Programming Assignments 1 & 2 601.455 and 601/655 Fall 2023

Please also indicate which section(s) you are in (one of each is OK)

Score Sheet

Name 1	Keerthana Thammana
Email	lthamma1@jhu.edu
Other contact information (optional)	
Name 2	Kiana Bronder
Email	kbronde1@jhu.edu
Other contact information (optional)	
Signature (required)	I (we) have followed the rules in completing this assignment We both are in 601.455 Keerthana T. Kiana Bronder

Grade Factor		
Program (40)		
Design and overall program structure	20	
Reusability and modularity	10	
Clarity of documentation and programming	10	
Results (20)		
Correctness and completeness	20	
Report (40)		
Description of formulation and algorithmic approach	15	
Overview of program	10	
Discussion of validation approach	5	
Discussion of results	10	
TOTAL	100	

1 Mathematical Approach

We developed a math package for 3D points, rotations, and frame transformations. We also implemented Arun's registration method, pivot calibration, and distortion correction, as covered in class.

1.1 Cartesian Math

Using a generic 3D point class from Github (cited in the code), we expanded on it to make it function as part of a frame. Each frame contains a 3D rotation matrix and 3D point, combining as follows:

$$\mathbf{F} = \begin{bmatrix} \mathbf{R} & \vec{p} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The inverse of a frame was calculated according to the class slides and with the knowledge that $R^{-1} = R^T$ as:

$$\mathbf{F^{-1}} = \begin{bmatrix} \mathbf{R^{-1}} & -\mathbf{R^{-1}} \cdot \vec{p} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R^T} & -\mathbf{R^T} \cdot \vec{p} \\ 0 & 1 \end{bmatrix}$$

Frame multiplication was implemented as:

$$\mathbf{F_A} \cdot \mathbf{F_B} = \begin{bmatrix} \mathbf{R_A} & \vec{p}_A \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R_B} & \vec{p}_B \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R_A} \cdot \mathbf{R_B} & \mathbf{R_A} \cdot \vec{p}_B + \vec{p}_A \\ 0 & 1 \end{bmatrix}$$

Point set to point set registration was implemented as:

$$\vec{v}_A = \mathbf{F_{AB}} \cdot \vec{v}_B = \mathbf{R_{AB}} \cdot \vec{v}_B + \vec{p}_{AB}$$

Where v_A is some point v with respect to frame A, and v_B is that same point with respect to frame B. $\mathbf{F_{AB}}$ is defined in Written Homework 1 as $\mathbf{F_A^{-1}} \cdot \mathbf{F_B}$ and implemented as such. Even if there are intermediary frames (e.g. $\mathbf{F_C}$ between $\mathbf{F_A}$ and $\mathbf{F_B}$), the above equation remains the same since it would be:

$$\vec{v}_A = \mathbf{F_{AC}} \cdot \mathbf{F_{CB}} \cdot \vec{v}_B = \mathbf{F_A^{-1}} \cdot \mathbf{F_C} \cdot \mathbf{F_C^{-1}} \cdot \mathbf{F_B} \cdot \vec{v}_B = \mathbf{F_A^{-1}} \cdot \mathbf{I} \cdot \mathbf{F_B} \cdot \vec{v}_B = \mathbf{F_A^{-1}} \cdot \mathbf{F_B} \cdot \vec{v}_B = \mathbf{F_{AB}} \cdot \vec{v}_B$$

1.2 Arun's Registration Method

We opted to implement a closed form solution for registration rather than an iterative method to have a quicker method. We used Arun's registration, which we describe below.

$$\bar{a} = \frac{1}{N} \sum_{i=1}^{N} \vec{a}_i \qquad \tilde{a}_i = \vec{a}_i - \bar{a}$$

$$\bar{b} = \frac{1}{N} \sum_{i=1}^{N} \vec{b}_i \qquad \tilde{b}_i = \vec{b}_i - \bar{b}$$

$$\mathbf{H} = \sum_i \begin{bmatrix} \tilde{a}_{i,x}\tilde{b}_{i,x} & \tilde{a}_{i,x}\tilde{b}_{i,y} & \tilde{a}_{i,x}\tilde{b}_{i,y} \\ \tilde{a}_{i,y}\tilde{b}_{i,x} & \tilde{a}_{i,y}\tilde{b}_{i,y} & \tilde{a}_{i,y}\tilde{b}_{i,z} \\ \tilde{a}_{i,z}\tilde{b}_{i,x} & \tilde{a}_{i,z}\tilde{b}_{i,y} & \tilde{a}_{i,z}\tilde{b}_{i,z} \end{bmatrix}$$

We then computed the singular value decomposition to get $\mathbf{H} = \mathbf{USV^t}$, where the rotation matrix $\mathbf{R} = \mathbf{VU^t}$. The translation was calculated using $T = \bar{b} - \mathbf{R}\bar{a}$. If the determinant of the rotation matrix is -1, the last columm of V was multiplied by -1 to make sure that the resulting rotation matrix has a determinant of 1. A determinant of -1 could be caused by having a very small rotation.

1.3 Pivot Calibration

For pivot calibration for the EM tracking data, we set up a least squares problem with 2 unknowns: p_{tip} and p_{dimple}, where p_{tip} is the translation of the tip from the probe, and p_{dimple} is the location of the pivot point as measured by the EM tracker.

We used the first frame of pivot calibration data to define a local probe coordinate system $\vec{g}_j$, and then translate the observations relative to this midpoint for every frame.

$$\vec{G}_0 = \frac{1}{N_G} \sum \vec{G}_j$$

$$\vec{g}_j = \vec{G}_j - \vec{G}_0$$

For each frame k of pivot data, we computed a transformation $F_G[k]$ using Arun's registration method between $\vec{g}_j$ and $\vec{G}_j$, such that $\vec{G}_j = F_G[k]\vec{g}_j$. Thus, $p_{dimple} = F_G[k]p_{tip}$.

