THA Programming Assignment #3

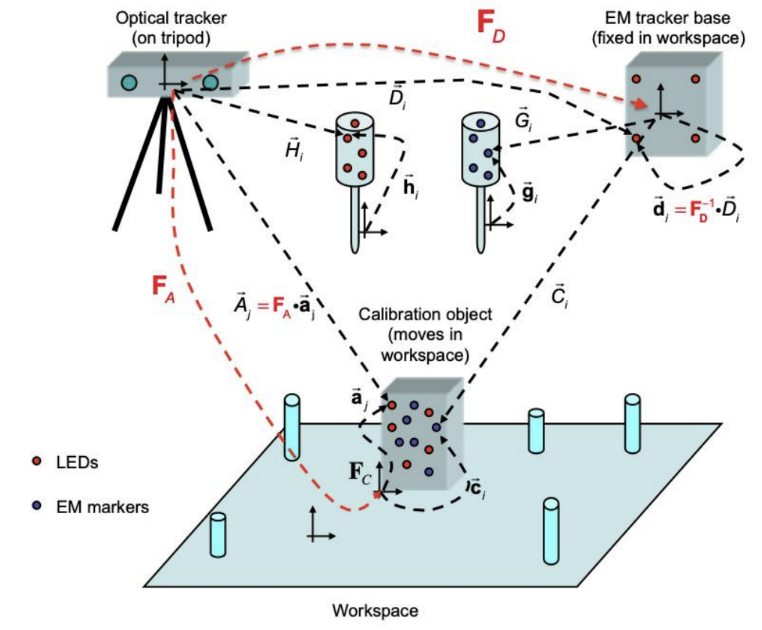
ME 397 ASBR, Sp 23, Dr. Farshid Alambeigi

Jared Rosenbaum and Steven Swanbeck

**PA**

*General Description:*

In this assignment, elements related to calibration, registration, and tracking are investigated for the stereotactic navigation system shown in Figure 1. Additionally, the Eye in Hand calibration problem is solved for a set of robot and sensor configurations, with and without noise.



*Figure 1: Stereotactic Navigation System used in PA1*

**PA1-1) 3D Point Set Registration Method Synthesis**

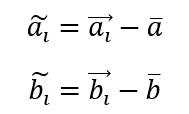
*Problem Description:*

This goal requested the creation of a 3D point set to 3D point set registration algorithm.

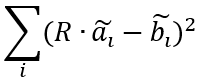
*Method for Solution:*

Given two point clouds, each taken from the same measurement source and looking at a single rigid object at two different configurations (a & b), the correspondence-based registration method is able to find the transformation between these configurations. It does this by first calculating the centroids for each configuration (ā & b̅)

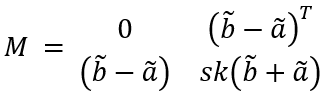
The next step is to calculate the deviation of each point in the cloud from the centroid (ã & ᵬ)



Next, we use the quaternion technique to find the rotation *R* that minimizes



To do this, we first set up a matrix *M* of the following form



Performing singular value decomposition on *M*, the last column of the resultant *V* matrix is the smallest singular value and can be viewed as a quaternion. By transforming it to a matrix we have our rotation *R*. Finally, we find the translation using



*Explanation of Program:*

In order to streamline the point set registration process, we created a generalized program that we could call any time we needed to relate one set to another. That function, *correspondence\_registration( )* is shown below.

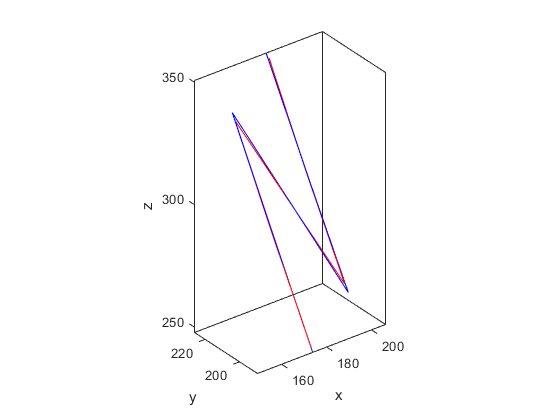
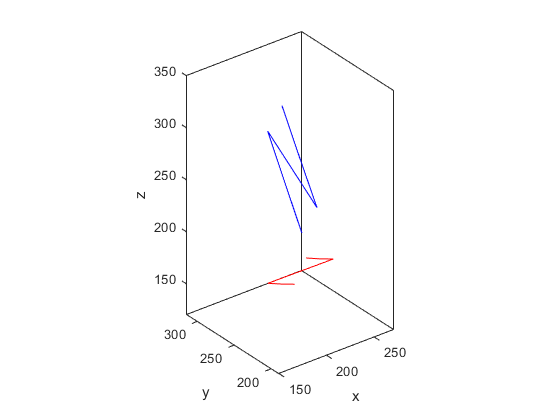
function [T, a\_bar, b\_bar] = correspondence\_registration(a\_set, b\_set)  
 %Takes two point clouds representing a single object in two frames, and relates them.  
 % step 1: calculating centroids  
 a\_sum = [0, 0, 0];  
 for i = 1:size(a\_set, 1)  
 a\_sum = a\_sum + a\_set(i, :);  
 end  
 a\_bar = a\_sum ./ size(a\_set, 1);  
  
