Homework/Programming Assignment #3 **Homework Due: 04/29/2022- 5:00PM**

Name/EID: Bryant Zhou /YZ26659
Email: YZ009@utexas.edu
Signature (*required*)
I/We have followed the rules in completing this Assignment.

Name/EID: Jonathan Salfity /JS222238
Email: J.Salfity@utexas.edu
Signature (*required*)
I/We have followed the rules in completing this Assignment.

Question	Points	Total
HA 1	100	
PA1	80	
PA2.1	20	
PA2. 2 (Bonus)	10	

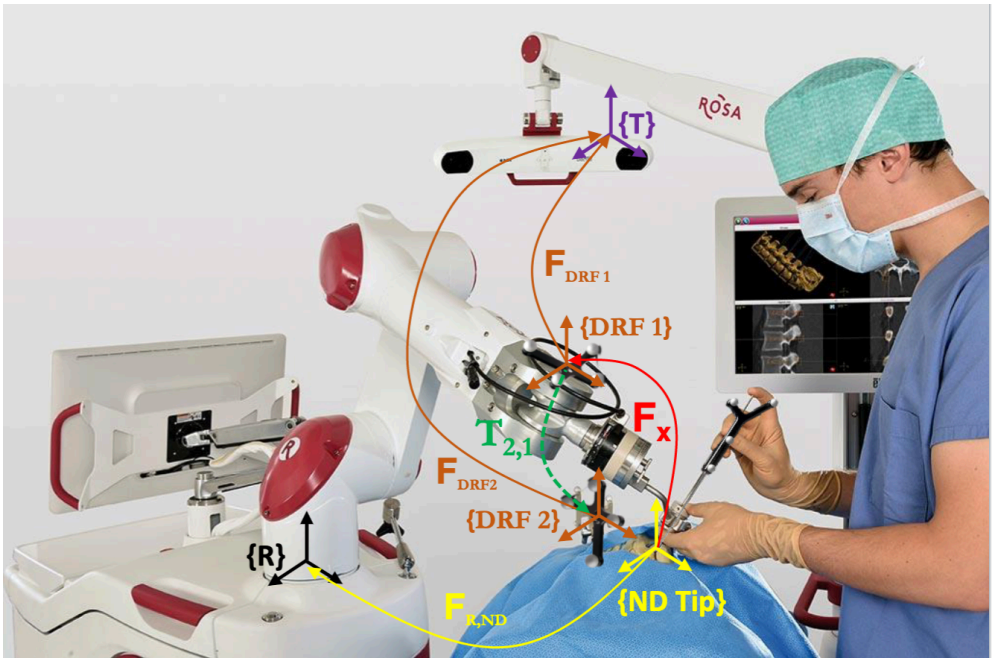
Instruction:

1. Remember that this is a graded assignment. It is the equivalent of a **take-home exam**.
2. **For PA questions**, you need to write a report showing how you derived your equations, describes your approach, test functions, and discusses the results. You should show your test results for each function.

3. You are to work **alone** or **in teams of two** and are **not to discuss the problems with anyone** other than the TAs or the instructor.
4. It is open book, notes, and web. But you should cite any references you consult.
5. Unless I say otherwise in class, it is due before the start of class on the due date mentioned in the P/H Assignment.
6. **Sign and append** this score sheet as the first sheet of your assignment.
7. Remember to submit your assignment in the Canvas.

HA #1

The following diagram is from Lecture Note W12-L2-1.



Our goal for this question is to find the transformation between the DRF1 and the tip of the ND, which is denoted as F_x in the figure above. $F_{T,ND}$ and $F_{R,ND}$ are given according to the question statement. Also, we know the transformation between the $DRF1$ frame and the camera frame, T_{DRF1}, and the transformation between the $DRF2$ frame and the camera frame, T_{DRF2}. Thus, we know the transformation between them, denoted by $T_{2,1}$. By defining the unknown transformation $T_{2,R}$ to be the transformation between the frame R and the $DRF2$ frame, we have the following equation

$$T_{2,1}F_x = F_{2,R}F_{R,ND}$$

Assuming we have k measurements,

$$T_{2,1_1}F_x = F_{2,R}F_{R,ND_1} \dots T_{2,1_k}F_x = F_{2,R}F_{R,ND_k}$$

The unknown transformation $F_{2,R}$ can be expressed as

$$F_{2,R} = T_{2,1_1}F_x F^{-1}_{R,ND_1}$$

Plugging in the above equation to the governing equation, we have

$$T_{2,1_k}F_x = T_{2,1_1}F_x F^{-1}_{R,ND_1}F_{R,ND_k}$$

Rearranging the above equation, we have

$$T_{2,1_1}^{-1}T_{2,1_k}F_x = F_x F^{-1}_{R,ND_1}F_{R,ND_k}$$

This is in the form of $AY = YB$, thus we can solve for F_x using the technique we learned in class.