The pivot point's location p_{dimple} in EM tracker coordinates is $p_{dimple} = \mathbf{R}_i p_{tip} + t_i$, which can be rearranged to get:

$$\begin{bmatrix} \mathbf{R}_i & -I \end{bmatrix} \begin{bmatrix} p_{tip} \\ p_{dimple} \end{bmatrix} = -t_i$$

Combining data from all frames gives us:

$$\begin{bmatrix} \mathbf{R}_1 & -I \\ \mathbf{R}_2 & -I \\ \vdots & \vdots \\ \mathbf{R}_N & -I \end{bmatrix} \begin{bmatrix} p_{tip} \\ p_{dimple} \end{bmatrix} = \begin{bmatrix} -t_1 \\ -t_2 \\ \vdots \\ -t_N \end{bmatrix} \qquad \leftarrow \mathbf{A}x = b$$

At first, we tried solving this equation with regular least squares, however, the error in our generated values was quite high. Instead, we solved this using singular value decomposition least squares, where

$$\mathbf{A}x = b \rightarrow \qquad \mathbf{U} \begin{bmatrix} \mathbf{S} \\ \mathbf{0} \end{bmatrix} \mathbf{V}^T x = \mathbf{b}$$

For pivot calibration of the optical tracking data, we used the same procedure as above, but calculated the transformation F_D before the pivot calibration for each frame to transform the optical tracker position into EM tracker coordinates.

1.4 Distortion Correction

Since the EM tracker values are affected by distortion, we computed a distortion correction function using Bernstein polynomials. Bernstein polynomials are given by the equation

$$B_{N,k}(v) = \binom{N}{k} (1-v)^{(N-k)} v^k$$

$\vec{p}$ is the known 3D ground truth values, given by $C_i^{expected}$, and $\vec{q}$ is the values returned by the EM sensor, given by C_i. Bernstein polynomials are unstable outside of the range [0,1], so the 3D points were normalized using a bounding box with the minimum and maximum for each axis. The following function was used for scaling: $u = \frac{q - q_{min}}{q_{max} - q_{min}}$

The interpolation function for 3D points is given by

$$P(c_0, ..., c_N; u_x, u_y, y_z) = \sum_{i=0}^{N} \sum_{j=0}^{N} \sum_{k=0}^{N} c_k F_{ijk}(u_x, u_y, u_z)$$

$$F_{ijk}(u_x, u_y, u_z) = B_{N,i}(u_x) B_{N,j}(u_y) B_{N,k}(u_z)$$

We used N = 5 because the distortion function gave accurate results, but using a higher N would have decreased error at the expense of using more computational power.

We then calculated the control points (the coefficients in the interpolation function given by the c) using SVD least squares on the following equation:

$$\begin{bmatrix} \vdots & \vdots & \vdots \\ F_{000}(\vec{u}_i) & \ldots & F_{555}(\vec{u}_i) \\ \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} c^x_{000} & c^y_{000} & c^z_{000} \\ \vdots & \vdots & \vdots \\ c^x_{555} & c^y_{555} & c^z_{555} \end{bmatrix} = \begin{bmatrix} \vdots & \vdots & \vdots \\ p^x_i & p^y_i & p^z_i \\ \vdots & \vdots & \vdots \end{bmatrix}$$

The coefficients are then used with the 3D interpolation function given above on a new point (normalized to the same q_min and q_max) to correct any distortion.

1.5 Stereotactic Navigation

After the distortion correction function was found, the EM pivot calibration was repeated, this time with $\vec{G}_i$ that have been corrected. Pivot calibration returns a p_tip defined in a local coordinate frame, g. For every $\vec{G}$ in a frame of the pointer in EM coordinates corresponding to a fiducial location $\vec{b}_i$ in CT coordinates, the transformation from the frame points to the local coordinate frame $\mathbf{F}_G$ was found and p_tip was transformed to find the fiducial location in EM coordinates $\vec{B}_i$. The transformation $\mathbf{F}_{reg}$ was found such that

$$\vec{b}_i = \mathbf{F}_{reg} \cdot \vec{B}_i$$

Using successive frames of $\vec{G}_i$ from EM navigation, the locations of the corresponding position in CT coordinates is calculated by finding the tip in EM coordinates using p_tip and g, and then transformed to CT coordinates using $\mathbf{F}_{reg}$.

2 Algorithms

2.1 Cartesian Math

Point3d : coords (x,y,z), frame name
Frame : R (3x3 NumPy array), p (Point3d), frame name

2.2 Arun's Registration Method

Transpose vectors if necessary. Calculate average and detract from vectors. Implement Arun's method by creating the H matrix using the calculated vectors. Calculate registration frame using SVD.

2.3 Pivot Calibration

Define the local coordinate system. Create the registration frame for $\vec{p}_{tip}$, $\vec{p}_{dimple}$ at each frame by definiting a transformation between the point cloud and the local coordinate system. Calculate the registration frame using SVD with least squares. Return the calculated $\vec{p}_{dimple}$.

2.4 Distortion Correction

Normalize given distorted points. Calculate Bernstein polynomial function for the normalized, distorted points. Use SVD with least squares to compute the interpolation coefficients. Multiply Bernstein polynomial function by the coefficients to yield the undistorted points.

2.5 Stereotactic Navigation

Compute $\vec{C}_i^{exp}$. Use $\vec{C}_i^{exp}$ and $\vec{C}_i$ to calculate the Bernstein polynomial coefficients and distorted points' bounds. Correct EM pivot frames for distortion, and use to calibrate the pointer tip and EM tracker position. Correct EM fiducial frames for distortion and use to calculate new pointer location. Register these new locations to the CT fiducials to find $\mathbf{F_{reg}}$. Correct EM navigation data for distortion and use to calculate new pointer location. Transform these new locations to the CT coordinate frame using $\mathbf{F_{reg}}$.

