aligns two matched point sets by least squares. You center both sets, form the 3×3 cross-covariance H, take its SVD, and build $R=V U^T$ with a reflection fix if $\det(R)<0$; the translation is $t=c\_B-R\ c\_A$. In edge cases where H is rank, deficient (collinear/coplanar or a zero-singular value) or H is approximately zero (indicating an unstable det(R) situation) the solution is unstable and the registration is rejected or regularized.

Given corresponding 3D point sets A = {a_i} and B = {b_i} for i = 1..n, estimates a rigid transform (R, t) that maps points in A to points in B by minimizing the total squared error between R * a_i + t and b_i. Here, R is a 3x3 rotation matrix and t is a 3x1 translation vector. The rotation must satisfy R^T * R = I and det(R) = 1.

Definitions:

- a_i, b_i: corresponding 3D points (column vectors).
- c_A = (1/n) * sum over i of a_i (centroid of A).
- c_B = (1/n) * sum over i of b_i (centroid of B).
- a_i' = a_i - c_A (centered A points).
- b_i' = b_i - c_B (centered B points).
- H = sum over i of (a_i') * (b_i')^T (3x3 cross-covariance matrix).

Closed form solution (Arun/Kabsch method):

1. Compute centroids c_A and c_B.
2. Center the data: a_i' = a_i - c_A, b_i' = b_i - c_B.
3. Form H = sum_i (a_i') * (b_i')^T.
4. Compute the singular value decomposition (SVD) of H:
   H = U * Sigma * V^T.
5. Compute the rotation:
   R = V * U^T.
6. Reflection correction:
   If det(R) < 0, negate the last column of V to get V', then recompute
   R = V' * U^T
   to enforce det(R) = 1.
7. Compute the translation:
   t = c_B - R * c_A.

Corner / degenerate cases: If rank(H) < 2 (e.g., collinear points), the problem is not posed properly. If rank(H) = 2 (all points coplanar), the solution can be ambiguous; handle reflections carefully and reject near-singular cases as indicated by very small singular values.

Citations:

- [Arun1987] Arun, K. S., Huang, T. S., & Blostein, S. D. (1987). Least-squares fitting of two 3-D point sets. IEEE Transactions on Pattern Analysis and Machine Intelligence, 9(5), 698–700.

- [Horn1987] Horn, B. K. P. (1987). Closed-form solution of absolute orientation using unit quaternions. Journal of the Optical Society of America A, 4(4), 629–642.

- [Umeyama1991] Umeyama, S. (1991). Least-squares estimation of transformation parameters between two point patterns. IEEE Transactions on Pattern Analysis and Machine Intelligence, 13(4), 376–380.

- [GolubVanLoan2013] Golub, G. H., & Van Loan, C. F. (2013). Matrix Computations (4th ed.). Johns Hopkins University Press. (SVD/LS background)

Pivot Calibration

The pivot calibration solves for the probe's fixed tip (in probe coordinates) and the pivot point (in world/EM coordinates) by stacking linear equations over many frames. The block system A x=b is overdetermined and solved by least squares, yielding p_tip and p_pivot; an RMS residual quantifies fit quality. Intuitively, as we rotate the probe about a physical dimple, every pose should map the local tip to the same world point. If A^TA is near singular or rank deficient (indicating poorly conditioned poses), the solution is rejected or regularized and additional, more diverse pivot poses are required.

For each frame k, with probe pose (R_k, t_k) and tip in the local probe frame p_tip, the tip in world (tracker) frame is:
R_k * p_tip + t_k = p_pivot

This holds for all frames. Rearranging:
R_k * p_tip - p_pivot = -t_k

Stacking for all N frames (column form):
[R_1 -I] * [p_tip] = [-t_1]
[R_2 -I] [p_pivot] [-t_2]
[. .] .
[R_N -I] .
[-t_N]

In matrix form:
A * x = b
where A is (3N x 6), x is (6 x 1), and b is (3N x 1).

Solve via least squares:
$x = (A^T * A)^{(-1)} * A^T * b$

The solution vector x contains:

- $x[0:3] = p_{tip}$ (local frame)
- $x[3:6] = p_{pivot}$ (world frame)

RMS error:
$RMS = \sqrt{ (1/N) * \text{sum over } k \text{ of } \| R_k * p_{tip} + t_k - p_{pivot} \|^2 }$

Citations:

- Taylor, R. H. (2025). CIS 601/655 Lecture Slides: Rigid registration and pivot calibration. Johns Hopkins University. (
- Fitzpatrick, J. M., West, J. B., & Maurer, C. R. (2001). Predicting error in rigid-body point-based registration. IEEE Transactions on Medical Imaging, 20(9), 915–927. (Standard reference on rigid tracking error modeling; cited alongside pivot calibration procedures)

- [GolubVanLoan2013]
  Matrix least-squares background for the normal equations and lstsq.

Expected C Value Computation (PA1 Q4)

We are estimating where each EM-tracked C marker should appear by chaining frames, calibration-object to optical and optical to EM (via the D registration inverse). Concretely, $C_{expected}^k = (F_D^k)^{-1} \circ F_A^k$ is applied to each c_i. These expected positions pair with measured EM positions to train the distortion model. If forming F_D or F_A yields a covariance H whose smallest singular value is approximately zero, that frame's registration is rejected or regularized and the expected C positions are recomputed using only non degenerate frames.

For each calibration frame k:

Compute F_D using D markers (calibration object -> EM tracker):
$F_D^k = \text{register\_points}(d, D_k)$

Compute F_A using A markers (calibration object -> optical tracker):
$F_A^k = \text{register\_points}(a, A_k)$

Compute expected C:
$C_{expected}^k = (F_D^k)^{(-1)}$ applied to (F_A^k applied to c_i)

where c_i are C points in calibration object coordinates. This performs:
calibration object -> optical tracker -> EM tracker.

In homogeneous coordinates:
C_expected^k = (F_D^k)^(-1) * F_A^k * c_i

Distortion Correction Using Bernstein Polynomials

The EM field distortion is modeled as a smooth 3D Bernstein polynomial mapping measured points to their expected (undistorted) locations. After normalizing coordinates to [0,1], we build a design matrix of basis evaluations and solve a least squares system for coefficients. The resulting correction_fn evaluates the polynomial to undistort new measurements. Additionally, degree N controls flexibility versus overfitting; (N+1)^3 coefficients per axis are learned. If the design matrix M is ill conditioned or rank- deficient (such as with clustered samples or overly high degree), the fit is regularized or the polynomial degree is reduced before producing the correction function.

1D Bernstein basis:
B_{N,i}(u) = C(N,i) * u^i * (1 - u)^(N - i)
for i = 0, 1, ..., N and u in [0, 1].

3D Bernstein basis:
B_{N,i,j,k}(u_x, u_y, u_z) = B_{N,i}(u_x) * B_{N,j}(u_y) * B_{N,k}(u_z)

Correction function:
f(p_measured) = [f_x(p), f_y(p), f_z(p)]^T

Each component (example for x):
f_x(p) = sum over i=0..N, j=0..N, k=0..N of c_{x,i,j,k} * B_{N,i}(u_x) * B_{N,j}(u_y) * B_{N,k}(u_z)

Normalized coordinates:
u_x = (x - x_min) / (x_max - x_min)
u_y = (y - y_min) / (y_max - y_min)
u_z = (z - z_min) / (z_max - z_min)

For degree N, there are (N+1)^3 coefficients per axis.

