


**Programming Assignments 1 & 2 601.455 and 601/655 Fall 2024**  
**Please also indicate which section(s) you are in (one of each is OK)**

**Score Sheet**

Name 1	Benjamin Miller (601.455)
Email	bmill119@jh.edu
Other contact information (optional)	
Name 2	Aryavrat Gupta (601.455)
Email	agupt110@jhu.edu
Other contact information (optional)	
Signature (required)	<p>I (we) have followed the rules in completing this assignment</p> <p align="center">               _____              _____         </p>

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	

## Introduction

The primary objective of Programming Assignment #1 was to develop methods for handling 3D points, enabling 3D point cloud registration and pivot calibration. Ultimately, such methods would be applied to calibrate the location of an optical probe and EM probe in EM coordinates, as well as compute the expected coordinates of a calibration object in EM coordinates. This report outlines the mathematical foundations, algorithmic implementation, program structure, validation steps, and analysis of the obtained results.

## Mathematical Approach

Our package utilizes the NumPy Python library to handle 3D points and to perform operations such as rotations and translations<sup>1</sup>. Using NumPy, 3D points can be represented as arrays and coordinate sets can be stored as arrays of size  $n \times 3$ , where  $n$  is the number of depth calibration points. Additionally, transformation matrices are stored as  $4 \times 4$  arrays in homogenous coordinates and are of the form:

$$T = \begin{bmatrix} R_{3 \times 3} & p_{3 \times 1} \\ 0 & 1 \end{bmatrix}$$

Where  $R$  is a  $3 \times 3$  array containing the rotation matrix and the translation vector  $p$  is  $3 \times 1$ . Therefore, in order to apply a rotation matrix to a point set, the point set is expanded to  $4 \times 4$ . The same will be true for an inverse frame transformation of the form:

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

Using NumPy arrays allows for efficient mathematical operations, such as vector algebra, matrix dot products, and singular value decomposition (SVD).

To align 3D point sets, we implement a point set registration algorithm. First, we calculate the average of both point sets. Let *source* and *target* be the two point sets. We compute their averages as:

$$\bar{p}_{\text{source}} = \frac{1}{N} \sum_{i=1}^N p_i,$$

$$\bar{p}_{\text{target}} = \frac{1}{M} \sum_{j=1}^M q_j$$

Here,  $N$  and  $M$  are the number of points in the source and target sets, respectively. The points are then centered by subtracting the appropriate average.

To solve for the optimal rotation matrix, we implement Arun's method<sup>2</sup>. Therefore, we compute the covariance matrix  $H$  of the centered point sets, given by:

$$H = \sum_{i=1}^N (p_i - \bar{p}_{\text{source}})(q_i - \bar{p}_{\text{target}})^T$$

We then apply SVD to the covariance matrix  $H$ :

$$H = USV^T$$

From the matrices  $U$  and  $V$ , we derive the optimal rotation matrix  $R$  using:

$$R = VU^T$$

We ensure that the rotation is proper by checking the determinant of  $R$ ; if “ $\det(R) < 0$ ”, we modify  $V$  to enforce a valid rotation.

The translation vector  $p$  can be computed as:

$$p = \bar{p}_{\text{target}} - R\bar{p}_{\text{source}}$$

To quantify the registration quality, we compute the RMSE between the target and transformed points:

$$\text{RMSE} = \sqrt{\frac{1}{N} \sum_{i=1}^N \|q_i - (Rp_i + p)\|^2}$$

The pivot calibration method aims to determine the coordinates of a  $p_{\text{post}}$  based on several frames of measured coordinates. For  $N$  frames, we initialize matrices  $A$  and vector  $b$  for a least squares optimization problem. Each frame contributes to the formulation of the augmented matrix  $[R \mid -I]$ , where  $I$  is the identity matrix and  $R$  is determined via point set registration, in this case with  $g_0$  as the source. This is constructed as follows:

$$A_k = [R_k \mid -I]$$

Each frame's  $A_k$  is vertically stacked to form a composite matrix  $A$ :

$$A = \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_N \end{bmatrix}$$

The vector  $b$  is assembled by stacking the negated translation vectors:

$$b = \begin{bmatrix} -p_1 \\ -p_2 \\ \vdots \\ -p_N \end{bmatrix}$$

The calibration is accomplished by solving the equation  $Ax = b$  where  $x = [t_g; p_{\text{dimple}}]$ .