ME 397: Algorithm for Sensor-Based Robotics. THA_3 Programming Assignment

Bryant Zhou / Jonathan Salfity

April 28, 2022

PA 1

Goal 1

For this part, we developed a MATLAB function, named “point_registration”, which takes the input of two sets of the point cloud data and returns a transformation matrix to map one set of point clouds to another.

point_registration.m

```
function [T] = point_registration(A,B)
% input:    A  point cloud
%           B  point cloud
% output:   T transformation matrix between two point clouds
% Reference: lecture W10-L2

N = size(A,2); % number of points in the point cloud

a_avg = 1/N*sum(A,2); % mean point of all the point cloud
b_avg = 1/N*sum(B,2);

A_dev = A-a_avg; % deviation from the mean
B_dev = B-b_avg;

% Using the eigenvalue decomposition method to calculate R
H = A_dev*B_dev';

delta = [H(2,3)-H(3,2) H(3,1)-H(1,3) H(1,2)-H(2,1)]';
G = [trace(H) delta';delta H+H'-trace(H)*eye(3)];

[V,D] = eig(G);

q0 = V(1,end);
q1 = V(2,end);
q2 = V(3,end);
q3 = V(4,end);
```

```
q(1) = q0;
q(2) = q1;
q(3) = q2;
q(4) = q3;

R = quaternion_2_rotation(q);

p = b_avg - R*a_avg;

T = [R p;0 0 0 1];

end
```

We tested the function with different provided debug data presented the result here in Table I.

Table I. Results for Goal 1.

Test No.	Dataset	Error norm
1	A, B	4.78e-13
2	B,C	5.04e-13
3	C, D	6.50e-13
4	D, E	2.10e-13
5	E, F	5.87e-13
6	F, G	4.16e-13
7	A, G	2.54e-13
8	D, G	2.63e-13

Goal 2

In this part, we developed a MATLAB function, named “pivot_calibration”, to perform the pivot calibration task. The function takes input of the transformation matrix between the sensor and the pointer of the calibration probe, and it outputs the locations of the calibration post and the marker tip.

pivot_calibration.m

```
function [b_post, b_tip] = pivot_calibration(F)
% input:    F  transformation matrix between sensor and probe
% output:   b_post  position of the calibration post
            b_tip   position of the marker tip
```

```
% Reference: lecture W11-L2
num_frames = size(F,1)/3;

R_all = zeros(3*num_frames, 6);

for i = 1:num_frames
    R_all((i-1)*3+1:(i-1)*3+3,1:3) = F((i-1)*3+1:(i-1)*3+3,1:3);
    R_all((i-1)*3+1:(i-1)*3+3,4:6) = -eye(3);
end

P_all = -F(:,4);

output = pinv(R_all)*P_all;

b_tip = output(1:3,:);

b_post = output(4:6,:);

end
```

Goal 3

In this part, we calculated the expected position of the EM markers on the calibration object. The calculation is based on the transformation F_D between the optical tracker and the EM tracker frames, and the transformation F_A between the calibration object and the optical tracker frames. When calculating F_D, we have the following equation:

$D_j = F_D d_j = R_D d_j + p_D$,

where D_j and d_j are available. We then have:

$D_j = [R_D \; p_D][d_j \; 1]^T$

$D_j([d_j \; 1]^T)^{-1} = [R_D \; p_D]$,

where the inverse is actually the pseudo-inverse here. Same procedures were used for calculating F_A. The expected EM marker locations were then calculated by:

$C_i^{(expected)} = F_D^{-1} F_A c_i$

The implementation is shown in the test file under the section named “Goal 3”. The test script is attached at the end of the first part of this report. We presented the difference between the expected location and the measured location in Table II.

Table II. Results for Goal 3.

Test No.	Dataset	Error norm
1	A	0.02

2	B	2.33
3	C	19.18
4	D	0.11
5	E	32.14
6	F	28.53
7	G	37.43

Goal 4

In this part, we performed the pivot calibration using the EM tracking data. We followed the suggested procedures shown in the assignment file. First, we used the first frame of the pivot calibration data to define the local probe frame. In doing so, we first found the midpoint of the data from the first frame, Then, we computed the location of the EM markers with respect to the midpoint, denoted as g_j. The transformation matrix was calculated as follows:

$G_j = F_G[k]g_j = R_G[k]g_i + p_G[k]$,

where k denotes the frame. With the transformation matrix, the pivot calibration was performed using the same way as demonstrated in “Goal 2”.

The algorithm is implemented in the function “test.m” under section “Goal 4”. The dimple locations are demonstrated in Table III.