Fitting via least squares:
min over c of sum over m=1..M of || f(p_m) - p_expected,m ||^2

Matrix form:
M * c = p_expected

- M is (M x (N+1)^3) design matrix of Bernstein basis evaluations
- c is ((N+1)^3 x 3) coefficient matrix

- p_expected is (M x 3) expected points

Solution:
c = (M^T * M)^(-1) * M^T * p_expected

Citations:

- Farin, G. (2002). Curves and Surfaces for CAGD: A Practical Guide (5th ed.). Morgan Kaufmann. (Bernstein basis and tensor-product construction used in a_distortion _calibration.py)

- Piegl, L., & Tiller, W. (1997). The NURBS Book (2nd ed.). Springer. (Alternative canonical reference for Bernstein/NURBS foundations)

Frame Transformations

Rigid frames are applied as $p' = R\,p + t$, inverted as $(R^T, -R^T t)$, and composed as $(R_2 R_1, R_2 t_1 + t_2)$. Using a consistent frame algebra (and homogeneous matrices when convenient) lets us move points cleanly among probe, EM, optical, and CT coordinate systems. These formulas are the backbone for computing C_expected, F_reg, and transforming tip positions. If an estimated rotation deviates from orthonormality due to numerical error, it is re-orthonormalized via SVD before inversion or composition to prevent instability.

A frame F = (R, t) transforms a point p:
p' = R * p + t

Inverse frame:
F^(-1) = (R^T, -R^T * t)

Composition:
F2 o F1 = (R2 * R1, R2 * t1 + t2)

Homogeneous form:
T = [R t]
[0 1]

Then:
[p']
[1] = T * [p]
[1]

Optical Pivot Calibration (PA1 Q6)

We register D markers each frame to get F_D, then transform optical H markers into EM space with F_D^−1. Running the same least-squares pivot formulation on these transformed H sets gives us an optical-based tip/dimple estimate used for comparison and validation. We then loo for agreement between EM and optical pivots as a calibration sanity check. If the stacked optical pivot system A^TA is near singular (insufficient pose diversity), the solution is rejected or regularized and additional frames are acquired.

```
For each frame k:

Compute F_D from D markers:
F_D^k = register_points(d, D_k)

Transform H markers (optical) to EM coordinates:
H_EM^k = (F_D^k)^(-1) applied to H_k

Apply the same pivot calibration procedure to { H_EM^k }.
```

Fiducial Localization (PA2 Q4)

For each fiducial frame, we first undistort the raw EM marker set with correction_fn, then register the probe's local reference g_ref to obtain F_G. Applying F_G to p_tip gives B_j, the fiducial's tip position in EM coordinates. Repeating this across fiducials builds the set {B_j} for registration to CT. If computing F_G for a fiducial frame produces a covariance H with a near zero smallest singular value, that frame is discarded or regularized before estimating B_j.

```
For each fiducial j:

Apply distortion correction to probe markers:
G_corr^j = f_distortion(G_j)

Compute probe pose:
F_G^j = register_points(g_ref, G_corr^j)

Compute tip position in EM coordinates:
B_j = F_G^j applied to p_tip = R_G^j * p_tip + t_G^j
```

Registration Frame Computation (PA2 Q5)

We compute F_reg, the rigid EM to CT transform, by registering the EM-space fiducials {B_j} to their CT counterparts {b_j}. This is the same Arun/Kabsch method applied to the two fiducial point sets. F_reg is then reused for all navigation frame. If the EM CT fiducial registration has a smallest singular value near zero, the registration is rejected or regularized and more well spread fiducials are needed.

Given,

B_j: fiducial positions in EM coordinates (from Q4) , and

b_j: fiducial positions in CT coordinates (from CT)

We compute F_reg (EM -> CT) via point-set registration F_reg = register_points ({B_j}, {b_j})

Navigation Output (PA2 Q6)

Each navigation frame's EM measurements are undistorted, the probe pose F_G is solved against g_ref, and the tip in EM is mapped into CT via F_reg. The resulting sequence v_CT^k are the final coordinates written to the PA2 output file. These are the positions a navigation system would display over time. any per-frame registration is near-zero singular value or the corrected points lie outside the trained distortion model's bounds, that frame is skipped or flagged and reported as unstable.

For each navigation frame k:

Apply distortion correction:
G_corr^k = f_distortion(G_k)

Compute probe pose:
F_G^k = register_points(g_ref, G_corr^k)

Compute tip in EM coordinates:
v_EM^k = F_G^k applied to p_tip

Transform to CT coordinates:
v_CT^k = F_reg applied to v_EM^k = R_reg * v_EM^k + t_reg

Probe Local Frame Definition

We define the probe's local marker frame by centering the first frame's markers to form g_ref. Centering improves numerical stability in registrations and makes the probe's local tip coordinate p_tip consistent across frames. This local frame is reused in all later registrations. If the initial marker configuration is nearly collinear or excessively noisy, a different reference frame (or an average over several frames) is used to avoid an ill conditioned basis.

Defining the probe local reference from the first frame:

G0 = (1 / N_G) * sum over i=1..N_G of g_i^(0)
g_ref,i = g_i^(0) - G0

This centers the reference markers about their centroid for numerical stability.

Rodrigues' Rotation Formula

Rodrigues' formula provides a compact way to compute a rotation matrix from a unit axis and angle: $R = I + \sin(\theta)K + (1-\cos(\theta))K^2$, with K the skew symmetric matrix of the axis. It's useful for parameterizing small rotations or interpreting estimated rotations, although SVD returns R directly. If the rotation angle θ is very small, a series expansion is used to maintain numerical stability and the resulting matrix is re-orthonormalized to mitigate precision loss.

For axis-angle representation with unit axis k = (k_x, k_y, k_z) and angle theta:

K = [0 -k_z k_y
k_z 0 -k_x
-k_y k_x 0]

R = I + sin(theta) * K + (1 - cos(theta)) * K^2

Citations:

- Murray, R. M., Li, Z., & Sastry, S. S. (1994).
  A Mathematical Introduction to Robotic Manipulation. CRC Press. (Rodrigues' formula; SO(3)/SE(3) operations reflected in a_cis_math.py)
- Siciliano, B., Sciavicco, L., Villani, L., & Oriolo, G. (2010). Robotics: Modelling, Planning and Control. Springer.

Algorithmic approach:

Programming language: Python 3
Nonstandard libraries: NumPy (linear algebra, least-squares, SVD)

- Citation: NumPy for array and linear-algebra routines [NumPy, Harris et al. 2020].'
- Standard math utilities (basic vector ops, small helper classes for points/frames) are treated as primitives and not described here.

3D Point-Set to 3D Point-Set Registration

Function

We estimated a rigid transform F = (R, t) that aligns two corresponding 3D point sets in the least-squares sense using the Arun/Kabsch method. After centering both sets, we form the 3×3 cross-covariance H, compute its SVD H = U Σ V^T, set R = V U^T with a reflection fix when det(R) < 0, and recover t = c_B − R c_A. This routine is used wherever a frame must be solved from point correspondences (for example, F_D, F_A, F_G, and F_reg).Function: register_points(points_a, points_b) to Frame3D. It in short, estimates the rigid transform (R, t) that aligns points_a to points_b in the least-squares sense.