3 Overview of Program

File Directory	
File	Description
FileIO.py	functions to read input and output1 files
GenerateOutput.py	functions to read the unknown dataset files and generate an output1 txt file
Point3d.py	class to create 3d point objects and perform cartesian math functions
Frame.py	class to do frame transformations, save frame data, and transform points.
Registration.py	an implementation of the Arun's registration method
EMPivotCalibration.py	implementation of EM tracker pivot calibration
OpticalPivotCalibration.py	implementation of optical tracker pivot calibration
DistortionCorrection.py	implementation of calculating and correcting Berstein polynomial distortion
testing.py	testing functions we used to test and debug our functions

3.1 Packages

Packages used: NumPy==1.26.0 (important functions include matrix functions and np.linalg.svd), Math (used math.comb, comes with python==3.11.0).

3.2 Code Structure

We created files for the different parts of the project.

GenerateOutput1.py and GenerateOutput2.py are the main files that should be run on input data to get output files 1 and 2, respectively.

> These files include functions to read the unknown dataset files and generate the desired output files. Output 1 includes EM pivot position, optical tracker pivot position, and expected C_is. Output 2 includes the probe tip positions.

testing.py is a script that takes an input debugging data set and outputs the error of our calculated variables compared to those given in out "-output1.txt" and "-output2.txt" files. It also includes a few component tests that we did in addition to the debugging tests.

> testRegistration() uses the given d from "-calbody.txt" and registers it to an "artificial" D. This "artificial" D was created on MATLAB via $D = \mathbf{F_D} \cdot d$ by randomly generating a rotation matrix $\mathbf{R}$ and translation vector $\vec{p}$ to create $\mathbf{F_D}$. Refer to the **Appendix** for a screenshot of this code.
> testDistortionReconstructsCexp() calculates the error of the C^{exp} that the distortion correction function yields and compares it to the given undistorted C.
> printPA2OutputErrors() prints the average calibration error of C_i^{exp}, the $\vec{p}_{EM}$ error, and the error of the probe tip positions $\vec{v}_i$ according to PA #2. printPA1OutputErrors() prints the average calibration error of C_i^{exp}, the $\vec{p}_{EM}$ error, and the $\vec{p}_{opt}$ error according to PA #1.

DistortionCorrection.py is a file containing multiple helper functions to calculate the undistorted points p from the given distorted points q.

bernstein() takes in N, k, and v and outputs the Bernstein polynomial according to:

$$B_{N,k}(v) = \begin{pmatrix} N \\ k \end{pmatrix} (1-v)^{N-k} v^k$$

bernsteinPolynomialF() uses bernstein() to create the full Bernstein polynomial function given the order, distorted points q, and the bounds of q.

$$P(c_0,\ldots,c_N;v) = \sum_{k=0}^{N} c_k \binom{N}{k} (1-v)^{N-k} v^k$$
$$= \sum_{k=0}^{N} c_k B_{N,k}(v)$$

calcDistortionCorrection() takes in the ground truth points with no distortion and the corresponding 3D points with distortion, and returns the coefficients for the Bernstein polynomials. correctDistortion() takes in 3D points with distortion and uses the coefficients to correct the distortion with Bernstein polynomials.

FileIO.py includes functions to read the calbody, calreadings, empivot, optpivot, output1 files.

For each type of file, the exact dimensions of the output arrays are specified in the python file, so that users can look at the description and understand how the array is formatted.

Point3d.py and Frame.py include classes to create 3d point objects, do transformations, save frame data, and other Cartesian math functions.

Each Point3d object stores the name of the frame it resides in in addition to its x, y, z coordinates. The base code for this class was taken off Github (cited in code) though we added several additional features (e.g., frame variable and error() function) as necessary for this scenario. Each Frame object object stores its 3D rotation matrix $\mathbf{R}$ and 3D position vector $\vec{p}$, as well as its neighboring Frames so that the whole system may be traversed from one Frame object.

Registration.py has an implementation of the Arun's registration, following the algorithm detailed above.

registrationArunMethod() takes in 2 point clouds and finds a transformation $\mathbf{F}$ such that $\mathbf{F} \cdot a = b$

EMPivotCalibration.py and OpticalPivotCalibration.py include functions to perform the EM tracker pivot calibration and optical tracker pivot calibration respectively.

pivotCalibration() takes in $\vec{G}$ corresponding to the pivot calibration frames and returns the location of the pivot in EM coordinates and the center of pointer to tip in local coordinates. GtoEM() converts a frame of $\vec{G}$ points and uses p_{tip} as well as the local coordinate frame used to computer p_{tip} to find the location of the pointer tip in EM coordinates. opticalCalibration() does the same as pivotCalibration() with the added transformation between the optical tracker and EM tracker.

4 Verification of Code

4.1 Cartesian Math

To verify the Frame and Point3d class objects functioned as intended, we created random but plausible $\vec{p}_{start,A}, \mathbf{F_A}, \mathbf{F_B}$, variables and compared our output $\vec{p}_{start,B}$ to one generated on MATLAB.

$$\vec{p}_{start,A} = \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \end{bmatrix}$$

$$\mathbf{F_A} = \begin{bmatrix} \mathbf{R_A} & \vec{p}_A \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1.5 \\ 0 & 1 & 0 & 0.8 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{F_B} = \begin{bmatrix} \mathbf{R_B} & \vec{p}_B \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} -0.2309 & -0.9699 & 0.0772 & 0.2 \\ -0.7747 & 0.1353 & -0.6177 & 0.6 \\ 0.5887 & -0.2025 & -0.7826 & 1.3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\vec{p}_{start,B} = \mathbf{F_{BA}} \cdot \vec{p}_{start,A} = \mathbf{F_B^{-1}} \cdot \mathbf{F_A} \cdot \vec{p}_{start,A}$$

4.2 Registration

We verified our Arun registration function by testing $\mathbf{F_D}$ against the first frame's D coordinates (using debugging file "a") and confirmed that $\mathbf{F_D^{-1}} \cdot \vec{D}_0 = \vec{d}_0$ on MATLAB.

Refer to **Appendix** for screenshots of outputs.

