Programming Assignments 1 & 2 601.455 and 601/655 Fall 2025

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

Score Sheet

Name 1	Justin Wang
Email	jwang519@jh.edu
Other contact information (optional)	
Name 2	Alan You
Email	ayou1@jh.edu
Other contact information (optional)	
Signature (required)	I(we)havefollowedtherules in completing this assignment JW ______________________________ AY ______________________________

Grade Factor		
Program (40)		
Design and overall program structure	2	
Reusability and modularity	0	
Clarity of documentation and programming	1	
Results (15)	0	
Correctness and completeness	15	
Report (45)	0	
Description of formulation and algorithmic approach	15	
Overview of program	10	
Discussion of validation approach	10	
Discussion of results	10	
TOTAL	100	

Mathematical and Algorithmic Approach

Cartesian Math Package for 3D coordinate computaions (Q1):

We decided to use numpy as the basis for our math, as it is a well-known and trustworthy python package.

Point Cloud Registration (Q2):

With two point clouds $\boldsymbol{a_i}, \boldsymbol{b_i} \in \mathbb{R}^3 \; ; \; i = 1, \ldots, n \in \mathbb{Z}$

Our goal is to find a frame transformation $F = [R, \boldsymbol{p}]$ such that $\boldsymbol{b_i} = F \cdot \boldsymbol{a_i} = R \cdot \boldsymbol{a_i} + \boldsymbol{p}$

The error or distance between corresponding points within the point clouds can be written as a least squares problem:

1. For each corresponding point:

$$e(R, \boldsymbol{p}) = \|\boldsymbol{b} - (R\boldsymbol{a} + \boldsymbol{p})\|^2$$

2. To minimize the error, we differentiate and set to zero:

$$\frac{\partial e(R, \boldsymbol{p})}{\partial \boldsymbol{p}} = -2(\boldsymbol{b} - R\boldsymbol{a} - \boldsymbol{p}) = 0$$

Solving for $\boldsymbol{p}$ allows us to isolate translation away from rotation. $\boldsymbol{p} = \boldsymbol{b} - R\boldsymbol{a}$

Now, all that is left is to solve for R

1. We remove parameter $\boldsymbol{p}$ from the error equation:

$$e(R) = \|\boldsymbol{b} - (R\boldsymbol{a} + (\boldsymbol{b} - R\boldsymbol{a})\|^2$$

2. Notice that this equals 0. This makes sense as rotation between two individual points cannot be resolved. This is why we need sufficient points in each point cloud.
3. Lets add in the rest of the point clouds:
   a. We start by centering the two point clouds, ie removing the translational aspect, as explained by Taylor[1].

$$\boldsymbol{a_i} = \boldsymbol{a_i} - \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{a_i}\,, \qquad \boldsymbol{b_i} = \boldsymbol{b_i} - \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{b}$$

   b. We then rewrite out error function considering all corresponding points.

$$e(R) = \frac{1}{n}\sum_{i=1}^{n} \|\boldsymbol{b_i} - (R\boldsymbol{a_i})\|^2$$

4. The Kabsch algorithm solves the error minimizing least squares function using SVD, as explained by Stachniss and Lawrence, et al. It is also referenced in Taylor's slides from Arun, et al.
    a. Create a cross-covariance matrix
    b. Compute SVD
    c. Solve for R
    d. Verify $R \in SO(3)$

The algorithmic implementation directly follows steps 3 and 4, using the numpy SVD library for solving the axes of correlation between $\boldsymbol{a_i}$ and $\boldsymbol{b_i}$

Pivot Calibration (Q3):

The goal here is that given multiple point clouds taken from the trackers on a tool while rotating it on its tip about a fixed pivot point, we can find the location of the pivot point in the tracker base's frame, as well as the offset of the tool tip in the tool frame.