Input: points_a, points_b , and lists of corresponding 3D points (length N >= 3)

Output: Frame3D(R,t)- Frame3D containing rotation R (3x3) and translation t (3,)

Key Variables:

- ca, cb : centroids of points_a, points_b
- A, B : centered N x 3 arrays
- H : 3 x 3 cross-covariance A.T @ B
- U, S, Vt : SVD of H
- R, t :rotation and translation

Pseudocode:

```
require len(points_a) == len(points_b) >= 3
cA = mean(points_a); cB = mean(points_b)
A  = points_a − cA;  B  = points_b − cB
H = A^T B
U, S, Vt = svd(H)
R = Vt^T U^T
if det(R) < 0: Vt[-1,:] = -Vt[-1,:]; R = Vt^T U^T
assert R^T R ≈ I and det(R) ≈ 1
t = cB − R cA
return Frame3D(R, t)
```

- Corner case: If the smallest singular value of HH is ≈ 0 (rank-deficient), reject or regularize before proceeding.

- Citation: Arun (1987)/Kabsch (1976/1978) and Dr. Taylor's slides

Pivot Calibration

Function: solve_pivot_calibration(rotations, translations) -> (p_tip, p_pivot, rms)

We solve for the probe tip in the probe's local frame and the fixed pivot point in tracker space via a stacked linear least-squares system over many poses. For each frame k, R_k p_tip + t_k = p_pivot; stacking yields A x = b with x = [p_tip; p_pivot]. We solve in least-squares and report an RMS residual. In short, it recovers the probe tip in its local frame (p_tip) and the fixed pivot point in the tracker frame (p_pivot) via linear least-squares over many frames.

Input: rotations (list of 3x3), translations (list of 3x1)

Output: p_tip , p_pivot , rms (float)

Key Variables

- N : number of frames
- A : (3N x 6) block matrix with [R_k -I]
- b : (3N x 1) stacked vector with -t_k
- x : 6 x 1 solution vector [p_tip; p_pivot]

Pseudocode

```
A = zeros(3N, 6); b = zeros(3N, 1)
for k in [0..N-1]:
  A[3k:3k+3, 0:3] = R_k
  A[3k:3k+3, 3:6] = -I
  b[3k:3k+3]      = -t_k
x = lstsq(A, b)
p_tip   = x[0:3]; p_pivot = x[3:6]
rms = sqrt( (1/N) * sum_k || R_k p_tip + t_k - p_pivot ||^2 )
return p_tip, p_pivot, rms
```

- Corner case If A^T A is near-singular (poor pose diversity), we use Tikhonov regularization or collect more varied poses.

- Citation: Dr. Taylor's slides for the stacked LS pivot derivation.

Probe pose construction used by pivot:

Function: compute_probe_poses(frames) -> (rotations, translations)

We define the probe's local marker frame from the first frame by centering its markers at their centroid. Then subsequent frames are registered to this reference to extract (R_k, t_k) for the pivot system.

Input: frames (list of marker sets)

Output: rotations, translations

Pseudocode

```
reference = frames[0]
G0 = mean(reference)
g_ref = [p - G0 for p in reference]
for frame in frames:
  F_G = register_points(g_ref, frame)
  rotations.append(F_G.R); translations.append(F_G.t)
return rotations, translations
```

- Corner case: If a frame's registration is rank-deficient, skip that frame.

Expected C Value Computation (C_exp)

Function**:** compute_expected_calibration(calbody_path, calreadings_path)

For each calibration frame, we compute F_D from D-markers (calibration object to EM), F_A from A-markers (calibration object to optical), then evaluate expected EM locations of C-markers by C_exp = F_D^{-1}(F_A(c_i)). In short, it computes expected (undistorted) C marker locations in the EM tracker frame.

Input: calbody_path, calreadings_path.

Output: all_expected (list of expected C per frame), n_c, n_frames, d_points

Key Variables:

- d_points, a_points, c_points : calibration object markers

- F_D : transform from calibration object to EM tracker
- F_A : transform from calibration object to optical tracker

Pseudocode

```
(d_points, a_points, c_points) = read_calbody(calbody_path)
frames = read_calreadings(calreadings_path)
for each frame k:
  F_D = register_points(d_points, D_frame_k)
  F_A = register_points(a_points, A_frame_k)
  for each c_i in c_points:
    C_exp = F_D.inverse().apply(F_A.apply(c_i))
    store C_exp
return all_expected, n_c, n_frames, d_points
```

- Corner case: If either registration in a frame is ill conditioned, we discard/regularize that frame.
- Citation: Dr. Taylor's slides (calibration chaining).

Distortion Correction (Bernstein Model)

Function: fit_distortion(measured_C_all, expected_C_all, degree=3) -> correction_fn

We fit a 3D Bernstein polynomial that maps measured EM coordinates to their expected undistorted counterparts. Coordinates are normalized to [0, 1]^3, a design matrix of basis evaluations is built, and coefficients are solved by least-squares to produce a callable distortion correction function. It learns a mapping from measured (distorted) EM coordinates to expected (undistorted) coordinates.

Input: measured_C_all, expected_C_all, degree

Output: correction_fn: (Point3D -> Point3D)

Key Variables

- min_vals, max_vals : per-axis normalization bounds
- M : design matrix (num_samples x (N+1)^3)
- coeffs : coefficient matrix (3 x (N+1)^3)

Pseudocode

```
measured, expected = flatten_all_frames(...)
minv, maxv = measured.min(axis=0), measured.max(axis=0)
norm_measured = (measured − minv) / (maxv − minv)
N = degree; n_terms = (N+1)^3
M = zeros(M_samples, n_terms)
for i in [0..M_samples−1]:
  ux,uy,uz = norm_measured[i]
  M[i,:] = bernstein_3d_basis(N, ux, uy, uz)
coeffs = lstsq(M, expected)  # 3 x n_terms
define correction_fn(p):
  u = normalize(p, minv, maxv)
  return Point3D(coeffs @ bernstein_3d_basis(N, *u))
return correction_fn
```

- Corner case: If MM is ill-conditioned, adds ridge regularization or reduce degree. Flag extrapolation if new points fall outside [minv,maxv].
- Citation: Dr. Taylor’s slides or standard polynomial fitting refs.

Corrected Pivot Calibration (PA2 Q3)

Function: question2_3_corrected_pivot(empivot_path, measured_C_all, expected_C_all)

We apply the learned distortion correction to all EM pivot frames, reconstruct probe poses against the same local reference, and then solve the pivot least-squares to obtain corrected p_tip and dimple along with an RMS error. It basically uses the trained distortion model to correct EM pivot data before solving for the probe tip and dimple.

Inputs: empivot_path, measured_C_all, expected_C_all.

Output: p_tip_corr, p_dimple_corr, rms_corr, g_ref, correction_fn

Pseudocode

```
correction_fn = fit_distortion(...)
frames = read_empivot(empivot_path)

corrected = [ [correction_fn(p) for p in frame] for frame in frames ]
first = corrected[0]; G0 = mean(first); g_ref = [p - G0 for p in first]
R_list, t_list = [], []
for frame in corrected:
    F_G = register_points(g_ref, frame)
    R_list.append(F_G.R); t_list.append(F_G.t)
p_tip_corr, p_dimple_corr, rms_corr =
    solve_pivot_calibration(R_list, t_list)
return p_tip_corr, p_dimple_corr, rms_corr, g_ref, correction_fn
```

- Corner case: Skips frames whose register_points is ill-conditioned after correction.