 b\_sum = [0, 0, 0];  
 for i = 1:size(b\_set, 1)  
 b\_sum = b\_sum + b\_set(i, :);  
 end  
 b\_bar = b\_sum ./ size(b\_set, 1);  
 % step 2: calculating deviations from centroids  
 a\_tilda = zeros(size(a\_set));  
 for i = 1:size(a\_set, 1)  
 a\_tilda(i, :) = a\_set(i, :) - a\_bar;  
 end  
  
 b\_tilda = zeros(size(b\_set));  
 for i = 1:size(b\_set, 1)  
 b\_tilda(i, :) = b\_set(i, :) - b\_bar;  
 end  
 % step 3: find R that minimizes (using quaternion approach here)  
 M = [];  
 for i = 1:length(a\_tilda)  
 M = [M;  
 0 b\_tilda(i, :) - a\_tilda(i, :);  
 (b\_tilda(i, :) - a\_tilda(i, :))' Axis2SkewSymmetricMatrix(b\_tilda(i, :) + a\_tilda(i, :))];  
 end  
  
 [U, S, V] = svd(M);  
  
 q = V(:, 4);  
 R = Quaternion2RotationMatrix(q);  
  
 % step 4: find p  
 p = b\_bar' - R \* a\_bar';  
  
 % step 5: putting it together  
 T = [R, p; zeros(1,3) 1];  
 end

The above function inputs two sets, the respective a and b, and performs the transformations described in the Method for Solution section to find the rigid transformation Tbb’.

*Answer / Test Cases:*

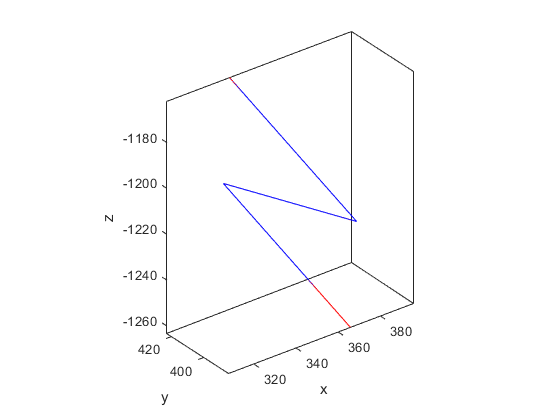
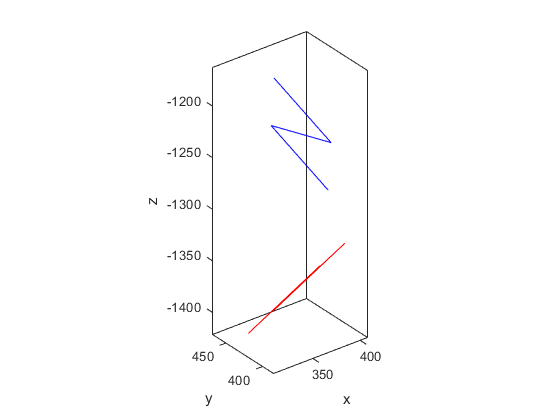
The registration function is used repeatedly throughout this THA; demonstration of its usage can be found in sections PA1-3, PA1-4, and PA1-5.

Some simple tests were done to confirm the functionality of the algorithm using some of the data taken from the provided files. An example of registration between two frames of the EM tracker data used for pivot calibration is shown below:



*Figure 2: Example of correspondence-based registration between two consecutive pointsets of EM tracker data captured during pivot calibration using the previously shown function.*

Note that because there is noise in the EM-tracker data, the individual points of the markers on each body do not perfectly correspond, though the calculated transform does produce the optimal registration possible given this noise. Shown next is an example with optical tracker data taken in consecutive frames during pivot calibration.



*Figure 3: Example of correspondence-based registration between two consecutive pointsets of optical tracker data captured during pivot calibration using the previously shown function.*

The assumption that the optical tracker does not have noise is proven in this test, as the pointsets perfectly overlap following registration.

**PA1-2) Pivot Calibration Method Synthesis**

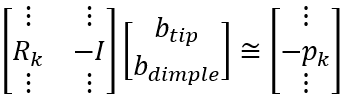
*Problem Description:*

This goal requested the creation of a pivot calibration algorithm.