4.3 Distortion Correction and Pivot Calibration

To verify our distortion correction works as expected, we calculated the average error for C^{exp}, P_{EM}, and v with both third- ($N = 3$) and fifth- ($N = 5$) order Bernstein polynomial functions. This is because increasing the order of the Bernstein polynomial function should increase our accuracy due to the increased dimensions for interpolation. The average error between our generated C^{exp} and the actual C^{exp} of debugging data sets a-f is shown below:

Average Error between expected Cs (mm)		
Set	N=3	N=5
a	0.0050605	0.0050604
b	0.4774513	0.4774513
c	3.5317987	0.4778655
d	0.0153251	0.0153251
e	6.7552322	1.5223972
f	6.2063371	1.6825068

The N=3 and N=5 is the order of the Bernstein polynomials. It is expected that the error decreases with a higher order polynomials, and our results show this, thus we concluded that the Bernstein polynomials work as expected.

To verify our pivot calibration works as expected, we calculated the squared error $\sqrt{(g_x - c_x)^2 + (g_y - c_y)^2 + (g_z - c_z)^2}$ between our generated p_{dimple}, or g in the above equation, and the estimated post positions, c, in the output files for the debug sets a-f for both the EM probe and the optical probe. The error for both are shown below:

Error in EM Pivot Positions (mm)		
Set	N=3	N=5
a	0.0051406	0.004706
b	0.0862319	0.375100
c	3.7649278	0.048246
d	0.0062949	0.004875
e	2.3060736	0.070010
f	3.1987930	0.204703

Set	Error in OPT Pivot Positions (mm)
a	0.0019461
b	0.0035317
c	0.0038377
d	0.0075295
e	0.0059191
f	0.0056626

Average Error in Probe Tip Positions (mm)		
Set	N=3	N=5
a	0.0065570	0.00693237
b	0.2818401	0.29584308
c	1.3563927	0.02370504
d	0.0091655	0.00872486
e	3.0988400	0.13803755
f	1.3886495	0.28922103

Because of the low amount of error (less than 1 mm) in both the EM and optical tracker pivot positions and the low amount of error in the probe tip positions in CT coordinates, we verified that our pivot calibration and stereotactic navigation works as expected.

5 Results

5.1 Discussion

Looking at the errors from the debugging data sets with $N = 5$, $\vec{C}_i^{exp}$ is very accurate for data sets a and d, less so for b and c, and at its worst for e and f. Data set a does not have any noise, so it makes sense that it is the most accurate, and d only has OT jiggle, which is likely offset by the frame-wise pivot calibration. On the other hand, data sets e and f have all 3 forms of disturbance–EM distortion, EM noise, and OT jiggle–which likely compounded to create the < 2 mm error. This implies the distortion correction function could be further optimized to better account for EM distortion.

The errors for the EM pivot position remain very small across all data sets, with the largest being not even 0.4 mm. The two larger errors belong to data sets b and f, which is odd given that data sets e and f contain all 3 forms of disturbance whereas b only has EM noise. Since e's error remains close in magnitude to data set c, we assume b and f have similar values of EM noise, and that they are large enough to strongly impact the error. We then suspect that data set e has a lower EM noise in lieu of higher EM distortion and/or OT jiggle, both of which seem to be countered well by our implementation of distortion correction and pivot calibration.

Unsurprisingly, the errors in OPT pivot position were consistently lower than all other variables despite there being significantly less frames of information to calculate it with. $\vec{p}_{opt}$ was not subjected to EM distortion and EM noise the way $\vec{p}_{EM}$ was, leaving only OT jiggle as any influencing factor. From the results of our PA #1, we already know that the OT jiggle is more or less accounted for by the optical calibration's implementation (since it works frame by frame), allowing its errors to remain very small across all data sets.

There is significantly higher error in the probe tip position $\vec{v}$ for data sets b, e, and f, but it remains to a much smaller scale than seen with $\vec{C}_i^{exp}$. We believe that these errors are due to the EM Noise since only these 3 data sets have this feature. However, their lower magnitude than $\vec{C}_i^{exp}$ suggests continuous correction and readjustment of $\vec{p}_{tip}$ can help to reduce the effects of EM distortion and OT jiggle.

Arun's registration method may not be the best to handle the level of noise present in these datasets with the "OT jiggle", so possibly switching to an iterative method that converges to an optima would have been better for the noisy datasets. Another potential solution is to remove outliers before using Arun's method. The pivot calibration datasets seem to not be too affected by noise because of the low error despite the use of Arun's method for some steps in the pivot calibration, however, Arun's method was used to calculate $C_{expected}$ which may lead to propogated error in the distortion correction function if there was error in the $C_{expected}$.

5.2 Unknown outputs

Our debugging results lead us to believe that our $\vec{C}_i^{exp}$ values lie within an approximately 1.7 mm error margin, our P_{EM} within 0.4 mm error, our P_{opt} within 0.008 mm error, and our $\vec{v}$ within 0.3 mm error. For the text files, refer to the OUTPUT folder in the zip file. The results are also in the appendix section, excluding $\vec{C}_i^{exp}$ due to the sheer size of those variables.

6 Partner Work

Both Kiana and Keerthana worked on all the python files as well as the report.

7 Differences from PA1 and PA2 Report

We added the distortion correction and stereotactic navigation sections to Mathematical Approach and Algorithms. We also developed more component testing for Verification of Code, and changed the Results section to account for PA2 results.

8 Appendix

8.1 Debugging Dataset Results

8.1.1 Dataset a

Params: 27, 125, pa2-debug-a-output1.txt
EM Pivot Position: 199.16, 199.24, 208.89
OPT Pivot Position: 399.59, 391.69, 193.47
$C_i^{expected}$s excluded due to large size.