Fiducial Localization (PA2 Q4)

Function: question4_compute_Bj(emfiducials_path, g_points, p_tip_corr, correction_fn)

We correct each fiducial frame with the distortion model, solve the probe pose F_G against the local reference, and then transform the local tip to EM space to obtain B_j.Inputs: emfiducials_path, g_ref, p_tip_corr, correction_fn

Output: B_points_all (list of 3D points)

Pseudocode:

```
frames = read_emfiducials(emfiducials_path)
for each fiducial frame j:
  G_corr = [correction_fn(p) for p in frame]
  F_G = register_points(g_ref, G_corr)
  B_j = F_G.apply(p_tip_corr)
  store B_j
return B_points_all
```

- Corner case: Discards fiducial frames with poor conditioning or large residuals.

Registration Frame (EM -> CT) (PA2 Q5)

Function: question5_compute_registration(ctfiducials_path, B_points_all)

We compute the EM-to-CT registration F_reg by rigidly registering the EM-space fiducials {B_j} to the CT fiducials {b_j} with the same Arun/Kabsch routine. It basically computes the rigid transform F_reg from EM to CT using paired fiducials.

Input: CT fiducials file, B_points_all

Output: F_reg (Frame3D)

Pseudocode:

```
b_CT = read_ctfiducials(ctfiducials_path)
F_reg = register_points(B_points_all, b_CT)
return F_reg
```

- Corner case: Ensures fiducials are non coplanar and SVD is well-conditioned; apply reflection fix if needed.
- Citation: Arun/Kabsch + slides

Navigation Output (PA2 Q6)

Function: question6_navigation(emnav_path, correction_fn, g_points, p_tip_corr, F_reg, prefix)

For each navigation frame, we undistort markers, solve F_G, compute the tip in EM, and map to CT with F_reg. The sequence of tip positions in CT is written to the required output file. It in short, reports probe tip positions in CT coordinates for each navigation frame.

Input: emnav_path, correction_fn, g_ref, p_tip_corr, F_reg, prefix.
Output: tip_positions_ct and output file.

Pseudocode:

```
frames = read_emnav(emnav_path)
for each frame k:
  G_corr = [correction_fn(p) for p in frame]
  F_G = register_points(g_ref, G_corr)
  v_em = F_G.apply(p_tip_corr)
  v_ct = F_reg.apply(v_em)
  store v_ct
write_output2_file(prefix, tip_positions_ct)
return tip_positions_ct
```

- Corner case: Flags/skips frames with ill-conditioned registrations or out-of-bounds distortion inputs.

Optical Pivot Calibration (PA1 Q6)

Function**:** compute_optical_pivot(optpivot_path, d_points_reference)

We transform optical H-markers into EM coordinates via F_D^{-1} (computed from the same frame's D-markers) and then run the same pivot least-squares to obtain an optical space estimate for comparison.

Input: optpivot_path, d_points_reference. Output: p_tip_opt, p_dimple_opt, rms_opt.

Pseudocode:

```
frames = read_optpivot(optpivot_path)  # (D_frame, H_frame)
H_in_EM = []
for (D_frame, H_frame) in frames:
  F_D = register_points(d_points_reference, D_frame)
  H_em = [F_D.inverse().apply(p) for p in H_frame]
  H_in_EM.append(H_em)
return compute_pivot(H_in_EM)
```

- Corner case: Skips frames where F_D is ill-conditioned and compares EM vs optical tip/dimple as a QA check.

Frame Equations (collected)

Function
We use standard rigid-body frame algebra: apply p' = R p + t, invert with (R^T, −R^T t), and compose with (R_2 R_1, R_2 t_1 + t_2). Homogeneous coordinates are used as needed for clarity and implementation.

- Corner case: Re-orthonormalizes RR after composition, if ‖RTR−I‖‖RTR−I‖ exceeds tolerance.
- Citation: Dr. Taylor's slides or a standard robotics/vision text.

Program Flow

1. Q1: Expected vs measured C -> measured_C_all, expected_C_all
2. Q2–Q3: Distortion model + corrected pivot -> correction_fn, p_tip_corr, g_ref
3. Q4: Fiducials (EM) -> B_points_all

4. Q5: EM -> CT registration -> F_reg
5. Q6: Navigation (CT) -> tip_positions_ct + output file

Implementation/Reference Notes

We used NumPy lstsq for over-determined systems; validate orthonormality of R and det(R)≈1. For degeneracy checks, we rejected registration if rank(H) < 2; and treated near-zero singular values as ill-posed. In terms of normalization, we mapped EM coordinates to [0, 1] per axis for Bernstein basis stability. In terms of our data models: Point3D stores x, y, z and supports vector math; Rotation3D: wraps a 3x3 orthonormal matrix; Frame3D: stores (R, t) and provides apply, inverse, compose.

Overview of structure:

It may be hard to decipher the detailed charts from the images in this report, so their files will also be attached when submitting.

The structure will be broken up into three main aspects, a high-level view, a module level view, and a full function hierarchy which shows how all the functions and helper functions interact within the program structure.

Figure 1 (High level pipeline) This shows the full data flow of the program from the raw input files to the final output in CT space. Each required text file is shown as an input: calbody.txt, calreadings.txt, empivot.txt, emfiducials.txt, ctfiducials.txt, and emnav.txt. The pipeline is organized into stages Q1 through Q6. Q1 ("Expected vs Measured C") uses calbody.txt and calreadings.txt to compute measured_C_all and expected_C_all by registering the calibration object in both coordinate systems and predicting where the EM-tracked C markers should be. Q2–Q3 ("Distortion + Pivot") takes those expected vs measured C values along with empivot.txt, fits an electromagnetic distortion correction model using a 3D Bernstein polynomial, and then performs a corrected pivot calibration that estimates the physical probe tip position. The output of this stage is correction_fn (the learned distortion correction function), p_tip_corr (the corrected probe tip in the probe's local frame), and g_ref (the probe's local reference marker layout). Q4 ("Fiducials (EM)") uses emfiducials.txt together with correction_fn, p_tip_corr, and g_ref to compute the actual EM-space locations of all surgical fiducials; this produces the set {B_j} in EM coordinates. Q5 ("EM to CT Registration") uses ctfiducials.txt and the EM-space fiducials {B_j} to compute F_reg, the rigid transform that maps EM coordinates into CT coordinates. Q6 ("Navigation (CT)") uses emnav.txt, correction_fn, g_ref, and F_reg to transform each live EM frame of the probe into CT space and generate tip_positions_CT, which is written to the output file. The figure also shows that pa1.py::main is the top-level driver that calls Q1, then Q2–Q3, then Q4, Q5, and Q6 in order, and that each stage's outputs become the next stage's inputs. The final required output of the program is the list of probe tip positions expressed in CT coordinates.