*Method for Solution:*

Given a set of known rigid transformations obtained by pivoting a pointer tool around a single point (referred to heretofore as a dimple), pivot calibration is the method of determining the translation btip, from the reference frame (centroid of the pointer tool) to the dimple, and the translation bdimple from the tracking base to the dimple. For each time step/measurement *k*, we know the transformation from the tracking base to the dynamic reference frame as shown in PA1-1, and can decompose this transformation into Fk and pk, used as follows

This can be set up as a least squares problem in the form of Ax = b, and solved as shown in the Explanation of Program section.



*Explanation of Program:*

In order to streamline the pivot calibration process, we created a generalized program that we could call any time we needed to perform pivot calibration. That function, *pivot\_calibration( )* is shown below.

function [b\_tip, b\_post, T\_bank] = pivot\_calibration(raw)  
 %This function performs pivot calibration on a set of frames viewing a pointer pivoting in a dimple  
  
 T\_bank = cell(1, size(raw, 2));  
 A = zeros(3 \* size(raw, 2), 6);  
 b = zeros(3 \* size(raw, 2), 1);  
  
 % our defined transformation between first frame and sensor w/  
 % arbitrary orientation  
 R0 = eye(3);  
 a\_sum = [0, 0, 0];  
 for i = 1:size(raw{1}, 1)  
 a\_sum = a\_sum + raw{1}(i, :);  
 end  
 a\_bar = a\_sum ./ size(raw{1}, 1);  
 P0 = a\_bar;  
 T0 = [R0 P0';  
 zeros(1,3), 1];  
 T\_bank{1} = T0;  
 A(1:3, :) = [T\_bank{1}(1:3, 1:3) -eye(3)];  
 b(1:3) = -T\_bank{1}(1:3, 4);  
  
 % calculating Ts between first frame and all following frames using  
 % registration, then transforming into sensor frame and accumulating A  
 % and b matrices  
 for i = 1:size(raw, 2) - 1  
 T\_bank{i + 1} = T\_bank{1} \* correspondence\_registration(raw{i}, raw{i + 1});  
 A(3 \* i + 1:3 \* i + 3, :) = [T\_bank{i + 1}(1:3, 1:3) -eye(3)];  
 b(3 \* i + 1:3 \* i + 3) = -T\_bank{i + 1}(1:3, 4);  
 end  
  
 % solving b\_tip and b\_post  
 x = (A' \* A) \ (A' \* b); % Least squares solved using transpose of A  
 b\_tip = x(1:3);  
 b\_post = x(4:6);  
 end

This function first takes the first frame of pivot data and stores the transformation from the EM tracker base to the pointer. Then, for each subsequent frame, the function *correspondance\_registration( )* as described in PA1-1 is used to calculate the subsequent transformations. All of these are then populated into the *A* and *b* matrices, and the least squares problem is solved using the equation *ATAx=ATb*.

*Answer / Test Cases:*

The pivot calibration function is used in parts 4 and 5 in this THA; demonstration of its usage can be found in sections PA1-4, and PA1-5.

**PA1-3) Calibration Object to EM Tracker Base**

*Problem Description:*

This problem requires computation of the “expected” values *Ci(expected)* that describe the position of the EM markers on the calibration object expressed in the EM tracker base coordinate system as determined using measurements taken of the EM base and calibration object using the optical tracker and the known locations of the EM markers on the calibration object. Since we assumed the optical tracker measurements have no error, these calculations, which are dependent only on optical tracker data and known distances on the rigid calibration object, represent the true positions of the EM markers within the frame of the EM base. This is useful to determine in this manner because the EM measurements will have noise, preventing as accurate of positions to be determined directly.

*Method for Solution:*

The first step to solving this problem is to calculate the transform between the optical tracker and EM base coordinate frames. This is done using measurements of the optical tracker markers on the EM base along with their known locations on the EM base itself. We use the previously covered registration algorithm (PA1-1) to compute the transform between the EM base (using the known positions of the optical markers on the EM base) and the optical tracker (using measurements of the EM base optical markers captured by the optical tracker) for each frame of data.

Next, the transform between the optical tracker and the calibration object must be determined for each frame of data. This is done in a similar method as above, using the registration algorithm to register the known positions of optical markers on the calibration object with respect to the measurements taken by the optical tracker for each frame of data.

Finally, the expected measurements *Ci* of the EM markers on the calibration object can be calculated using the pair of transformations associated with each of the pairwise registration steps for each frame of data and the known positions of the EM markers on the calibration object as

This is computed for each EM marker on the calibration object for each frame of data.

