
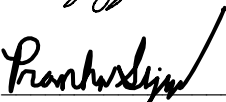


Programming Assignments 1 & 2 601.455 and 601/655 Fall 2025
 Please also indicate which section(s) you are in (one of each is OK)

Score Sheet

Name 1	Luiza Brunelli Buhner
Email	LBRUNEL1@JHU.EDU
Other contact information (optional)	
Name 2	Pranhav Sundararajan
Email	psundar2@jhu.edu
Other contact information (optional)	
Signature (required)	I (we) have followed the rules in completing this assignment <div>   </div>

Grade Factor		
Program (40)		
Design and overall program structure	20	
Reusability and modularity	10	
Clarity of documentation and programming	10	
Results (15)		
Correctness and completeness	15	
Report (45)		
Description of formulation and algorithmic approach	15	
Overview of program	10	
Discussion of validation approach	10	
Discussion of results	10	
TOTAL	100	

Programming Assignment #1

Luiza Brunelli
Pranhav Sundararajan

Goal

1. Develop (or develop proficiency with) a Cartesian math package for 3D points, rotations, and frame transformations.
2. Develop a 3D point set to 3D point set registration algorithm
3. Develop a “pivot” calibration method.

Problem 4

To compute the transformations F_d for problem 4a, we first developed a function called `find_transformation.m`. This function has two inputs, P and Q , and has two outputs, R and t . P is the matrix of vectors before they are transformed and Q is the matrix of vectors after they have been transformed. The outputs R and t form the best-fitting transformation matrix F_d , where $F_d = [R|t]$. To find the best fitting transformation that maps P onto Q , singular value decomposition is used by implementing the orthogonal Procrustes method. First, we will define the size of the inputs and outputs. P and Q both must be $N \times 3$ matrices of 3D points (N 3D points). R will be a 3×3 matrix and t will be a 3×1 vector. The equation that this algorithm will try to find a best-fit transformation for is $Q_i = RP_i + t$, where i spans across all points 1 to N .

The methodology of the implemented algorithm is as follows:

1. The means or average vectors of both P and Q are computed and subtracted from P and Q , respectively, such that both sets have a mean of zero, forming new matrices P_m and Q_m .
2. The cross covariance matrix is then calculated from the two matrices P_m and Q_m using the formula $H = \text{transpose}(P_m) * Q_m$.
3. The cross covariance matrix H , can be decomposed using singular value decomposition into U and V , where $H = U * E * \text{transpose}(V)$. Using the `svd` function in matlab, we can find U and V .
4. We then calculate the rotation matrix, R , by first calculating another matrix S , where $V * S * \text{transpose}(U) = R$ and S equals the identity matrix where the last term in the matrix equals the determinant of $V * \text{transpose}(U)$.
5. Now that we have U , V , and S , we can compute R using the formula described above: $R = V * S * \text{transpose}(U)$.

6. Once we have R , we can calculate the best fitting t or translation vector using the equation $t = m_q - R * m_p$, where m_q is the average vector of Q and m_p is the average vector of P that we computed in step 1 to normalize P and Q .
7. Therefore, the outputs are R as a 3×3 matrix and t as a 3×1 vector, which when concatenated form the corresponding transformation matrix F .

This is the main function used to compute each of Problem 4a, 4b, and 4d. To extract the original vectors and the tracker coordinates from the input files, we created three functions to parse through the relevant text files and store the necessary information. The first function is called `read_block.m`. This simply takes in a text file's ID and the number of columns to read from the file and places them in an output matrix. This matrix is an $N \times 3$ file where the columns represent the x , y , and z coordinates as listed in the input text files. Since the fixed landmarks were presented in the `CALBODY.txt` file, we created another function called `read_calbody.m` which is used to read each the appropriate number of records for the d , a , and c fixed vectors, using `read_block.m`, and store them in separate $N_i \times 3$ matrices, where N_i represents the number of records for each d , a , and c . The third helper file was `read_calreadings.m`, which was functionally similar to `read_calbody.m` but was altered such that the outputs contained cells of the measured positions D , A , and C in similar matrices, where the number of cells equals the number of data frames according to the file.

These functions are all used together in the functions `problem4a.m`, `problem4b.m`, and `problem4d.m`. These functions, when called in the master script `pal.m`, solve the question corresponding to the function name. Function `problem_4a.m` has two inputs: the location of the `calbody.txt` file and the `calreadings.txt` file. The `read_calbody.m` function then outputs the known locations of the optical markers placed at the base unit of the electromagnetic tracking system, d . The `read_calreadings.m` function outputs the measured locations relative to the optical tracker of the electromagnetic base markers, D . Since the measured locations are stored in a collection of cells called D_{cells} , where the number of cells is equal to the number of data frames and each cell contains a specified number of measurements from the input text file. If we call each matrix of measured vectors inside each cell D_k , then d and D_k are passed as inputs into the `find_transformation.m` function, which as previously described, outputs the best-fit rotation matrix and transformation vector. The rotation matrix and transformation vector are concatenated and the resulting transformation matrix F is standardized, with a fourth row consisting of all zeroes except for the last term which is a 1. The final 4×4 matrix F is placed in the k th cell, if the k th data frame was used as the input. This is done for every one of the data frames, creating an output with N cells where N is the number of data frames and each cell contains the full 4×4 transformation matrix F_d for that particular data frame.