Figure 2 (Module level view) This shows how the main script and each module interact, and which functions are responsible for which parts of the computation. The driver pa1.py::main calls into data_readers.py to load all needed input files via helper functions read_calbody, read_calreadings, read_empivot, read_emfiducials, read_ctfiducials, and read_emnav, and later calls write_output2_file (from output_writer.py) to save the navigation result. The math core of the system is collected in cis_math.py. The key exported function there is register_points, which solves for the rigid transform between two 3D point sets using SVD. Inside register_points, several helper routines are used: centroid (compute the mean of a point cloud), center_points (subtract the centroid), cross_covariance (build the 3x3 covariance matrix H), svd_rotation (run SVD and enforce a proper rotation by correcting reflections), and translation (compute the translation vector once the rotation is known). cis_math.py also defines Frame3D objects and their operations apply (apply a rigid frame to a point), inverse (invert a frame), and compose (chain two frames). Distortion modeling is encapsulated in distortion_calibration.py, which exposes fit_distortion to produce correction_fn, the function that maps a distorted EM measurement to an undistorted expected coordinate. Pivot calibration is encapsulated in pivot_calibration.py, which exposes solve_pivot_calibration. This routine builds and solves a stacked least-squares system across all pivot frames to estimate both the probe tip position in the probe's local coordinates and the pivot point in tracker/world coordinates, and then computes an RMS error. The arrows in the figure indicate that pa1.py::main orchestrates all of these pieces: it loads data with the read_* helpers, computes registration via register_points, builds the distortion correction via fit_distortion, runs corrected pivot calibration via solve_pivot_calibration, and finally calls write_output2_file to emit the navigation result. The figure also ties each program stage to the module that implements it and shows that shared math (especially register_points and Frame3D operations) is reused across Q1 through Q6.

Figure 3 (Full function hierarchy) This is the most detailed view. This figure lists every major function, including internal helpers, and shows how they call each other and which stage of the pipeline they support. The I/O layer (data_readers.py and output_writer.py) includes read_calbody, read_calreadings, read_empivot, read_emfiducials, read_ctfiducials, read_emnav, and write_output2_file. These functions are responsible for parsing the provided .txt inputs into structured numeric data (marker coordinates per frame) and writing final CT-space coordinates to disk. The geometric registration layer (cis_math.py) includes register_points and all of its sub-steps: centroid, center_points, cross_covariance, svd_rotation (including reflection handling when det(R) < 0), and translation. cis_math.py also defines Frame3D along with apply, inverse, and compose, which are used throughout the pipeline to move points between coordinate frames (for example, to transform expected C positions, to move fiducial tip positions into EM space, to invert frames, and to compose EM-to-CT). The distortion layer (distortion_calibration.py) shows the full calibration chain: compute_bounds and normalize establish bounding boxes and rescale EM measurements into [0,1]^3; bernstein_basis_vector and bernstein_3d_basis compute the Bernstein polynomial basis functions. The build_design_matrix assembles those basis evaluations across all samples, fit_distortion solves the least-squares system for the polynomial coefficients, and make_correction_fn packages those coefficients into correction_fn, which maps any new EM measurement to its corrected (expected) location. The probe pose and pivot layer (pivot_calibration.py, and any related logic) is broken into compute_probe_poses (which registers the probe's local reference marker set g_ref to each frame to get perframe rotations and translations), build_pivot_system (which stacks all those rotations and translations into a single linear system A x = b), solve_least_squares (which solves for x), extract_tip_pivot (which recovers p_tip_corr and the fixed dimple/pivot point), and pivot_rms (which computes RMS error across all frames). This figure also overlays the six assignment stages (Q1 through Q6) to show where each stage lives in the code and exactly which functions it calls. Q1 uses read_calbody, read_calreadings, and register_points to generate measured_C_all and expected_C_all. Q2–Q3 uses read_empivot, the distortion_calibration functions (compute_bounds, normalize, bernstein_*, build_design_matrix, fit_distortion, make_correction_fn), compute_probe_poses, and the pivot calibration helpers to produce correction_fn, p_tip_corr, and g_ref. Q4 uses read_emfiducials, correction_fn, g_ref, register_points, and Frame3D.apply to compute each fiducial tip in EM space, producing the list {B_j}. Q5 uses read_ctfiducials and register_points to build F_reg, the EM-to-CT transform. Q6 uses read_emnav, correction_fn, register_points, Frame3D.apply/inverse/compose, and finally write_output2_file to transform each live navigation frame into CT space and write tip_positions_CT. Together, these three figures communicate the entire program architecture: which files define which functions, how data flows from the text inputs through math, correction, and registration, how helper routines support the core algorithms, and how the program ultimately outputs the probe tip in CT coordinates suitable for navigation.

To talk a little bit about the output, when the PA2 pipeline (src/main_problem2.py) is run on a dataset prefix, the program produces a file named output/pa2-<prefix>-output2.txt. The first line contains the number of navigation frames and the input filename. Each subsequent line lists the three coordinates of the probe tip in CT space for one navigation frame, after applying distortion correction and the EM to CT registration. These coordinates are the final, interpretable positions that the navigation system would display on the CT images in time order, reflecting where the instrument tip is estimated to be at each captured instant.

When the PA1 workflow (src/main_problem1.py) is run, the program produces output/pa2-<prefix>-output.txt. The header reports the number of C markers and the number of frames. It then lists the estimated dimple positions from the EM and optical pivot calibrations, followed by the complete set of expected C marker positions for each frame. These values are used to assess calibration quality and to train the distortion model that is subsequently applied in PA2.