1. First step is data collection. The setup describes each of the tool's having trackers in the handle, which is what the trackers (both EM and optical) get readings of. The one important thing to note here is that every reading is taken with the tool tip positioned in the same pivot point, which is fixed in the tracker base's frame. All raw data was provided as part of the assignment.
2. For each frame, we use the point cloud registration algorithm described previously to find the frame transforms from the tracker base frame to tool frame. Let $\boldsymbol{F_1} \dots \boldsymbol{F_j}$ be the calculated frame transforms from $\boldsymbol{j}$ point cloud registrations of the same set of tool trackers, where $\boldsymbol{F_i} = [R_i, \boldsymbol{p_i}]$.
3. Let

   $\boldsymbol{p_{world}}$ = position of tool tip in world frame

   $\boldsymbol{p_{tip}}$ = position of tool tip in the tool frame

   $\boldsymbol{p_{pivot}}$ = position of the pivot in the world frame

   We can see that for any of the calculated frame transforms,

   $$\boldsymbol{p_{world}} = R_i \cdot \boldsymbol{p_{tip}} + \boldsymbol{p_i}$$

   But from out data collection procedure, we can see that $\boldsymbol{p_{world}} = \boldsymbol{p_{pivot}}$, which gives

   $$\boldsymbol{p_{pivot}} = R_i \cdot \boldsymbol{p_{tip}} + \boldsymbol{p_i}$$

   Rearranging:

   $$R_i \cdot \boldsymbol{p_{tip}} - \boldsymbol{p_{pivot}} = -\boldsymbol{p_i}$$

   After computing j unique frame transforms, we can stack the measurements together to arrive at this system of equations:

$$\begin{bmatrix} R_1 & -I \\ \cdots & \cdots \\ R_j & -I \end{bmatrix} \begin{bmatrix} p_{tip} \\ p_{pivot} \end{bmatrix} = \begin{bmatrix} p_1 \\ \cdots \\ p_j \end{bmatrix}$$

We can see that to solve for p_{tip} and p_{pivot}, we just need a way to solve this $Ax = b$ problem.

4. Since this is a linear system, we can use a least squares method to solve x. In our case, we decided to use SVD based least squares solver, which outputs the final values of $\boldsymbol{p_{tip}}$ and $\boldsymbol{p_{pivot}}$

Finding Expected Calibration Object (Q4):

The goal of this part is to find two transformations: F_D and F_A. Both can be easily found using the PCR algorithm.

1. For F_D: Register PC $\boldsymbol{d_i}$ from "calbody.txt" and frames D_i from "calreadings.txt"
2. For F_A: Register PC $\boldsymbol{a_i}$ from "calbody.txt" and frames A_i from "calreadings.txt"
3. Then, we find $\boldsymbol{C}_i = F_D^{-1} F_A \boldsymbol{c}_i$

The algorithmic implementation uses a loop to iterate through each frame, computing a unique $F_D, F_A, F_C = F_D^{-1} F_A$ for each frame. That is then applied to the calbody position $\boldsymbol{c_i}$ so we can visualize the calbody "move" between frames.

EM Probe Pivot Calibration (Q5):

The goal of this problem is to perform pivot calibration on readings taken from the EM trackers on one of the tools.

1. First, given the data, we must parse it to get all of the readings into the proper format to allow point cloud registration for each frame. From each "empivot.txt", we construct G_all, which contains all N frames of tracker coordinates in EM base frame. We also declare a ProbeCalibration object, which we named emprobe to make this process more intuitive.
2. To find the tracker coordinates in tool frame, we used to recommended approach, which is to take the first frame of tracker coordinates in EM frame, take the centroid of it as the origin of the tool frame, then subtract the coordinates of the centroid from all the other tracker coordinates in EM frame to find the tracker coordinates relative to the tool origin/in the tool frame. We save these coordinates in the local_frame_points attribute of the emprobe object.
3. Now that we have N point clouds of the EM tool trackers in EM base frame stored in G_all and a EM tool trackers in tool frame (which stay the same across all N frames),

we can perform PCR to determine F_G, the transformations from EM base frame to tool frame, for all N frames. We store all N F_G in T_{all}.

4. Now that we have N HTMs collected from a pivot calibration procedure in T_{all}, we can feed T_{all} into the pivot calibration function described above to achieve the final $\boldsymbol{p}_{tip}$ and $\boldsymbol{p}_{pivot}$ as desired.