Problem 4b was solved in a very similar manner to Problem 4a. File `pal.m` calls `problem4b.m`, which is functionally the exact same as `problem4a.m`. It requires the same two inputs and outputs a cell of N elements, where each cell contains the 4×4 transformation matrix F_a corresponding to that data frame and N is equal to the number of data frames in the input file. Inside the function, the `find_transformation.m` method is called again but this time the inputs to that are the known locations of markers on the calibration object (a_j) and the relative measured positions (A_j).

The file `problem4d.m` is called by the `pa1.m` file to solve problem 4d. This function builds off of `problem4b.m`, where it calculates the 4x4 transformation matrices F_d and F_a for each data frame and stores them in a separate cell for d and a . To calculate the C_i expected for each c_i in one data frame, the transformation matrices F_d and F_a for that data frame were extracted. Then, the inverse of F_d was calculated and the dot product of that and F_a was also calculated to produce a transformation matrix T . This matrix T was used to transform each vector c_i inside a particular data frame, and the resulting transformed vectors C_i expected were stored in a cell for that data frame. This process was carried out for all data frames, resulting in a cell with N elements with each element containing C_1 expected to C_N expected. Therefore, `pa1.m` calls `problem4a.m`, `problem4b.m`, and `problem4d.m` and stores their outputs, which are later written to an output text file after Problem 5 and Problem 6 are solved.

Problem 5

For problem 5, the goal was to apply the EM tracking data to perform a pivot calibration for the EM probe and determine the position relative to the EM tracker base coordinate system of the dimple in the calibration post. To achieve this, we first developed a function called `pivot_calibration.m` and a helper function `read_pivot_data.m`.

The `read_pivot_data.m` function is responsible for opening and reading the `EMPIVOT.txt` file. It will first read the header line to determine the number of EM markers on the probe (NG) and the number of data frames ($Nframes$). It then iterates through each frame, reading the NG sets of 3D coordinates and storing them in an $NG \times 3$ matrix. Each of those is then stored in an array (G_frames), which will be the output of the function.

The core part of the solution is in the `pivot_calibration.m` function, which takes the pivot data and calculates the pivot point by solving a system of linear equations.

The methodology of the implemented algorithm is as follows:

1. The function takes the G_frames array as input. The first frame is used to define a local probe coordinate system. G_0 is calculated as the mean of the marker positions in this first frame. The local coordinates of the markers are defined relative to G_0 : $g = G_first_frame - G_0$; these vectors represent the fixed locations of the markers in the probe's coordinate system.
2. For each frame k in 1 to $Nframes$, we use `find_transformation` to compute the rotation R and translation p that maps the local probe marker g to the observed marker coordinates in that frame G_k , which gives us transformation for each frame.
3. We then use the equation $P_dimple = R_k * t_G + p_k$ (given the location of P_dimple remains fixed in the tracker's coordinate system and t_G is fixed relative to the probe's local coordinate system).
4. We construct a system of equations in the form $Ax=b$ by stacking equations from all $Nframes$. A is a matrix with all R_k matrices and $-I$ (3×3) and b is a vector formed by negating the translation vectors $-p_k$. Finally, x is a 6×1 vector containing the coordinates of t_G and P_dimple .

5. This system is solved for x using MATLAB's \backslash operator, which provides a solution minimizing the error across all frames.
6. The output and final result is P_dimple , which is extracted from the last 3 elements of the solution vector x .

The file `problem5.m` takes the `empivot_file` path as input, calls `read_pivot_data.m` to get the necessary data, uses that for `pivot_calibration.m`, and returns the final calculated coordinates of the EM pivot post, P_dimple_em .

Problem 6

For problem 6, the goal was to apply the optical tracking data to perform a pivot calibration of the optical tracking probe. It involves performing a similar pivot calibration as problem 5 for the optical tracking probe to find the position of the same dimple post, but now relative to the optical tracker's coordinate system and transformed into the EM tracker's coordinate system. This involves an additional transformation since the optical tracker itself may not be stationary relative to the EM tracker base between frames. We implemented this solution in `problem6.m` with a helper function `read_optpivot_data.m`.