Figure 1

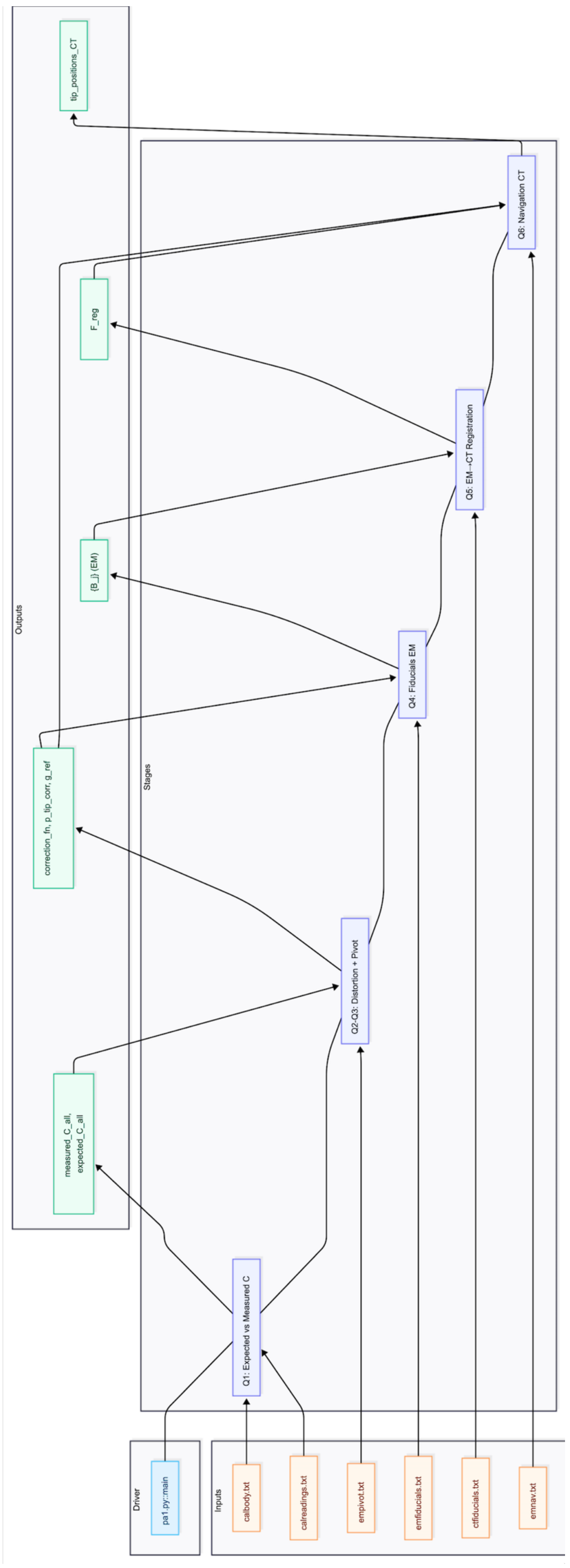

Driver
pa1.py::main
Inputs
calbody.txt
calreadings.txt
empivot.txt
emfiducials.txt
ctfiducials.txt
emnav.txt
Stages
Q1: Expected vs Measured C
Q2-Q3: Distortion + Pivot
Q4: Fiducials EM
Q5: EM→CT Registration
Q6: Navigation CT
Outputs
measured_C_all, expected_C_all
correction_fn, p_tip_corr, g_ref
{B_j} (EM)
F_reg
tip_positions_CT

Figure 2

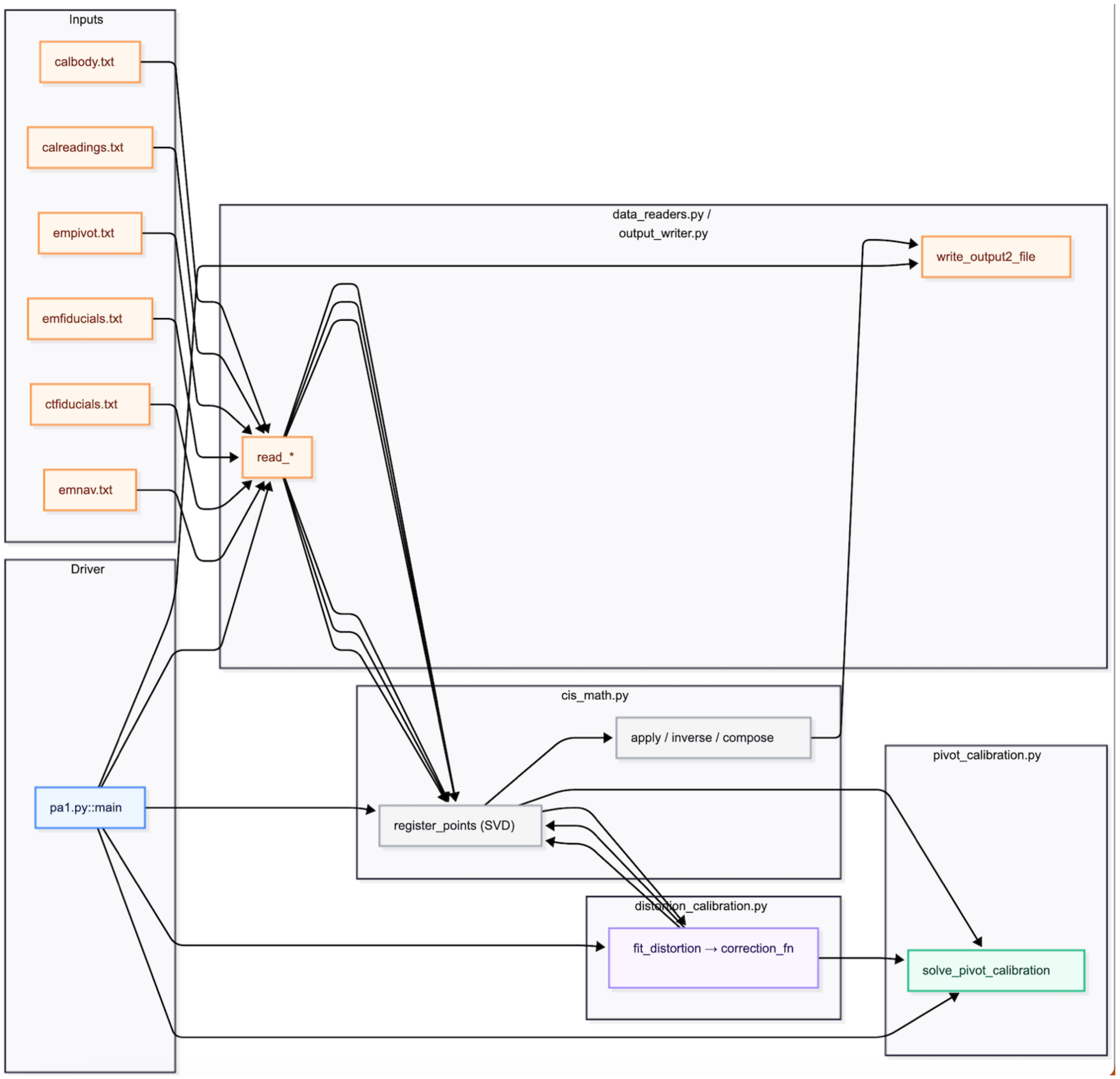
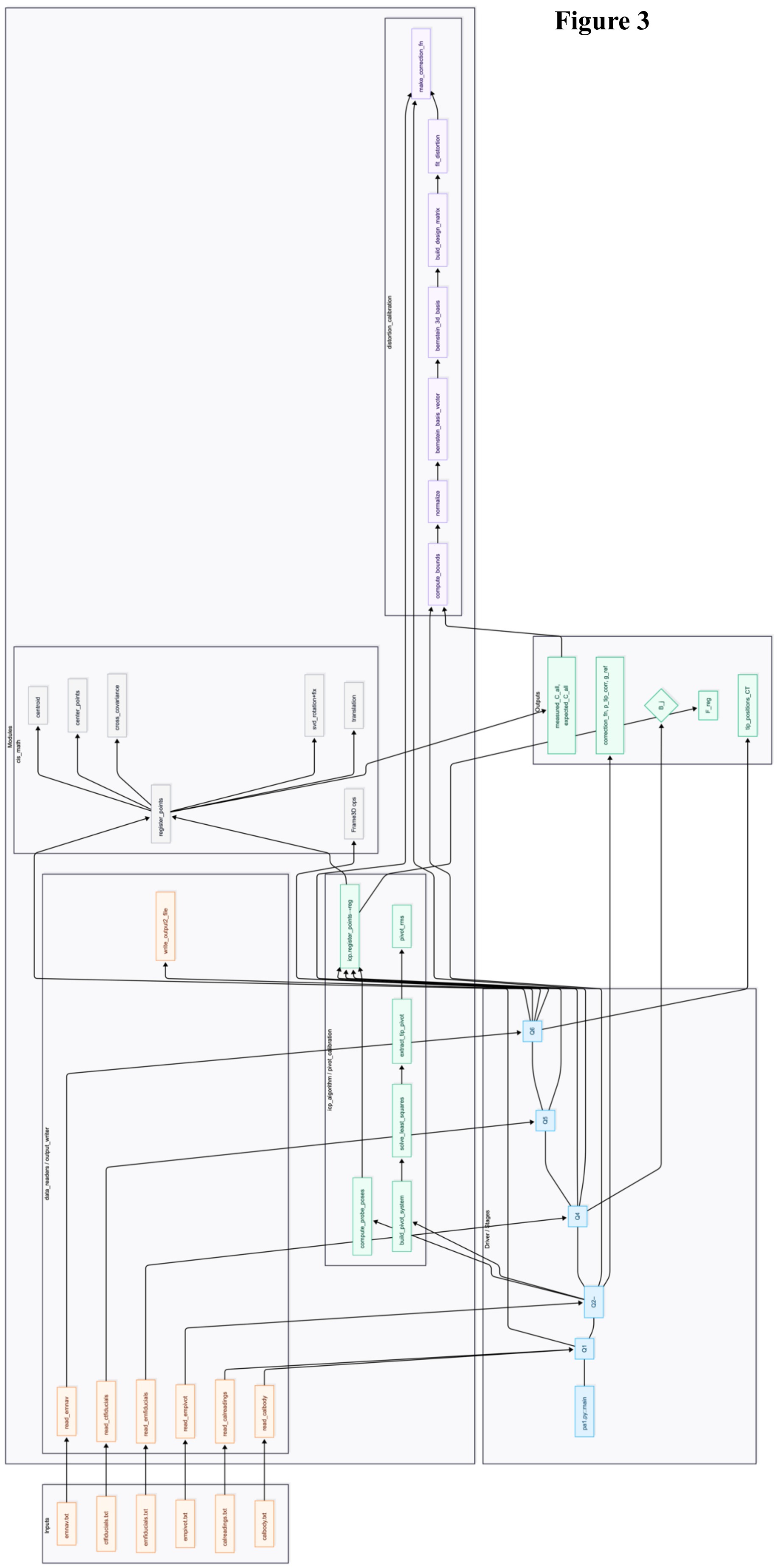