Optical Probe Pivot Calibration (Q6):

The goal of this problem is to perform pivot calibration on readings taken from the optical trackers on one of the tools. Much of this is the same as in Q5.

1. First, given the data, we must parse it to get all the readings into the proper format to allow point cloud registration for each frame. From each "optpivot.txt", we construct H_all, which contains all N frames of tracker coordinates in EM base frame. We also construct D_all, which contains the locations of the optical trackers on the EM base. This is necessary for properly transforming the optical trackers from optical base frame to EM base frame for each measurement. We also declared a ProbeCalibration object, which we named optprobe to make this process more intuitive.
2. To find the tracker coordinates in tool frame, we used to recommended approach, which is to take the first frame of tracker coordinates in optical frame, take the centroid of it as the origin of the tool frame, then subtract the coordinates of the centroid from all the other tracker coordinates in optical frame to find the tracker coordinates relative to the tool origin/in the tool frame. We save these coordinates in the local_frame_points attribute of the optprobe object.
3. Now that we have N point clouds of the optical tool trackers in optical base frame stored in H_all, optical tool trackers in tool frame (which stay the same across all N frames), and D_all which allows us to find the transformation F_D from optical base frame to EM base frame for each of the readings, we can perform PCR to determine F_G, the transformations from optical base frame to tool frame in EM base frame coordinates, for all N frames. The only difference from the procedure from Q5 is that while we iterate over H_all, we must first find the associated F_D for each reading, and apply that F_D to the optical tracking markers to get H_em, which is a single readings in H_all but in EM base frame coordinates. We use H_em to calculate each of the F_G. We store all N F_G in T_{all}.
4. Now that we have N HTMs collected from a pivot calibration procedure in T_{all}, we can feed T_{all} into the pivot calibration function described above to achieve the final $\boldsymbol{p}_{tip}$ and $\boldsymbol{p}_{pivot}$ as desired.

Structure of Code

The source code is structured in a way that promotes code reuse and easy debugging. As each algorithm was developed, multiple helper scripts were created under the ./utils folder:

- ./utils/parse.py
    - Contains functions that each segment the given file and dataset type into the frame/point-containing blocks outlined in the header of each dataset.
- ./utils/plot.py
    - Contains functions to plot point clouds, offering a visual guide for debugging transformation chains.
- ./utils/calculate_errors.py
    - Contains functions meant to calculate a variety of errors for difference use cases (RMS, difference in HTM, min/max/mean). This allows for quick debugging and additional validation of our algorithms.
- ./utils/write_out.py
    - Formats the computed data into the desired output format for this assignment into out/pa1-output-1.txt
- ./utils/calibrate.py
    - Contains the ProbeCalirbation object which is designed to allow intuitive pivot calibration of a tool starting with point clouds. It owns the functions for PCR and pivot calibration, and has two fields: one for the name of the tool, one for the location of the tracking markers on the tool in tool frame.

To validate our algorithms, testing scripts are written in ./tests. We included graphs to help visualize the results of PCR and pivot calibration, as well as the RMSE.

For ease of grading, questions 4, 5, 6 are fully contained in main.py, with the final output text file being generated at the end. The comments clearly delineate the code between each question, with the code for each of the questions' sections being fully self-contained.

Testing and Validation

Validating PCR and Pivot Calibration

Validating the PCR and Pivot Calibration algorithms is a must. Our method of validation follows a general ethos of creating a known transformation (F_{ab} for PCR, F_{tip} for pivot) and generating two identical data sets. Then, we apply the known transformation to one. If via

PCR or Pivot Calibration, we can solve for the known transformation again, below an error threshold, we have validated the algorithms.

The error transformation, angular error, and translation error are calculated using the formulas below:

$$\Delta F = F_{known}^{-1} \cdot F_{estimate} = [\Delta R, \Delta \boldsymbol{p}], \qquad e_\theta = \cos^{-1} \frac{tr(\Delta R) - 1}{2}, \qquad e_t = \|\Delta \boldsymbol{p}\|$$

Since pivot calibration only returns the translational component of the frame transformation, we only consider $\Delta \boldsymbol{p}$ when validating pivot.