The vector  $x$  is computed using the pseudoinverse method:

$$x = (A^T A)^{-1} A^T b$$

The solution vector  $x$  contains the optimized translation vector  $t_g$  and the pivot point  $p_{\text{dimple}}$ :

$$t_g = x[:3], \quad p_{\text{dimple}} = x[3:]$$

Finally, we compute the residuals to assess the calibration quality as:

$$\text{residuals} = A \cdot x - b$$

The square of the norm of the residuals serves as the objective function to evaluate the calibration's accuracy.

## Algorithm Overview

In order to compute the expected values for the calibration object in EM tracker coordinates, we first computed the transformation between the optical tracker and the EM tracker coordinate systems. Algorithmically, this involved computing the transformation  $F_D$ , which is expressed as:

$$T_D = \begin{bmatrix} R_D & p_D \\ 0 & 1 \end{bmatrix}$$

Where  $R_D$  is the  $3 \times 3$  rotation matrix and  $p_D$  is the  $3 \times 1$  translation vector. The transformation was determined using point set registration. Following this, we computed the transformation  $F_A$  in the same fashion. By applying point set registration to pairs of corresponding points, we derived the rotation and translation necessary to align the calibration object with the optical tracker. With both transformations established, we proceeded to compute the expected coordinates of points in the calibration object frame in the EM tracker system. This computation can be expressed as:

$$C_{\text{expected}} = T_D^{-1} \cdot T_A \cdot C$$

Where  $C$  denotes the points in the calibration object frame, and  $C_{\text{expected}}$  yields the corresponding expected coordinates in the EM tracker frame.

To perform pivot calibration for the EM probe, we first define a local coordinate system using the first frame of pivot calibration data. This begins with computing the midpoint of the initial frame:

$$g_{\text{mid}} = \frac{1}{N} \sum_{i=1}^N g_i$$

Where  $g_i$  are the observed points. Next, we center relative to this midpoint:

$$g' = g - g_{\text{mid}}$$

This transformation effectively centers the probe measurements around the origin of the local coordinate system. For each frame of pivot data, we compute the transformation that maps the probe coordinates to the EM tracker coordinates. This is achieved by using the `point_set_registration` function, which calculates the optimal rotation matrix  $R_i$  and translation vector  $p_i$  that best aligns the current frame's data with the previously centered points. After gathering all rotation and translation results, the `pivot_calibration` function is called, which takes these transformations as input to compute the final tip position of the probe relative to the EM tracker.

To perform pivot calibration for the optical probe, we must first transform the positions of the optical tracker beacons into the EM tracker coordinates by computing  $F_D$  by the same method as for calculating  $C_{\text{expected}}$ , and subsequently perform a pivot calibration.

## Program Structure

The essential files in the program package include: readData.py, pointSetRegistration.py, pivotCalibration.py, computeExpectedValues.py, pivotEMCalibration.py, pivotOptCalibration.py and generateOutput.py, along with various testing files.

readData.py is responsible for loading and parsing input data from input files. It contains several functions, including read\_calbody, read\_calreadings, read\_emptivot, and read\_optpivot, that read formatted data and store it in a format suitable for further processing, facilitating smooth data handling throughout the program.

pointSetRegistration.py contains the method point\_set\_registration, designed to align two sets of 3D points by computing the optimal rotation and translation. This function takes two NumPy arrays as inputs, each representing the coordinates of the source and target points, respectively. The source and target arrays should be  $n \times 3$ , where  $n$  is the number of points in each set. The method implements Arun's method for point set registration and calculates root mean squared error between the transformed source points and the target points. The function outputs a tuple consisting of the optimal rotation matrix  $R$  and the translation vector  $p$ .

pivotCalibration.py includes the pivot\_calibration routine, which is designed to compute the location of the pointer tip within the local pointer frame based on calibration data represented by frames. This method takes two inputs: a list of rotation matrices and a list of translation vectors, which collectively represent the calibration data. The method performs pivot calibration as described above and outputs two components: the translation vector  $t_g$  and the position of the pointer tip  $p_{dimple}$ .