N_{frames} : 4, pa2-debug-a-output2.txt
79.23, 88.90, 70.69
65.56, 105.70, 106.85
67.09, 152.54, 131.78
100.78, 139.65, 160.30

8.1.2 Dataset b

Params: 27, 125, pa2-debug-b-output1.txt
EM Pivot Position: 202.61, 205.80, 199.74
OPT Pivot Position: 401.45, 402.11, 206.90
$C_i^{expected}$s excluded due to large size.

N_{frames} : 4, pa2-debug-b-output2.txt
126.00, 125.69, 157.95
110.87, 76.06, 161.48
132.05, 46.56, 35.12
75.74, 61.48, 134.22

8.1.3 Dataset c

Params: 27, 125, pa2-debug-c-output1.txt
EM Pivot Position: 207.86, 192.14, 205.74
OPT Pivot Position: 400.30, 396.71, 192.77
$C_i^{expected}$s excluded due to large size

N_{frames} : 4, pa2-debug-c-output2.txt
40.87, 97.35, 85.80
95.98, 63.22, 108.59
48.31, 150.57, 105.32
154.70, 135.34, 52.94

8.1.4 Dataset d

Params: 27, 125, pa2-debug-d-output1.txt
EM Pivot Position: 197.08, 202.27, 209.02
OPT Pivot Position: 402.59, 394.38, 204.58
$C_i^{expected}$s excluded due to large size.

N_{frames} : 4, pa2-debug-d-output2.txt
165.54, 94.25, 158.84
106.21, 59.09, 158.43
71.85, 94.91, 61.70
158.91, 105.44, 158.66

8.1.5 Dataset e

Params: 27, 125, pa2-debug-e-output1.txt
EM Pivot Position: 196.62, 201.82, 205.83
OPT Pivot Position: 407.78, 404.12, 209.43
$C_i^{expected}$s excluded due to large size.

N_{frames} : 4, pa2-debug-e-output2.txt
109.62, 169.33, 140.75
85.87, 120.19, 140.02
43.05, 129.16, 92.05
43.44, 152.94, 54.40

8.1.6 Dataset f

Params: 27, 125, pa2-debug-f-output1.txt
EM Pivot Position: 193.07, 195.44, 196.48
OPT Pivot Position: 396.86, 404.76, 202.51
$C_i^{expected}$s excluded due to large size.

N_{frames} : 4, pa2-debug-f-output2.txt
46.72, 52.32, 59.18
55.65, 85.94, 71.38
61.47, 125.81, 154.84
112.95, 40.15, 156.46

8.2 Unknown Dataset Results

8.2.1 Dataset g

Params: 27, 125, pa2-unknown-g-output1.txt
EM Pivot Position: 191.43, 194.08, 190.28
OPT Pivot Position: 400.99, 397.27, 200.26
$C_i^{expected}$s excluded due to large size.

N_{frames} : 4, pa2-unknown-g-output2.txt
82.71, 163.25, 169.78
93.36, 136.48, 140.72
49.12, 47.29, 76.59
71.27, 133.21, 161.19

8.2.2 Dataset h

Params: 27, 125, pa2-unknown-h-output1.txt
EM Pivot Position: 190.55, 195.18, 207.73
OPT Pivot Position: 391.84, 396.79, 197.44
$C_i^{expected}$s due to large size.

N_{frames} : 4, pa2-unknown-h-output2.txt
55.54, 61.83, 78.33
95.05, 147.83, 88.83
145.41, 40.32, 168.49
132.87, 117.58, 123.89

8.2.3 Dataset i

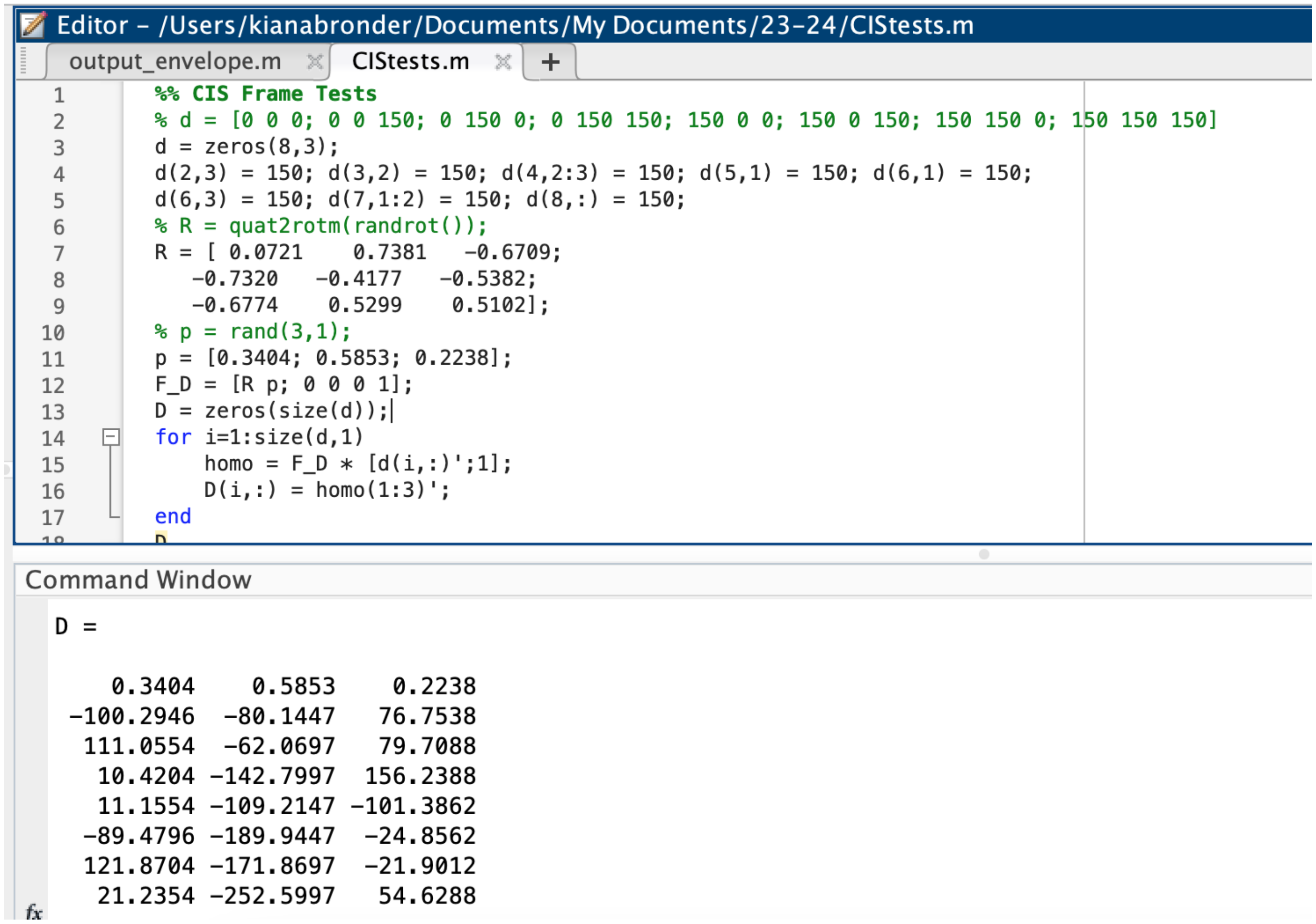
Params: 27, 125, pa2-unknown-i-output1.txt
EM Pivot Position: 209.25, 209.44, 204.19
OPT Pivot Position: 397.71, 409.68, 198.73
$C_i^{expected}$s excluded due to large size. N_{frames} : 4, pa2-unknown-i-output2.txt
155.62, 70.08, 47.76
40.03, 122.07, 171.51
60.90, 91.81, 153.48
109.76, 67.21, 27.92