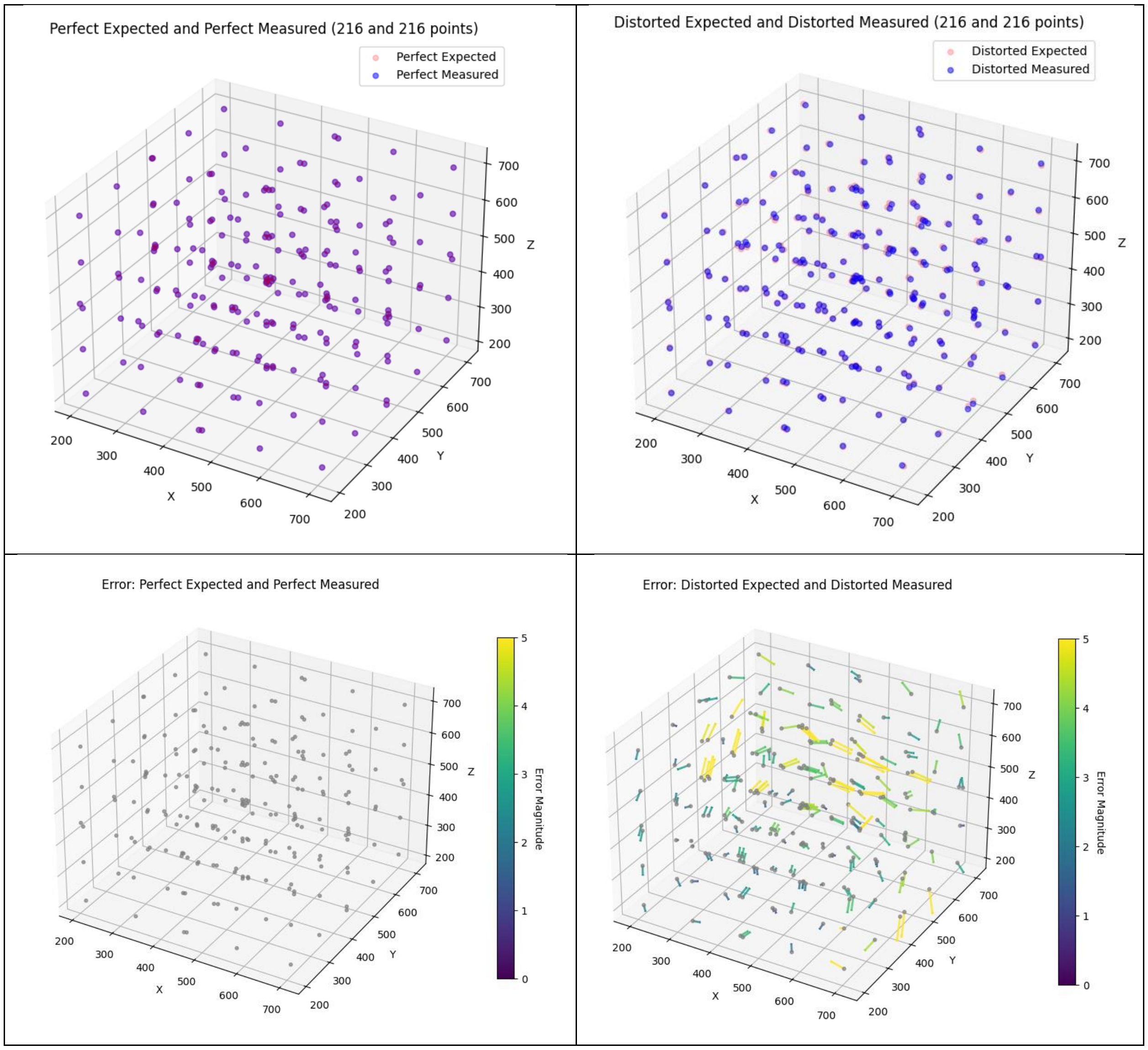
Table. Difference			
PCR		**Pivot Calibration**	
Angle Error ($\boldsymbol{\theta}$)	2.1073424255447017e-08	**Tip Trans Error (mm)**	2.2842349715679205e-15
Trans Error (mm)	3.5214902285927624e-14	**Pivot Trans Error (mm)**	1.6764000044290905e-15

Validating Expected Calibration Object

For the perfect (a) set and distorted (c) set, we can compare the "calbody" and transformed "calreadings" data sets. We can compute an RMS error that summarizes the general alignment of the transformed and measured point clouds.