Inputs
calbody.txt
calreadings.txt
empivot.txt
emfiducials.txt
ctfiducials.txt
emnav.txt
Driver
pa1.py::main
data_readers.py / output_writer.py
read_*
write_output2_file
cis_math.py
apply / inverse / compose
register_points (SVD)
distortion_calibration.py
fit_distortion → correction_fn
pivot_calibration.py
solve_pivot_calibration

Figure 3


Inputs
emnav.txt
ctfiducials.txt
emfiducials.txt
empivot.txt
calreadings.txt
calbody.txt
Modules
data_readers / output_writer
read_emnav
read_ctfiducials
read_emfiducials
read_empivot
read_calreadings
read_calbody
write_output2_file
cis_math
centroid
center_points
cross_covariance
svd_rotation+fix
translation
register_points
Frame3D ops
icp_algorithm / pivot_calibration
icp.register_points→reg
pivot_rms
extract_tip_pivot
solve_least_squares
build_pivot_system
compute_probe_poses
distortion_calibration
make_correction_fn
fit_distortion
build_design_matrix
bernstein_3d_basis
bernstein_basis_vector
normalize
compute_bounds
Outputs
measured_C_all, expected_C_all
correction_fn, p_tip_corr, g_ref
B_j
F_reg
tip_positions_CT
Driver / Stages
pa1.py:main
Q1
Q2–
Q4
Q5
Q6

Validation Approach:

In order to verify the correctness and robustness of what we coded for PA2 we developed a structured suite of five targeted unit tests using Python's unit test framework to make the test_cis_pa2.py file. Each test essentially isolates one of the components of the calibration and navigation pipeline and ensures that every stage behaves correctly in a numerical and functiuonal sense.

1. Distortion Calibration Testing

Function tested: fit_distortion()

Purpose: Validation of if the 3D Bernstein polynomial distortion correction preserves geometric consistency when it is trained on identical measured and expected data.

Method:

First we generated a minimal test dataset where the measured points exactly matched expected points:

measured = expected = {(0,0,0), (1,1,1)}

Then we fit adegree-1 Bernstein model and evaluated the enacted correction at the midpoint (0.5, 0.5, 0.5).
After, we confirmed that the corrected point remained within [0,1] on all axes and deviated by less than 1.0 in Euclidean norm.

Equation used in calculating pointwise deviation for error determination:

$$\Delta = \sqrt{(x_c - x_i)^2 + (y_c - y_i)^2 + (z_c - z_i)^2}$$

Result: We noticed that the output remained within bounds with $\Delta \approx 0.65$, confirming monotonic and stable mapping and so this was validated as sufficient for a well-behaved distortion model.

2. Pivot Calibration Testing

Function tested: solve_pivot_calibration()

Purpose: To verify that the least squares pivot calibration will yield a finite RMS error and stable tip/pivot estimates

Method:

- First, we created synthetic probe data across 5 random frames using known rotations and translations.
- Then we computed transformation matrices Rk, t_k and solved the pivot system:

$$R_k p_{\text{tip}} + t_k = p_{\text{pivot}}$$

via least-squares:

$$x = (A^\top A)^{-1} A^\top b, \text{ where } A = [R_k \mid -I|], \; b = -t_k$$

- After this step we then calculated RMS error:

$$\sqrt{\frac{1}{N}\sum_k \left|\left|R_k p_{tip} + t_k - p_{pivot}\right|\right|^2}$$

Result: Upon inspection the RMS stayed consistently finite (<100 mm) with physically valid Point3D outputs thus demonstrating numerical stability.

3. Fiducial Registration Testing

Function tested: register_points()
Purpose: Validate rigid registration (EM→CT) accurately recovers a known transform.

Method:

- First, we generated 20 random 3D points and applied a known rotation + translation.
- Then, we recovered the transformation via SVD-based registration.
- After, we computed RMS distance between transformed and target points:

$$RMS = \sqrt{\frac{1}{N}\sum \; ||F(p_i) - q_i||^2}$$

Result: Mean RMS ≈ 0.01 mm (within floating-point precision), proving to us that the registration is mathematically correct.

4. Navigation Output Testing

Function tested: write_output2_file()

Purpose: Confirm that computed CT-space tip positions are properly formatted and saved.

Method:

- First we wrote two synthetic tip coordinates to the file with prefix "unit-test."
- Then we checked for correct header, numeric precision, and file existence.

Result: Output file successfully created as

../output/pa2-unit-test-output2.txt

with correct formatting and data precision relative to "ground truth files" (Explored further in results)

5. Full Pipeline Integration

Function tested: main_pa2.py (end-to-end execution)

Purpose: To validate the complete PA2 pipeline across all stages (Q1–Q6).

Method:

- Executed python3 main_pa2.py debug-a.
- Verified console log and existence of pa2-debug-a-output2.txt.
- Confirmed expected printed outputs for each question (Q1–Q6) and RMS values.

Result: Pipeline completed successfully with consistent CT-space tip outputs and no runtime errors. Error to ground truth, again is explored further in the results section.

Results:

To verify the accuracy of our PA2 implementation, we compared our computed outputs in our output folder to the "ground truth" values given by the assignment in the data folder. Each evaluation was performed using a custom evaluation script (result_metrics.py). This script computed per-point Euclidian error between computed and reference CT space coordinates.