8.2.4 Dataset j

Params: 27, 125, pa2-unknown-j-output1.txt
EM Pivot Position: 204.30, 193.33, 199.46
OPT Pivot Position: 407.88, 391.90, 202.10
$C_i^{expected}$s excluded due to large size

N_{frames} : 4, pa2-unknown-j-output2.txt
72.20, 83.15, 72.29
66.08, 43.21, 73.45
163.65, 44.87, 125.82
162.80, 59.48, 152.47

8.3 Debugging Examples



Editor – /Users/kianabronder/Documents/My Documents/23-24/CIStests.m

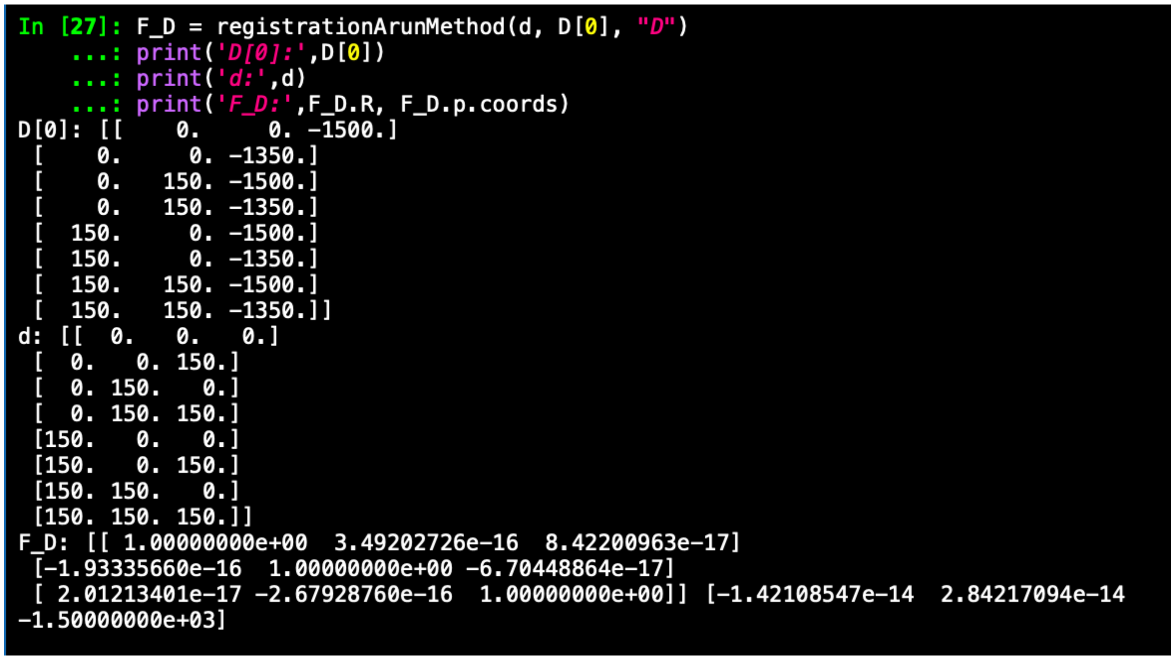
output_envelope.m CIStests.m

```
%% CIS Frame Tests
% d = [0 0 0; 0 0 150; 0 150 0; 0 150 150; 150 0 0; 150 0 150; 150 150 0; 150 150 150]
d = zeros(8,3);
d(2,3) = 150; d(3,2) = 150; d(4,2:3) = 150; d(5,1) = 150; d(6,1) = 150;
d(6,3) = 150; d(7,1:2) = 150; d(8,:) = 150;
% R = quat2rotm(randrot());
R = [ 0.0721    0.7381   -0.6709;
   -0.7320   -0.4177   -0.5382;
   -0.6774    0.5299    0.5102];
% p = rand(3,1);
p = [0.3404; 0.5853; 0.2238];
F_D = [R p; 0 0 0 1];
D = zeros(size(d));
for i=1:size(d,1)
    homo = F_D * [d(i,:)';1];
    D(i,:) = homo(1:3)';
end
```