*Explanation of Program:*

The snippet of our main file (shown in the *Other Relevant Code* section) relevant for this part of the problem is shown below:

disp('------------3)---------------')  
 % Repackaging the original object for ease of use  
 em\_package = cell(1, size(package2, 2));  
 for i = 1:size(package2, 2)  
 em\_package{i} = package2{i}{1};  
 end  
  
 calobject\_package = cell(1, size(package2, 2));  
 for i = 1:size(package2, 2)  
 calobject\_package{i} = package2{i}{2};  
 end  
  
 calobject\_package2 = cell(1, size(package2, 2));  
 for i = 1:size(package2, 2)  
 calobject\_package2{i} = package2{i}{3};  
 end  
  
 % Calibration using the optical markers  
 [T\_bank\_em] = opt\_calibration(package1{1}, em\_package);  
 FD\_bank = T\_bank\_em;  
  
 % % Verification  
 % a\_set = package1{1};  
 % b\_set = em\_package{8};  
 % plotBody(a\_set, b\_set)  
 % a\_set\_p = [];  
 % for i = 1:size(a\_set, 1)  
 % temp = T\_bank\_em{8} \* [a\_set(i, :) 1]';  
 % a\_set\_p = [a\_set\_p; temp(1:3)'];  
 % end  
 % plotBody(a\_set\_p, b\_set)  
  
 [T\_bank\_cal] = opt\_calibration(package1{2}, calobject\_package);  
 FA\_bank = T\_bank\_cal;  
  
 % % Verification  
 % a\_set = package1{2};  
 % b\_set = calobject\_package{8};  
 % plotBody(a\_set, b\_set)  
 % a\_set\_p = [];  
 % for i = 1:size(a\_set, 1)  
 % temp = T\_bank\_cal{8} \* [a\_set(i, :) 1]';  
 % a\_set\_p = [a\_set\_p; temp(1:3)'];  
 % end  
 % plotBody(a\_set\_p, b\_set)  
  
 % Computing c\_expected  
 c\_expected = cell(1, size(package2, 2));  
 for i = 1:size(package2, 2)  
 c\_sub = zeros(size(package2{1}{3}));  
 for j = 1:size(package2{1}{3}, 1)  
 inter = FD\_bank{i} \ FA\_bank{i} \* [package1{3}(j, :) 1]';  
 c\_sub(j, :) = inter(1:3)';  
 end  
 c\_expected{i} = c\_sub;  
 end  
 c\_expected{1} - calobject\_package2{1};  
.

.

.  
 % Comparison  
 fprintf('\n\n\n')  
 disp('------------Comparison of Results---------------')  
 for i = 1:size(repackage, 2)  
 fprintf('-----%i-----\n', i)  
 diff = round(repackage{i}(:, :) - package5{i}(:, :), 2);  
 end  
 diff;  
 max(max(abs(diff)))  
 mean(mean(diff.^2))

This code starts by repackaging the nested cell arrays we originally constructed to read in the data from the provided .txt files to make it more easily used in this problem. Following this, we call a new function *opt\_calibration( )*, shown in the *Other Relevant Code* section, which calculates the transform between the local object frame (either the calibration object or EM base featuring known positions of optical markers) and the optical tracker frame using the measurements taken at each timestep. These transforms *FD* and *FA* are stored for the next step, which loops through the ci positions of the EM trackers on the calibration object and computes the transform between those positions and the EM base frame using the previously shown equation.

*Answer / Test Cases:*

For this THA, we were provided with 7 debug cases. For PA1-3, the output data is extremely lengthy and difficult to parse through by eye. In order to alleviate this problem, we compare our output data with the corresponding debug output file by subtracting the provided outputs from our outputs, creating a large set of numbers showing how similar our outputs are to the demo outputs. To quantify this comparison, we present the maximum difference calculated from a piecewise comparison of the absolute difference between the calculated and provided expected outputs and the mean-squared error (MSE) of all these differences across each point *Ci* for each frame of data.

*Table 1: Differences between outputs for PA1-3*

| Debug Case | Maximum Difference (*mm*) | Mean-Squared Error (*mm2*) |
| --- | --- | --- |
| A | 0.01 | 7.41e-06 |
| B | 0.49 | 0.08 |
| C | 0.85 | 0.10 |
| D | 0.01 | 3.33e-06 |
| E | 4.16 | 1.45 |
| F | 3.82 | 1.75 |
| G | 4.55 | 2.03 |

As shown in these results, these errors are quite small relative to the *Ci*, which have components ranging between 200 and 700 mm. These differences between the calculated and expected outputs can perhaps be explained by floating point errors between our algorithms and MATLAB instances and those used to produce these values from the output files. However, they are generally close to the provided outputs and the discrepancies of this magnitude would be unlikely to cause significant issues in all but the most delicate operations.

The full outputs to this question can be found within the “unknown-output1” txt files for each case that are submitted with this report.

**PA1-4) Pivot Calibration in EM Tracker Base Frame**

*Problem Description:*

This problem requested the performance of pivot calibration on the EM probe to determine the position relative to the EM tracker base coordinates.

*Method for Solution:*

Because this problem is solely a pivot calibration question, all that has to be done is to input the data into the *pivot\_calibration( )* function!

*Answer / Test Cases:*

For this THA, we were provided with 7 debug cases. The output for PA1-4 for each test case is shown below, next to the corresponding debug output.