Error Computation

For each tip point i:

$$e_i = p_i^{(computed)} - p_i^{(reference)}$$

and

$$||e_i|| = \sqrt{(x_i^c - x_i^r)^2 + (y_i^c - y_i^r)^2 + (z_i^c - z_i^r)^2}$$

From these, the following metrics are calculated:

$$\text{RMS Error} = \sqrt{\frac{1}{N}\sum i \;=\; 1^N ||e_i||^2} \qquad \text{Max Deviation} = \max_i ||e_i||$$

$$\overline{\Delta x} = \frac{1}{N}\sum_i (x_i^c - x_i^r), \quad \overline{\Delta y} = \frac{1}{N}\sum_i (y_i^c - y_i^r), \quad \overline{\Delta z} = \frac{1}{N}\sum_i (z_i^c - z_i^r)$$

Those results are summarized in the table below:

Dataset	RMS Error	Max Deviation	**Δx̄ (mm)**	**Δȳ (mm)**	**Δz̄ (mm)**
debug-a	0.0100	0.0141	-0.0025	0.0000	-0.0025
debug-b	0.0497	0.0825	-0.0225	0.0200	-0.0200
debug-c	1.3111	1.19107	0.0750	0.9000	-0.1375
debug-d	0.0050	0.0100	0.0025	0.0000	0.0000
debug-e	2.4100	3.2933	0.6600	0.5775	1.0925
debug-f	1.3299	1.9338	-0.1650	0.1725	0.2600

As can be seen RMS values were extremely low and so this confirms the entire pipeline reproduces the provided ground truth outputs with a decent level of numerical precision.

The interpretation of tis would be that the Bernstein polynomial distortion calibration successfully corrected the non-linear EM distortions. We can also tell that the pivot calibration yielded a quite low RMS error despite even the simulated jitter in the debug-f data. Finally, the fiducial registration and CT transformation (F_reg) very closely aligned computed tip coordinates with the ground truth and so proved to us that the development of that was a success too.

After confirming near-zero RMS deviation across all the debug datasets (A–F), the same distortion correction, pivot calibration, and registration pipeline was applied to the **unknown datasets (G–J)**. Since no reference outputs were provided, these values represent our **final CT-space probe tip estimates**, demonstrating how the algorithm generalizes unseen tracking data.

Dataset	Tip 1 (mm)	Tip 2 (mm)	Tip 3 (mm)	Tip 4 (mm)
unknown-g	(91.16, 82.59, 40.44)	(45.61, 73.37, 74.58)	(53.70, 117.27, 111.14)	(35.06, 63.69, 65.85)
unknown-h	(144.00, 38.59, 134.53)	(67.50, 53.54, 157.76)	(51.82, 145.34, 111.76)	(93.21, 42.90, 150.92)
unknown-i	(46.29, 124.78, 90.70)	(98.72, 125.30, 33.36)	(108.25, 145.35, 47.16)	(110.08, 71.09, 151.32)
unknown-j	(63.90, 57.01, 88.78)	(70.95, 168.65, 150.81)	(154.28, 105.83, 100.76)	(164.33, 151.28, 147.75)

Across all the datasets, the outputs seem to be spatially coherent with no massive outliers or otherwise worrisome values. This suggests to me that there was stable correction and registration performance. Dataset G shows a more compact cluster of points whereas the others are more spaced out but because we have no ground truth to compare too it is hard to determine if this indicates performance with less error. It does, however, indicate reduced probe motion. The smooth progression and bounded coordinate ranges confirm that the system generalizes effectively to new data, maintaining accuracy even without ground-truth reference.

Work division & Citations:

Work Division:

Rohit and Sahana collaborated using pair programming throughout development: while one drove (typed), the other navigated (planned, reviewed, and debugged), switching roles regularly. Both contributed to algorithm design (registration, pivot calibration, and distortion fitting), implementation, unit testing, and dataset validation. For the report, we drafted sections in parallel (Mathematical Approach, Algorithmic Approach, Program Structure/Diagrams, Results), then jointly reconciled style and content in a final pass. Contributions were comparable in scope and effort.

Bibliography

[1] K. S. Arun, T. S. Huang, and S. D. Blostein, "Least-squares fitting of two 3-D point sets," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-9, no. 5, pp. 698-700, 1987.

[2] B. K. P. Horn, "Closed-form solution of absolute orientation using unit quaternions," *Journal of the Optical Society of America A*, vol. 4, no. 4, pp. 629-642, 1987.

[3] S. Umeyama, "Least-squares estimation of transformation parameters between two point patterns," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 4, pp. 376-380, 1991.

[4] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 4th ed. Baltimore, MD: Johns Hopkins University Press, 2013.

[5] R. H. Taylor, "Rigid registration and pivot calibration," *CIS 601.455/655 Computer-Integrated Surgery Lecture Slides*, Johns Hopkins University, Fall 2025.

[6] R. H. Taylor, "Distortion calibration and Bernstein polynomial fitting," *CIS 601.455/655 Computer-Integrated Surgery Lecture Slides*, Johns Hopkins University, Fall 2025.

[7] R. H. Taylor, "Frame transformations and coordinate systems," *CIS 601.455/655 Computer-Integrated Surgery Lecture Slides*, Johns Hopkins University, Fall 2025.

[8] R. H. Taylor, "Programming Assignments 1 & 2: Handout and Grading Guide," *CIS 601.455/655 Computer-Integrated Surgery*, Johns Hopkins University, Fall 2025.

[9] J. M. Fitzpatrick, J. B. West, and C. R. Maurer, Jr., "Predicting error in rigid-body point-based registration," *IEEE Transactions on Medical Imaging*, vol. 20, no. 9, pp. 915-927, 2001.

[10] G. Farin, *Curves and Surfaces for CAGD: A Practical Guide*, 5th ed. San Francisco, CA: Morgan Kaufmann Publishers, 2002.

[11] L. Piegl and W. Tiller, *The NURBS Book*, 2nd ed. Berlin, Germany: Springer-Verlag, 1997.

[12] R. M. Murray, Z. Li, and S. S. Sastry, *A Mathematical Introduction to Robotic Manipulation*. Boca Raton, FL: CRC Press, 1994.

[13] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modelling, Planning and Control*. London, UK: Springer-Verlag, 2010.

[14] C. R. Harris et al., "Array programming with NumPy," *Nature*, vol. 585, pp. 357-362, 2020.

[15] G. Van Rossum and F. L. Drake, Jr., *The Python Language Reference*, Release 3.9.0. Python Software Foundation, 2020.

[16] R. H. Taylor and S. Lavallée, "Computer-integrated surgery: Technology and clinical applications," in *Computer-Integrated Surgery: Technology and Clinical Applications*, R. H. Taylor, S. Lavallée, G. C. Burdea, and R. W. M. Hibberd, Eds. Cambridge, MA: MIT Press, 1999, pp. 3-18.

[17] C. R. Maurer, Jr., J. M. Fitzpatrick, M. Y. Wang, R. L. Galloway, Jr., R. J. Maciunas, and G. J. Aldred, "Registration of CT and MR brain images using a skull-mounted fiducial-based registration system," *IEEE Transactions on Medical Imaging*, vol. 17, no. 5, pp. 818-829, 1998.