Table. Expected Calibration Object Error		
Stat	**Perfect Set (a) (mm)**	**Distorted Set (c) (mm)**
Mean	0.004673	3.510549
RMS	0.005029	3.778813
Std	0.001858	1.398385
Min	0.000930	1.016064
Max	0.009547	7.094743

Table. Perfect vs Distorted Sets comparing Expected vs Measured	
Perfect Set (a)	**Distorted Set (c)**

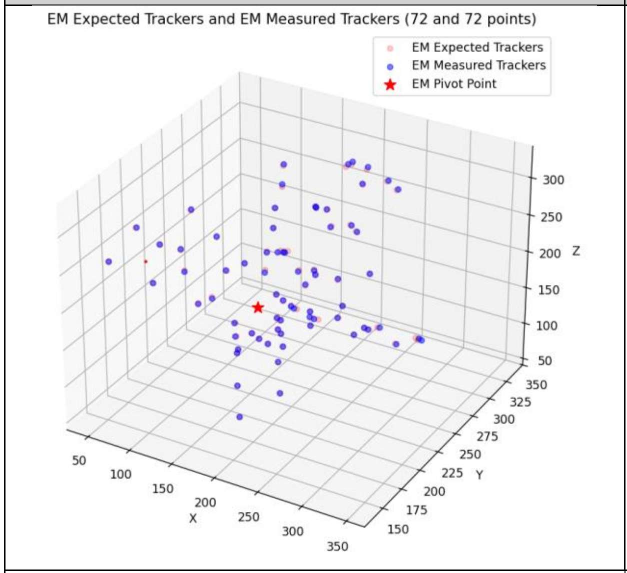
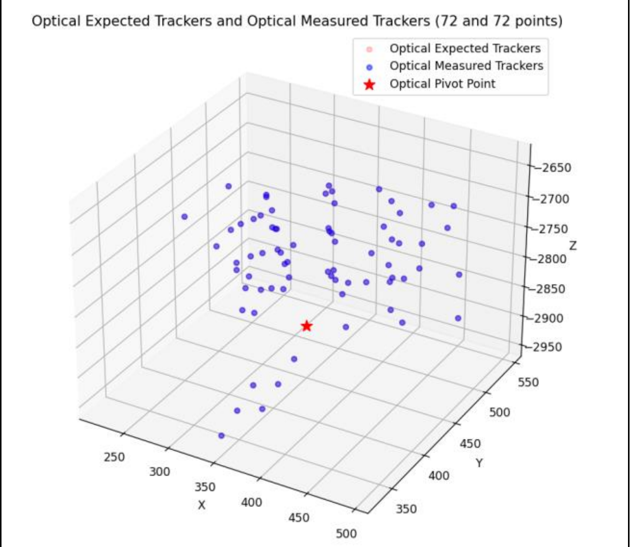
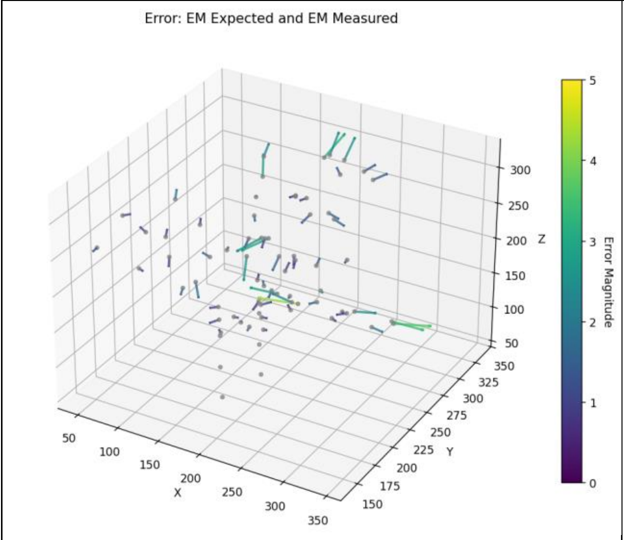
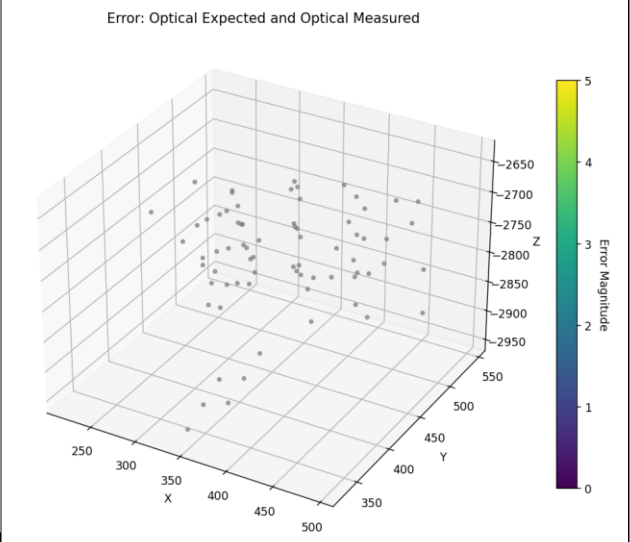