*Table 2: Outputs for PA1-4*

| Debug Case | Our Output | Debug Output |
| --- | --- | --- |
| A | 200.68, 198.28, 195.84 | 200.68, 198.28, 195.84 |
| B | 191.35, 205.54, 208.61 | 191.35, 205.54, 208.61 |
| C | 198.17, 196.64, 192.06 | 198.05, 196.74, 192.02 |
| D | 190.17, 206.33, 198.34 | 190.17, 206.33, 198.34 |
| E | 188.20, 209.19, 205.18 | 188.03, 208.86, 205.17 |
| F | 200.37, 192.18, 202.05 | 201.14, 192.51, 201.74 |
| G | 207.27, 206.40, 200.21 | 207.05, 206.05, 200.49 |

In the table above, it is clear that our outputs are nearly identical to the debug outputs, with the error never being more than 0.77mm.

These outputs can be found within the “unknown-output1” txt files for each case that are submitted with this report.

**PA1-5) Pivot Calibration in Optical Tracker Frame**

*Problem Description:*

This goal requested the performance of pivot calibration for the optical tracking probe, also in the EM tracking coordinates.

*Method for Solution:*

First, we must transform the optical probe measurements into the frame of the EM tracker base. We do this by first performing correspondence registration between the calibration body and the calibration reading. Using this, we are able to calculate FD, the transformation between the optical tracker and the EM tracker, and perform pivot calibration in the frame of the EM tracker using the optical tracker’s readings.

*Explanation of Program:*

The snippet of our main file (shown in the *Other Relevant Code* section) relevant for this part of the problem is shown below:

% 5)  
 disp('------------5)---------------')  
 % first transform optical probe measurements into em frame  
 em\_package2 = cell(1, size(package4, 2));  
 for i = 1:size(package4, 2)  
 em\_package2{i} = package4{i}{1};  
 end  
 [T\_bank\_em2] = opt\_calibration(package1{1}, em\_package2);  
  
 h\_em = cell(1, size(package4, 2));  
 for i = 1:size(package4, 2)  
 h\_id = zeros(size(package4{i}{2}));  
 for j = 1:size(package4{i}{2}, 1)  
 p\_i = T\_bank\_em2{i} \ [package4{i}{2}(j, :) 1]';  
 % p\_i = [package4{i}{2}(j, :) 1]'  
 h\_id(j, :) = p\_i(1:3)';  
 end  
 h\_em{i} = h\_id;  
 end  
  
 % now perform pivot calibration in same way as before  
 [b\_tip\_opt, b\_post\_opt, ~] = pivot\_calibration(h\_em)

The code used to perform the optical pivot calibration is shown above.

*Answer / Test Cases:*

For this THA, we were provided with 7 debug cases. The output for PA1-5 for each test case is shown below, next to the corresponding debug output.

*Table 3: Outputs for PA1-5*

| Debug Case | Our Output | Debug Output |
| --- | --- | --- |
| A | 409.22, 409.47, 200.73 | 409.22, 409.47, 200.72 |
| B | 390.29, 395.56, 207.94 | 390.29, 395.56, 207.94 |
| C | 407.55, 403.72, 197.93 | 407.55, 403.72, 197.93 |
| D | 399.40, 408.69, 192.02 | 399.40, 408.69, 192.02 |
| E | 397.44, 397.24, 209.16 | 397.44, 397.24, 209.16 |
| F | 404.50, 391.03, 201.44 | 404.50, 391.03, 201.44 |
| G | 409.60, 405.41, 195.10 | 409.60, 405.41, 195.09 |

In the outputs shown above, it is clear that our outputs match the debug outputs exactly, except for a few rare rounding errors.

These results are combined with those of the previous sections and outputted in the “unknown-output1” txt files corresponding to each of the unknown cases. These are included in the zip file for this submission.

**PA2-1) Eye in Hand, Noiseless**

*Problem Description:*

This goal requested the solution to the unknown transform *X* from the camera frame to the robot end-effector frame using the quaternion analytical approach.

*Method for Solution:*

Knowing the robot end effector configurations Ei and the sensor configurations Si, the Eye-in-Hand problem becomes an AX=XB problem, of the form

To solve this, we split it into two steps: solving for the rotation and then the translation using least squares.

First, we solve for the rotation in quaternion form. Decomposing RA and RB into their scalar and vector parts, we construct a matrix A and set up a least squares problem as



We then construct a matrix M by stacking A from every timestep



Taking the singular value decomposition of M, the fourth column of V is the unit quaternion representation of RX. Next, we solve for displacement using least squares:

*Explanation of Program:*

For this goal, several programs were created. The first of these functions is *relative\_motion\_matrix( )*, which creates the A and B quaternion sets for each time frame.

function [A\_quats, B\_quats, A\_cell, B\_cell] = relative\_motion\_matrix(q\_Robot\_config, q\_camera\_config, t\_Robot\_config, t\_camera\_config)  
 %This function creates the A & B matrices as described on W12-L1 slide 16  
 A\_cell = cell(1, size(q\_Robot\_config, 1)-1);  
 B\_cell = cell(1, size(q\_camera\_config, 1)-1);  
  