The `read_optpivot_data.m` function parses the `OPTPIVOT.txt` file and, for each of the N frames, it reads the ND coordinates of the EM base markers as seen by the optical tracker D_k and the NH coordinates of the optical probe markers H_k . It outputs 2 arrays D_frames and H_frames , containing this data for each frame.

The methodology of the implemented algorithm in `problem6.m` is as follows:

1. The function takes the `calbody_file` and the `optpivot_file` as inputs. It reads the known locations of the markers on the EM base d from the `calbody` file using the `read_calbody.m` helper function. It also reads the frame by frame measurements of the EM base markers D_k and the optical probe markers H_k using `read_optpivot_data.m`.
2. We then choose the EM tracker base's coordinate system as our reference and, for each frame k , the transformation $F_D[k]$ from the EM base coordinate system to the optical tracker's coordinate system is calculated by calling `find_transformation(d, D_k)`, which finds the registration between the known EM base marker locations d and their observed positions D_k in that frame.
3. The inverse transformation is then calculated for each frame using the `inver_transform.m` helper function, which maps points from the optical tracker's coordinate system to the EM tracker's coordinate system. The inverse transformation is defined by $R_inv = R'$ and $t_inv = -R'*t$ and is applied to the optical probe marker readings for each frame, H_k , to get a new set of points H_prime_k (these represent the locations of the optical probe markers in the EM tracker base's coordinate system).
4. The array containing the H_prime_k frames is then passed to the `pivot_calibration.m` function from problem 5 and returns P_dimple_opt (position of the dimple post with respect to the EM tracker base, determined by optical pivot data).

Testing

To verify if the program is working correctly, we implemented a simple method to calculate the average percent relative error of the vectors present in the output file we produced and the debug output files that were given to us. The vectors from our output file was converted into a matrix, 3 columns wide, and the vectors from the expected output given to us was converted into another matrix 3 columns wide. Then the difference matrix was calculated by subtracting the calculated output matrix from the expected output matrix. Then, the Euclidean norm of each difference vector was calculated. Similarly, the Euclidean norm of each vector from the expected output was also calculated. The Euclidean norm of the difference vector was divided by the Euclidean norm of the expected vector, giving us the relative error for each vector in the dataset. The average relative error was then calculated from all of the vectors. This value was then multiplied by 100 to finally give us the percent relative error for the entire dataset. This was done for all debugging datasets given. The average percent relative errors are in the table below for each file.

Comparison Files	Average Percent Relative Error between measured and expected vectors
Debug-A	0.0002%
Debug-B	0.0573%
Debug-C	0.0574%
Debug-D	0.0012%
Debug-E	0.1704%
Debug-F	0.2307%
Debug-G	0.2053%

We can see that the error for Debug-A is 0.0002% and that for Debug-D it's only slightly higher. This can be attributed to there being no EM distortion or EM noise present in either file as these were the only two files satisfying that criteria. Debug-D does have OT-jiggle, which could be why there is minimal error. Debug-B and Debug-C have some error, but less than Debug E-G, and this may be because B and C had only EM noise and EM distortion, respectively. It is also interesting that Debug_E had less error than F or G because E contains EM distortion and OT jiggle but F and G contain all three types of noise. Therefore, the average percent relative error results seen here are understandable and seem to be correlated with the amount of noise/distortion that has been applied on the individual datasets, showing the validity of our code.

Table of Results for Unknown Data

Input Test Files	Est. pos. With EM probe	Est. pos. With optical probe
Unknown-H	190.46, 199.45, 186.46	395.82, 402.08, 192.61
Unknown-I	207.57, 191.94, 197.84	399.87, 408.28, 194.15
Unknown-J	197.92, 205.56, 188.93	408.50, 408.47, 203.17
Unknown-K	187.13, 199.33, 206.38	396.67, 393.62, 191.14

Discussion of Results

The table above describes the estimated positions with the EM and optical probes that we calculated from the four input test files that were not used for debugging our code. The expected C vectors are placed in the output text file in the attached zip file. From the error analysis conducted on the debugging test files, we can infer that the positions in the table for file H and I would have the most error when compared to the rest of the debugging input files or similar error to debugging files F and G. This is because these 4 input datasets were the only ones that contained EM distortion, EM noise, and OT jiggle. From the table showing percent relative error between the measured and expected vectors from the debugging files, the datasets which had all three sources of noise were the ones with the highest average percent relative error. There was no summary as to how datasets J and K were created, so we cannot be certain what the relative error would be for these datasets. However, the values estimated for positions with the EM and optical probe for files J and K are similar to the output positions we calculated from files B-D, so it is possible that datasets J and K had one or two of the three sources of noise applied to them.

Project Contributions

Both partners met to discuss all problems and how to solve them, Pranhav then specifically focused on problem 4 and evaluation of our results while Luiza focused on 5 and 6, helping each other throughout the process.