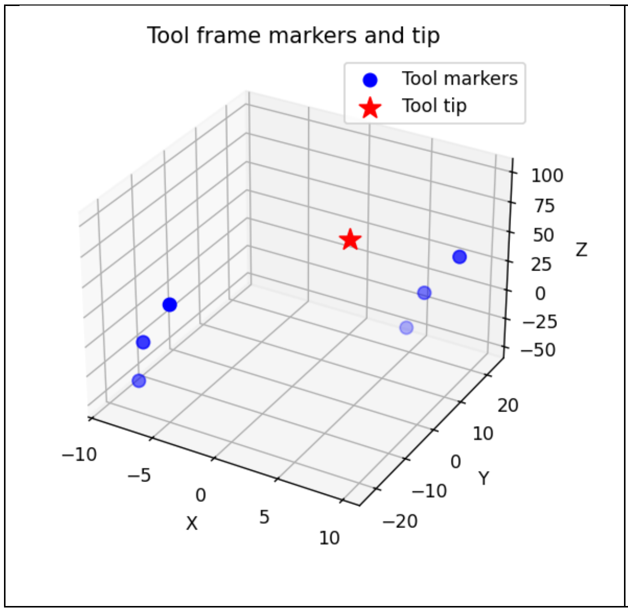
*The shape of the distorted error may give insight into accounting to distortion in PA2.

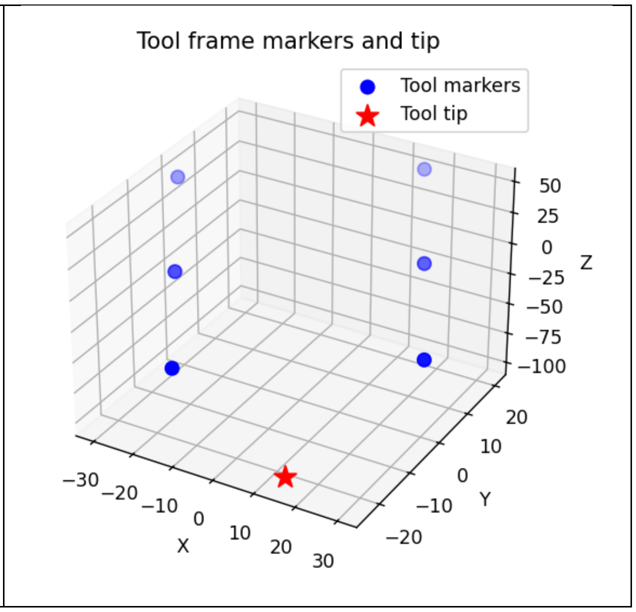
Validating Pivot Calibration

Since Pivot Calibration allows us to complete the transformation chain to find the actual tip in EM or opt coordinates. We compute an expected tip position and compare it to the measured to validate the completed transformation chain is correct.