 A\_quats = zeros(size(q\_Robot\_config, 1)-1, 4);  
 B\_quats = zeros(size(q\_camera\_config, 1)-1, 4);  
  
 E\_all = cell(1, size(q\_Robot\_config, 1));  
 S\_all = cell(1, size(q\_camera\_config, 1));  
  
 for i = 1:size(E\_all, 2)  
 Ra = Quaternion2RotationMatrix(q\_Robot\_config(i, :)/norm(q\_Robot\_config(i,:))); %Given quaternions are not perfectly normalized - must normalize to get real valued solution quaternions  
 Pa = t\_Robot\_config(i, :);  
 E\_all{i} = [Ra Pa'; zeros(1,3) 1];  
 end  
 for i = 1:size(S\_all, 2)  
 Rb = Quaternion2RotationMatrix(q\_camera\_config(i, :)/norm(q\_camera\_config(i,:))); %Given quaternions are not perfectly normalized - must normalize to get real valued solution quaternions  
 Pb = t\_camera\_config(i, :);  
 S\_all{i} = [Rb Pb'; zeros(1,3) 1];  
 end  
 for i = 1:size(q\_Robot\_config, 1)-1  
 A\_cell{i} = E\_all{i} \ E\_all{i+1};  
 B\_cell{i} = S\_all{i} / S\_all{i+1};  
 end  
  
  
 for i = 1:size(A\_cell, 2)  
 A\_quats(i,:) = RotationMatrix2Quaternion(A\_cell{i}(1:3,1:3));  
 end  
 for i = 1:size(B\_cell, 2)  
 B\_quats(i,:) = RotationMatrix2Quaternion(B\_cell{i}(1:3,1:3));  
 end  
 end

This function inputs the four matrices contained in the *data\_quaternion* input files, parses them into readable cells, normalizes the quaternions, and transforms them into the correct frames. The next function used is *M\_set( )*, which receives the two sets of quaternions created by *relative\_motion\_matrix( )* and constructs the matrix M as described in the Methods for Solution section.

function [M] = M\_set(quat\_set1, quat\_set2)  
 %Accepts two matrices of arbitrary length with quaternions as rows and produces the full M matrix used  
 %for hand-eye calibration problems.  
 if size(quat\_set1, 1) ~= size(quat\_set2)  
 disp('[WARNING] Different numbers of quaternion measurements provided.')  
 num\_quats = min(size(quat\_set1, 1), size(quat\_set2));  
 else  
 num\_quats = size(quat\_set1, 1);  
 end  
 % M = zeros(num\_quats \* 4, 4);  
 M = [];  
 for i=1:num\_quats  
 [M\_i] = M\_quat(quat\_set1(i, :)', quat\_set2(i, :)');  
 M = [M; M\_i];  
 end  
 end

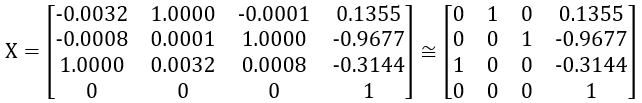
These two functions are both used by *EyeinHand\_Calibration( )*, which as the name suggests, performs the entire Eye in Hand calibration method.

function [Tx, Px\_all, A\_cell, B\_cell] = EyeinHand\_Calibration(q\_Robot\_config, q\_camera\_config, t\_Robot\_config, t\_camera\_config)  
 %This function receives and input of the four matrices and performs eye in  
 %hand calibration  
  
 [A\_quats, B\_quats, A\_cell, B\_cell] = relative\_motion\_matrix(q\_Robot\_config, q\_camera\_config, t\_Robot\_config, t\_camera\_config);  
  
  
 [M\_all] = M\_set(A\_quats, B\_quats);  
 [U, S, V] = svd(M\_all);  
 qx = V(:, end);  
 Rx = Quaternion2RotationMatrix(qx);  
  
 % solve ls problem  
 Px\_all = zeros(3, size(A\_quats, 1));  
 A = [];  
 b = [];  
 for i = 1:size(Px\_all, 2)  
 A\_mat = (A\_cell{i}(1:3, 1:3) - eye(3));  
 Px\_all(:, i) = pinv(A\_mat) \* (Rx \* B\_cell{i}(1:3, 4) - A\_cell{i}(1:3, 4));  
 A = [A; A\_cell{i}(1:3, 1:3) - eye(3)];  
 b = [b; Rx \* B\_cell{i}(1:3, 4) - A\_cell{i}(1:3, 4)];  
 end  
 Px = lsqr(A, b);  
  
 % Px = zeros(3, 1);  
 Tx = [Rx Px; zeros(1, 3), 1];  
 end

This function receives the input information from the data files, parses the information into useful quaternions using *relative\_motion\_matrix()*, assembles the M matrix using *M\_set()*, and then performs singular value decomposition on *M* to find the rotation *Rx*. Next, the least squares problem is solved by populating *A* and *b* matrices and solving the least squares problem. Finally, the Eye in Hand algorithm outputs the transformation matrix *Tx*.