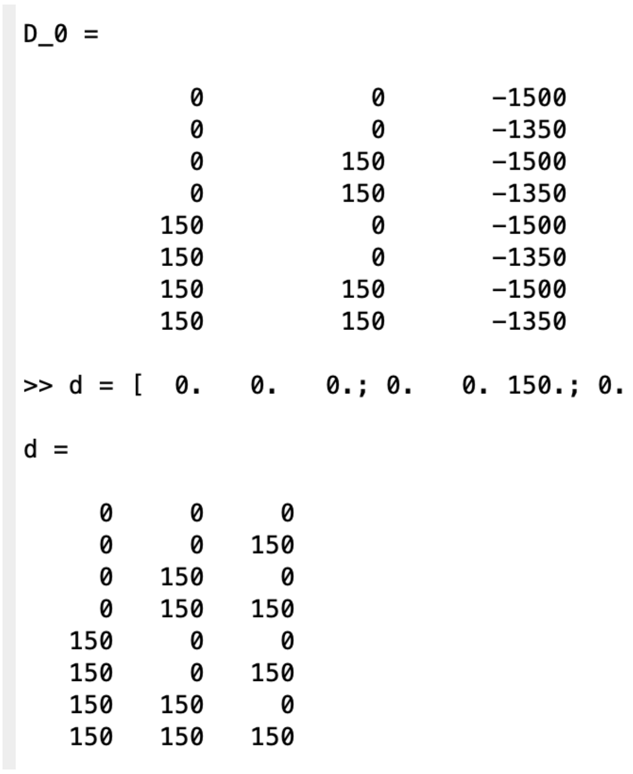
Command Window

```
D =

    0.3404    0.5853    0.2238
 -100.2946  -80.1447   76.7538
  111.0554  -62.0697   79.7088
   10.4204 -142.7997  156.2388
   11.1554 -109.2147 -101.3862
  -89.4796 -189.9447  -24.8562
  121.8704 -171.8697  -21.9012
   21.2354 -252.5997   54.6288
```

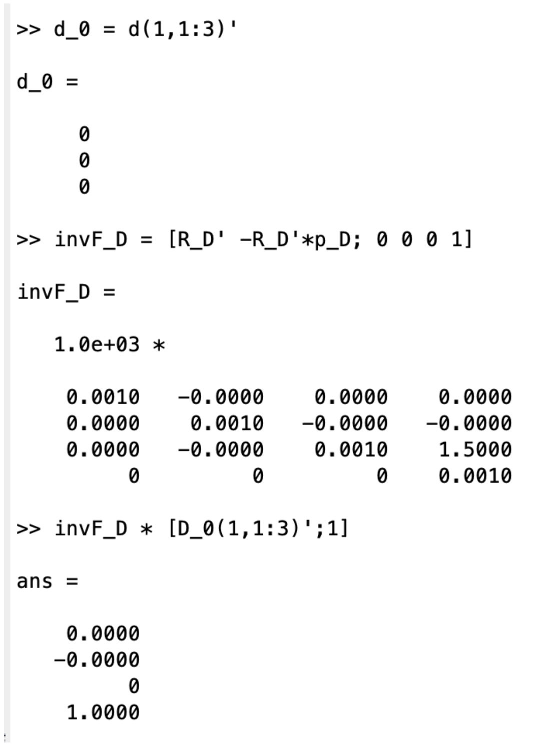
Kiana Bronder and Keerthana Thammana



```
In [27]: F_D = registrationArunMethod(d, D[0], "D")
    ...: print('D[0]:',D[0])
    ...: print('d:',d)
    ...: print('F_D:',F_D.R, F_D.p.coords)
D[0]: [[    0.     0. -1500.]
 [    0.     0. -1350.]
 [    0.   150. -1500.]
 [    0.   150. -1350.]
 [  150.     0. -1500.]
 [  150.     0. -1350.]
 [  150.   150. -1500.]
 [  150.   150. -1350.]]
d: [[  0.   0.   0.]
 [  0.   0. 150.]
 [  0. 150.   0.]
 [  0. 150. 150.]
 [150.   0.   0.]
 [150.   0. 150.]
 [150. 150.   0.]
 [150. 150. 150.]]
F_D: [[ 1.00000000e+00  3.49202726e-16  8.42200963e-17]
 [-1.93335660e-16  1.00000000e+00 -6.70448864e-17]
 [ 2.01213401e-17 -2.67928760e-16  1.00000000e+00]] [-1.42108547e-14  2.84217094e-14
-1.50000000e+03]
```

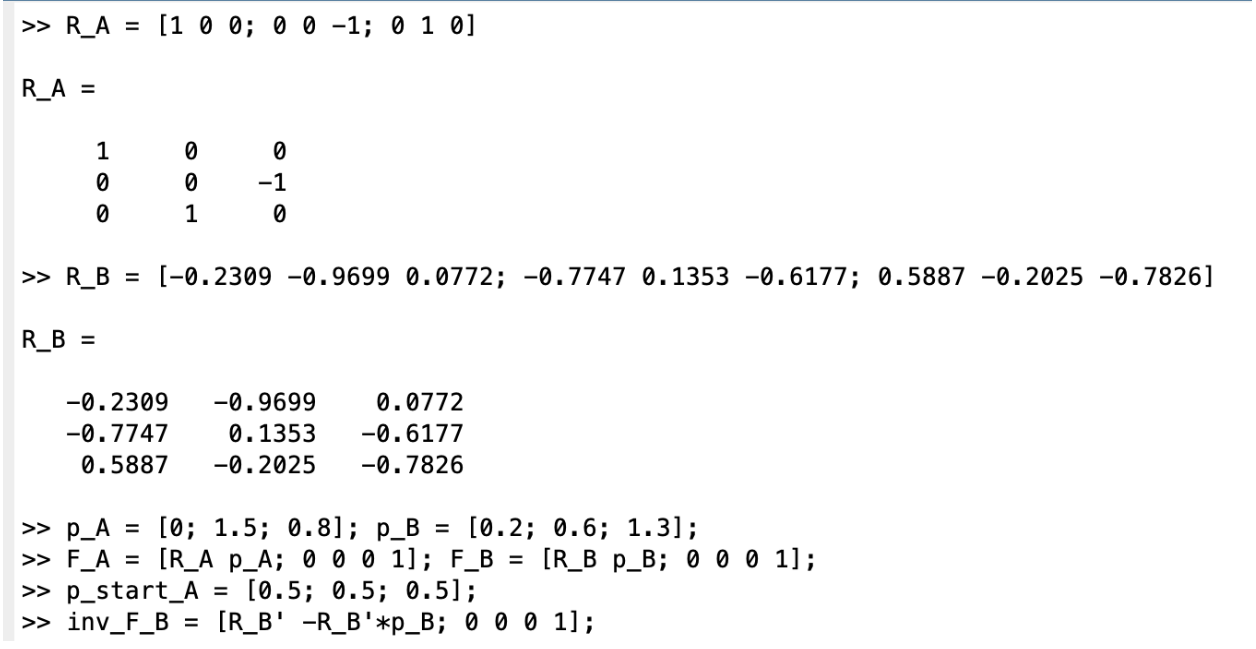


```
D_0 =

             0             0         -1500
             0             0         -1350
             0           150         -1500
             0           150         -1350
           150             0         -1500
           150             0         -1350
           150           150         -1500
           150           150         -1350

>> d = [  0.   0.   0.; 0.   0. 150.; 0.

d =

     0     0     0
     0     0   150
     0   150     0
     0   150   150
   150     0     0
   150     0   150
   150   150     0
   150   150   150
```