Table. Pivot Calibration Transformation Chain Error		
Stat	**EM Tracker (mm)**	**Optical Tracker (mm)**
Mean	1.327943	0.005136
RMS	1.611128	0.005814
Std	0.912305	0.002725
Min	0.000000	0.000000
Max	4.220601	0.010552

Table. EM vs Optical Pivot Calibration	
EM Tracker	**Optical Tracker**
	
	





Discussion of Results

Overall, our PCR and Pivot Calibration algorithms produced very good results. On "perfect" datasets (undistorted, no noise), we have registration errors that are approaching zero. Ther exists far more error on the distorted sets. However, since our algorithms are validated to be working with perfect sets, any of the remaining differences can be attributed fully to the distortion. This allows us to confidently segment the distortion as the registration error to model and compensate for in PA2. Plotting the data provided a useful visual guide on translational pose differences. Specifically, plotting the error vectors on the distorted sets gives us insight on the "direction" of the distortion, which will be helpful in when curve-fitting the distortion PA2.

Our data validation and testing method was robust, but there are concerns that our sample size is too small. We initially considered looking at edge cases for the PCR and pivot calibration algorithms with extreme transformations, however after research at the literature and mathematical structure of the algorithms, we found that this was unnecessary. The only cases that could make these algorithms fail is insufficient and degenerate data sets. PCR requires non-colinear, sufficiently large, and non-reflective data sets while Pivot Calibration required sufficiently large and diverse data sets. We found that the input data provided this well.

Alan and Justin both contribute equally to the research and mathematical development of the PCR and pivot calibration procedures. Alan implemented the final PCR algorithm and

Justin implemented the final Pivot Calibration Algorithm. The ./util helper functions were created by each partner as needed.

Works Cited

[1] Taylor, R. H. (n.d.). *Point cloud to point cloud rigid transformations* [PowerPoint slides]. Computer Integrated Surgery I (601.455/655), Johns Hopkins University, Whiting School of Engineering. Laboratory for Computational Sensing and Robotics.

[2] Murray, Richard M. (2012). A mathematical introduction to robotic manipulation. CRC Pr I Llc 2012.

[3] Stachniss, C. (2021). *ICP & Point Cloud Registration – Part 1: Known data association & SVD* [Video]. YouTube.

[4] Lawrence, J., Bernal, J., & Witzgall, C. (2019, October 9). *A purely algebraic justification of the Kabsch-Umeyama algorithm*. *Journal of Research of the National Institute of Standards and Technology, 124*, 124028.

[5] K. Arun, et. al., IEEE PAMI, Vol 9, no 5, pp 698-700, Sept 1987

[6] Taylor, R. H. (n.d.). *Calibration* [PowerPoint slides]. Computer Integrated Surgery I (601.455/655), Johns Hopkins University, Whiting School of Engineering. Laboratory for Computational Sensing and Robotics.

[7] Harris, C.R., Millman, K.J., van der Walt, S.J. et al. *Array programming with NumPy*. Nature 585, 357–362 (2020.

[8] McKinney, W. (2010). Data structures for statistical computing in python. In S. van der Walt & J. Millman (Eds.), Proceedings of the 9th Python in Science Conference (pp. 51–56).

[9] J. D. Hunter, "Matplotlib: A 2D Graphics Environment", Computing in Science & Engineering, vol. 9, no. 3, pp. 90-95, 2007PCR: Taylor Slides and Kabsch SVD, ICP & Point Cloud Registration - Part 1: Known Data Association & SVD (Cyrill Stachniss, 2021), A Purely Algebraic Justification of the Kabsch-Umeyama Algorithm, K. Arun, et. al., IEEE PAMI, Vol 9, no 5, pp 698-700, Sept 1987