*Answer / Test Cases:*

The transformation matrix between the ‘Eye’ and ‘Hand’ that was calculated using this data is:



This result is reasonable given our expectations of the problem. Although we are not given information about the orientations of the eye or hand frames, the z-axis of the hand is most likely the axis about which the final wrist joint rotates and extends away from the robot into space. This choice of frames is the most standard way of defining them. Taking this to be true, our resulting transformation matrix indicates that the orientation of the eye frame is nearly orthogonal to the hand frame, but instead has its x-axis facing in very nearly the same direction as the hand frame’s z-axis. Looking at the translation vector, the eye frame is translated in the negative z-direction, that is further back from the hand in its z-direction, with some x- and y-offsets that are likely explained by the offsets necessary to mount the camera to the robot. Since it is highly likely that the hand and eye are defined with frames that are roughly orthogonal when mounted, and the translation vector is of reasonable magnitude with a realistic breakdown of its xyz components, we are confident in this answer.

**PA2-2) (Bonus) Eye in Hand, Noisy**

*Problem Description:*

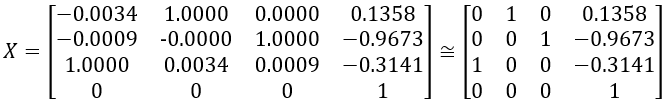
This goal requested the solution to the unknown transform *X* from the camera frame to the robot end-effector frame using the quaternion analytical approach. This time the data is noisy, and in a second scenario we are only using half of the data sets at a time.

*Method for Solution:*

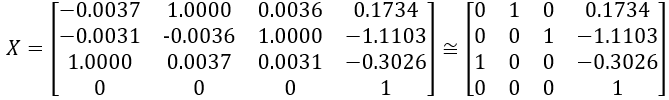
For this goal, the solutions are found with the same programs as used in PA2-1. The only difference is that we are instead reading from data\_quaternion\_noisy, and that for part b, we only read half the data at a time.

*Answer / Test Cases:*

Using the data produced by the provided *data\_quaternion\_noisy()* function and solving the problem the same way as shown in PA2-1), the following transformation matrix between the hand and eye was calculated:



This result is very similar to that of the previous problem. If we instead only consider half of the noisy data, the matrix is instead calculated as:



This result does differ more significantly from the original matrix calculated with the noise-free data than the previous one. This suggests that the fully noisy dataset is sufficiently large with sufficiently evenly-distributed noise that the result, which is effectively an indirect averaged result of the individual measurements, is not altered significantly by the presence of the noise. This appears to not be the case when fewer noisy measurements are considered, which does noticeably result in an orientation that differs a bit more from the reasonable expectation of orthogonal hand and eye frames.

To further compare these results, we computed the results of the calculations

Where and are the computed transformations between each set of camera and robot data for each frame. Under the assumption that the measurements are perfectly reliable (which is not necessarily the case here, as the presence of noise in the data is known), these two calculations should yield the same results if the matrix *T* is accurate. By computing the mean-squared error between the results of these calculations for each frame of data, we can quantify the adherence of the final output transformation to the collected data. This was done for the three solutions thus far shown coming from the three different sets of data and is summarized in the table below:

*Table 4. MSE in PA2-2*

| Dataset | MSE |
| --- | --- |
| Full, noiseless | 0.0871 |
| Full, noisy | 0.0871 |
| Partial, noisy | 0.0525 |

These results again confirm the similarity between the transformations produced by considering the full noiseless and noisy datasets. Notably, the MSE associated with the transformation produced by considering a partial set of the noisy data is lower than the others, which is to be expected as a more perfect fit is likely to be obtained when fitting to fewer points, regardless of the presence of noise.

Other Relevant Code

The code used to perform the entirety of PA1 is shown below.

clc; clear; format compact;  
  
 [package1] = read\_txt('pa1-debug-a-calbody.txt'); % works (contains di, ai, ci)  
 [package2] = read\_txt('pa1-debug-a-calreadings.txt'); % works (contains n\_frames, each of di, ai, ci)  
 [package3] = read\_txt('pa1-debug-a-empivot.txt'); % works (contains n\_frames, each of gi)  
 [package4] = read\_txt('pa1-debug-a-optpivot.txt'); % works (contains n\_frames, each of di, hi)  
 [package5] = read\_txt('pa1-debug-a-output1.txt'); % works (contains n\_frames, each of ci)  
  
 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
 % Checking functionality of correspondence registration algorithm  
 a\_set = package3{1};  
 b\_set = package3{2};  
  
 [T, a\_bar, b\_bar] = correspondence\_registration(a\_set, b\_set);  
  
 a1 = [a\_set(4, :) 1]';  
 b1 = [b\_set(4, :) 1]';  
 T \* a1;  
 T \ b1;  
  