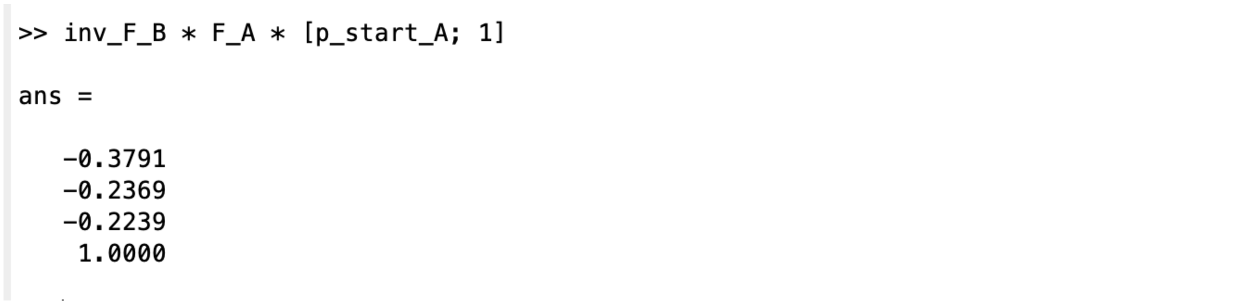


```
>> d_0 = d(1,1:3)'

d_0 =

     0
     0
     0

>> invF_D = [R_D' -R_D'*p_D; 0 0 0 1]

invF_D =

   1.0e+03 *

    0.0010   -0.0000    0.0000    0.0000
    0.0000    0.0010   -0.0000   -0.0000
    0.0000   -0.0000    0.0010    1.5000
         0         0         0    0.0010

>> invF_D * [D_0(1,1:3)';1]

ans =

    0.0000
   -0.0000
         0
    1.0000
```

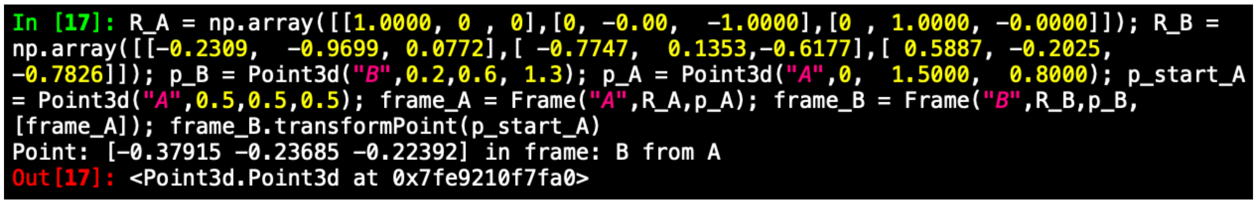
ans == d_0 → verified



```
>> R_A = [1 0 0; 0 0 -1; 0 1 0]

R_A =

     1     0     0
     0     0    -1
     0     1     0

>> R_B = [-0.2309 -0.9699 0.0772; -0.7747 0.1353 -0.6177; 0.5887 -0.2025 -0.7826]

R_B =

   -0.2309   -0.9699    0.0772
   -0.7747    0.1353   -0.6177
    0.5887   -0.2025   -0.7826

>> p_A = [0; 1.5; 0.8]; p_B = [0.2; 0.6; 1.3];
>> F_A = [R_A p_A; 0 0 0 1]; F_B = [R_B p_B; 0 0 0 1];
>> p_start_A = [0.5; 0.5; 0.5];
>> inv_F_B = [R_B' -R_B'*p_B; 0 0 0 1];
```



```
>> inv_F_B * F_A * [p_start_A; 1]

ans =

   -0.3791
   -0.2369
   -0.2239
    1.0000
```



```
In [17]: R_A = np.array([[1.0000, 0 , 0],[0, -0.00,  -1.0000],[0 , 1.0000, -0.0000]]); R_B =
np.array([[-0.2309,  -0.9699, 0.0772],[ -0.7747,  0.1353,-0.6177],[ 0.5887, -0.2025,
-0.7826]]); p_B = Point3d("B",0.2,0.6, 1.3); p_A = Point3d("A",0,  1.5000,  0.8000); p_start_A
= Point3d("A",0.5,0.5,0.5); frame_A = Frame("A",R_A,p_A); frame_B = Frame("B",R_B,p_B,
[frame_A]); frame_B.transformPoint(p_start_A)
Point: [-0.37915 -0.23685 -0.22392] in frame: B from A
Out[17]: <Point3d.Point3d at 0x7fe9210f7fa0>
```