computeExpectedValues.py contains the computeExpectedValue method, which accepts a file prefix as an argument to compute expected calibration values from 3D point cloud data. The method first utilizes the read\_calbody and read\_calreadings functions from readData.py to parse calibration data from files associated with the provided prefix. The method begins by reading the calibration body data, which includes the number of data points  $N_D$ , angles  $N_A$ , and calibration points  $N_C$ , along with their respective coordinates. It then proceeds to read the calibration readings, which provides the number of frames  $N_{frames}$  and corresponding frame data. Using the point\_set\_registration function it computes the transformation matrices  $T_D$  and  $T_A$ , and computes the expected values for  $C$ , as previously described. The method returns the number of calibration points  $N_C$ , the number of frames  $N_{frames}$ , and the array of expected calibration points  $C_{arr}$ .

pivotEMCalibration.py contains the em\_pivot\_calibration method, which is designed to calibrate the EM tracker. This method accepts a file prefix as an argument and outputs the EM coordinate position of the EM probe tip. The process begins by parsing the input file using the read\_emptivot function, which retrieves frame data, the number of geometry points  $N_g$ , and the number of frames  $N_{frames}$ . EM Pivot Calibration is then performed as described above.

pivotOptCalibration.py also contains a single method, opt\_pivot\_calibarion, which is designed to calibrate the optical probe's position in an EM tracking system. This method accepts a file prefix as an argument and utilizes the read\_optpivot method to parse frame data, ultimately outputting the EM coordinates of the optical probe tip. The calibration process begins by reading

the input file, which yields frame data along with the number of geometry points  $N_H$ ,  $N_D$ , and  $N_{frames}$ . It also receives calibration data using the `read_calbody` function. Optical pivot calibration is then carried out as previously described. This method returns the  $3 \times 1$  EM coordinate positions  $t_g$  and the dimple position  $p_{dimple}$  in a  $6 \times 1$  array.

Finally, `generateOutput.py` is a script designed to facilitate the generation of output files containing calibration and expected position data for various probes based on provided file prefixes. This method uses `computeExpectedValue`, `em_pivot_calibration`, and `opt_pivot_calibration`, and creates a properly formatted output file, facilitating comparison to debugging data.

## Debugging Methods

Apart from using the debugging files to compare outputs, we primarily used our own test files and print statements inside functions to debug code. The print statements ensured that our `point_set_registration` function manipulated the arguments provided in the function correctly and returned outputs in the correct form  $R, p$ . Our test file `testPointSetRegistration.py` validated the method for simpler transformations including simple translation, rotation and mirror transformation. We tested data retrieval from files by printing out return values for functions in `readData.py`, and ensured consistency in the format of `frames_data` returned. To test `compute_expected_values`, we compared the output to the desired output in the debugging files. Testing pivot calibration was painful, because we tried numerous methods for solving the least squares methods, including various numpy and scikit.learn libraries, which were giving multiple outputs for  $x$  in our system  $Ax = b$ , for changing bounds. We resolved the problem by using the aforementioned pseudo-inverse method. Our test files `testPivotCalibration.py` and `testEMPivotCalibration.py` were invaluable in our quest to acquire the correct output. After `pivot_calibration` was thoroughly tested in this manner, implementing `opt_pivot_calibration` was straightforward as it involved only an extra rotation compared to `em_pivot_calibration`. We have removed our debugging print statements from our functions to improve code clarity and have added doc strings and sufficient comments to render the code readable.

## Program Validation - Debugging

To begin the process of debugging our methods, we began by checking the function of `point_set_registration` and `pivot_calibration` using simple generated data, such as a 45-degree angle or a 180 rotation. Once validated, we moved to the provided debugging data, which demonstrates sufficient testing of the `em_pivot_calibration`, `opt_pivot_calibration`, and `compute_expected_values` methods. Below, we have provided two tables describing our output from the debugging data set. We have not provided the expected points C, however they can be located in individual `output.txt` files in the OUTPUT folder of the zipped submission.