 % Checking equivalency of centroids  
 T(1:3,1:3)\*a\_bar' + T(1:3,4);  
 b\_bar;  
 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
  
 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
 % 3)  
 disp('------------3)---------------')  
 % Repackaging the original object for ease of use  
 em\_package = cell(1, size(package2, 2));  
 for i = 1:size(package2, 2)  
 em\_package{i} = package2{i}{1};  
 end  
  
 calobject\_package = cell(1, size(package2, 2));  
 for i = 1:size(package2, 2)  
 calobject\_package{i} = package2{i}{2};  
 end  
  
 calobject\_package2 = cell(1, size(package2, 2));  
 for i = 1:size(package2, 2)  
 calobject\_package2{i} = package2{i}{3};  
 end  
  
 % Calibration using the optical markers  
 [T\_bank\_em] = opt\_calibration(package1{1}, em\_package);  
 FD\_bank = T\_bank\_em  
  
 [T\_bank\_cal] = opt\_calibration(package1{2}, calobject\_package);  
 FA\_bank = T\_bank\_cal  
  
 % Computing c\_expected  
 c\_expected = cell(1, size(package2, 2));  
 for i = 1:size(package2, 2)  
 c\_sub = zeros(size(package2{1}{3}));  
 for j = 1:size(package2{1}{3}, 1)  
 % inter = FD\_bank{i} \ FA\_bank{i} \* [calobject\_package2{i}(j, :) 1]';  
 inter = FD\_bank{i} \ FA\_bank{i} \* [package1{3}(j, :) 1]';  
 c\_sub(j, :) = inter(1:3)';  
 end  
 c\_expected{i} = c\_sub;  
 end  
  
 % c\_expected{1} - calobject\_package2{1}  
 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
  
 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
 % 4)  
 disp('------------4)---------------')  
 [b\_tip\_em, b\_post\_em, ~] = pivot\_calibration(package3)  
 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
  
 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
 % 5)  
 disp('------------5)---------------')  
 % first transform optical probe measurements into em frame  
 em\_package2 = cell(1, size(package4, 2));  
 for i = 1:size(package4, 2)  
 em\_package2{i} = package4{i}{1};  
 end  
 [T\_bank\_em2] = opt\_calibration(package1{1}, em\_package2);  
  
 h\_em = cell(1, size(package4, 2));  
 for i = 1:size(package4, 2)  
 h\_id = zeros(size(package4{i}{2}));  
 for j = 1:size(package4{i}{2}, 1)  
 p\_i = T\_bank\_em2{i} \ [package4{i}{2}(j, :) 1]';  
 % p\_i = [package4{i}{2}(j, :) 1]'  
 h\_id(j, :) = p\_i(1:3)';  
 end  
 h\_em{i} = h\_id;  
 end  
  
 % now perform pivot calibration in same way as before  
 [b\_tip\_opt, b\_post\_opt, ~] = pivot\_calibration(h\_em)  
 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
  
 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
 % Write to output file  
  
 % creating new package object (in same form as read package5)  
 repackage = cell(size(package5));  
 repackage{1} = b\_tip\_em';  
 repackage{2} = b\_tip\_opt';  
  
 NC = size(c\_expected{1}, 1);  
 bookmark = 0;  
 for i = 3:size(repackage, 2)  
 repackage{i} = c\_expected{i - 2};  
 % repackage{i} = calobject\_package2{i - 2}; % this should be wrong, but it matches the provided debug output;  
 end  
  
 write\_txt(repackage);  
 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
  
 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
 % Comparison  
 fprintf('\n\n\n')  
 disp('------------Comparison of Results---------------')  
 for i = 1:size(repackage, 2)  
 fprintf('-----%i-----\n', i)  
 diff = round(repackage{i}(:, :) - package5{i}(:, :), 2)  
 % diff = max(abs(round(repackage{i}(:, :) - package5{i}(:, :), 2)))  
 end

The *opt\_calibration( )* is shown below.

function [T\_bank] = opt\_calibration(calbody, calreading)  
 %UNTITLED4 Summary of this function goes here  
 % Detailed explanation goes here  
 T\_bank = cell(1, size(calreading, 2));  
  
 % calculating Ts between frames using registration  
 for i = 1:size(calreading, 2)  
 T\_bank{i} = correspondence\_registration(calbody, calreading{i});  
 end  
 end

Contributions of Group Members

Jared Rosenbaum completed Homework Assignment PA1-2, 1-4

Steven Swanbeck completed PA 1-1, 1-3

Responsibility was shared on PA 2-1, 2-2, 1-5

References

1. Murray, R.M., Li, Z., & Sastry, S.S. (1994). A Mathematical Introduction to Robotic Manipulation (1st ed.). CRC Press. <https://doi.org/10.1201/9781315136370>
2. Lynch, K. M., & Park, F. C. (2017). *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press.