Data	EM Probe Position (mm)			Optical Probe Position (mm)		
	x	y	z	x	y	z
pa1-debug-a	190.6	207.4	209.2	400.6	402.1	203.5
pa1-debug-b	194.1	209.9	201.2	397.4	395.8	200.2
pa1-debug-c	195.6	200.0	205.2	392.6	409.4	202.1
pa1-debug-d	201.1	192.0	208.7	408.7	401.4	208.7
pa1-debug-e	200.6	202.5	195.5	392.8	392.0	209.8
pa1-debug-f	193.9	189.1	208.6	402.7	396.8	204.8
pa1-debug-g	201.0	196.6	205.5	405.6	395.0	207.6

*Table 1: The computed x, y, z coordinates of the EM and optical probe positions (mm) from each debugging file.*

Data	Mean Error in EM Pivot Probe Position (mm)	Mean Error in Optical Probe Position (mm)	Mean Error in $C_{expected}$ (mm)
pa1-debug-a	0.000	0.000	0.002
pa1-debug-b	0.000	0.010	0.458
pa1-debug-c	0.000	0.000	0.483
pa1-debug-d	0.000	0.000	0.010
pa1-debug-e	0.022	0.000	1.362
pa1-debug-f	0.010	0.000	1.923
pa1-debug-g	0.000	0.010	1.700

*Table 2: The computed mean error in the generated EM and optical probe positions, as well as the expected C coordinate (mm), for each debugging file.*

According to the mean error in the debugging data sets, the EM and optical pivot calibration functions are successful with minimal error. The maximum error in EM probe position occurs in pa1-debug-e and constitutes 0.022 mm. The maximum error in optical probe position is 0.010 mm, occurring in both pa1-debug-b and pa1-debug-g. Values less than 0.3 mm are within the range of random noise; thus, the computed values are sufficiently close to the expected probe positions. Therefore, we are confident in the accuracy of the EM and optical pivot calibration functions.

As for the values of  $C_{expected}$ , there is some error present in files pa1-debug-b, pa1-debug-c, pa1-debug-e, pa1-debug-f, and pa1-debug-g, constituting at greatest an error of 1.923 mm (pa1-debug-f). File pa1-debug-b, which presented an average error of 0.46 mm ( $>0.3$  mm), is due to the introduction of EM noise. File pa1-debug-c, which presented an average error of 0.48 mm, is due to the introduction of EM Distortion. Although these error values are relatively low, the error becomes significant when compounded with other forms of error. For files pa1-debug-e, pa1-debug-f, and pa1-debug-g, there is both EM distortion and OT jiggle present, with the addition of EM noise for f and g. With all errors compounding, the mean error reaches significant levels up to 1.923 mm. To correct, we will apply appropriate distortions in programming assignment 2.

## Results

Data	EM Probe Position (mm)			Optical Probe Position (mm)		
	x	y	z	x	y	z
pa1-unknown-h	190.46	199.45	186.46	395.82	402.08	192.61
pa1-unknown-i	207.57	191.94	197.84	399.87	408.28	194.15
pa1-unknown-j	197.92	205.56	188.93	408.5	408.47	203.17
pa1-unknown-k	187.13	199.33	206.38	396.67	393.62	191.14

Table 3: Generated  $x$ ,  $y$ ,  $z$  coordinates of the EM and optical probe positions (mm) from the unknown data sets.

## Discussion

Although the expected values for the EM pivot, optical pivot, and expected C coordinates were not provided for the unknown data set, we are confident in our results based on minimal error in the debugging data sets. It is likely, based on the previously noted error in the expected C coordinates, that there is variance of up to 2 mm in pa1-unknown-h and pa1-unknown-i, since both have EM noise, distortion and OT jiggle. In programming assignment 2, we will correct for this error propagation by sufficiently accounting for EM noise and distortion or by applying corrective distortions.

## Who Did What

For the program, Aryavrat completed readData.py, computeExpectedValues.py, pivotOptCalibration.py, and generateOutput.py. Ben completed pointSetRegistration.py and pivotEMCalibration.py. Together, we solved pivotCalibration.py.

For the report, Aryavrat completed Program Validation - Debugging and Results. Ben completed Mathematical Approach, Algorithm Overview, Program Structure, and Discussion.

## References

1. Harris, C.R., Millman, K.J., van der Walt, S.J. et al. Array programming with NumPy. Nature 585, 357–362 (2020). DOI: 10.1038/s41586-020-2649-2. ([Publisher link](#)).
2. K. S. Arun, T. S. Huang and S. D. Blostein, "Least-Squares Fitting of Two 3-D Point Sets," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-9, no. 5, pp. 698-700, Sept. 1987, doi: 10.1109/TPAMI.1987.4767965.