Table III. Results for Goal 4.

Test No.	Dataset	Dimple Location
1	A	201.0315 198.3800 219.0379
2	B	234.3169 213.3715 256.6872
3	C	232.6392 203.4099 180.3550
4	D	175.2335 203.2702 212.7573
5	E	176.7096

		238.9282 211.1143
6	F	209.8630 200.0714 199.2998
7	G	209.8630 200.0714 199.2998

Goal 5

In this part, we performed the pivot calibration for the optical tracking probe. We first transformed the measurements from the optical tracker frame to the EM tracker frame. Then the same procedure was performed as the one shown in “Goal 4”.
The algorithm is implemented in the function “test.m” under section “Goal 5”. The dimple locations are demonstrated in Table IV.

Table IV. Results for Goal 5.

Test No.	Dataset	Dimple Location
1	A	-2.2234e+03 0.4767e+03 -2.0778e+03
2	B	236.4789 324.4156 47.4153
3	C	-0.0214e+03 0.1044e+03 -1.3062e+03
4	D	521.1882 396.2295 -166.4366
5	E	-930.4991 -68.5051 202.5211
6	F	429.6714 506.5747 -43.0008
7	G	-79.1684

		400.8392 25.0553

The test script is shown below. Each of the five goals is under its corresponding section, except for “Goal 2” which is only a calibration function that is shown previously.

```
clc
close all
clear all

%% Goal 1
% Extract debug file
A = readmatrix('debug/pa1-debug-a-output1.txt');
A(1,:) = [];
A = A';
B = readmatrix('debug/pa1-debug-b-output1.txt');
B(1,:) = [];
B = B';
C = readmatrix('debug/pa1-debug-c-output1.txt');
C(1,:) = [];
C = C';
D = readmatrix('debug/pa1-debug-d-output1.txt');
D(1,:) = [];
D = D';
E = readmatrix('debug/pa1-debug-e-output1.txt');
E(1,:) = [];
E = E';
F = readmatrix('debug/pa1-debug-f-output1.txt');
F(1,:) = [];
F = F';
G = readmatrix('debug/pa1-debug-g-output1.txt');
G(1,:) = [];
G = G';

T = point_registration(D,G);
R = T(1:3,1:3);
p = T(1:3,4);
b = R*D+p;
error = norm(mean(G-b,2));

disp(['Point cloud registration is completed. The norm of the error is ',...
    num2str(error)])

%% Goal 2

%% Goal 3
calBody_cell = readcell('debug/pa1-debug-g-calbody.txt');
num_d = cell2mat(calBody_cell(1,1));
num_a = cell2mat(calBody_cell(1,2));
num_c = cell2mat(calBody_cell(1,3));
calBody = readmatrix('debug/pa1-debug-g-calbody.txt')';
calReading = readcell('debug/pa1-debug-g-calreadings.txt');
num_frames = cell2mat(calReading(1,4));
```

```
calReading = readmatrix('debug/pa1-debug-g-calreadings.txt')';
idx = 0;
d = zeros(4,num_d);
a = zeros(4,num_a);
d(1:3,:) = calBody(:,1:num_d);
d(4,:) = ones(1,num_d);
a(1:3,:) = calBody(:,num_d+1:num_d+num_a);
a(4,:) = ones(1,num_a);
c = calBody(:,num_d+num_a+1:end);
C_exp = zeros(3,num_frames*num_c);
for i = 1:num_frames
    D = calReading(:,(i-1)*(num_d+num_a+num_c)+1:(i-1)*(num_d+num_a+num_c)+num_d);
    F_D = D*pinv(d);
    F_D = [F_D;0 0 0 1];
    A =
calReading(:,num_d+(i-1)*(num_d+num_a+num_c)+1:(i-1)*(num_d+num_a+num_c)+num_d+num_a);
    C =
calReading(:,num_d+num_a+(i-1)*(num_d+num_a+num_c)+1:(i-1)*(num_d+num_a+num_c)+num_d+n
um_a+num_c);
    F_A = A*pinv(a);
    F_A = [F_A;0 0 0 1];
    F = pinv(F_D)*F_A;
    C_exp(:,(i-1)*num_frames+1:(i-1)*num_frames+num_c) = F(1:3,1:3)*c+F(1:3,4);
    C_error = abs(C_exp(:,(i-1)*num_frames+1:(i-1)*num_frames+num_c)-C);
end

disp(['Expected value of C is calculated, and the difference from the distorted
location ',...
    'of C is ', num2str(norm(C_error))])

%% Goal 4
emPivot_cell = readcell('debug/pa1-debug-g-empivot.txt');
num_G = cell2mat(emPivot_cell(1,1));
num_frame = cell2mat(emPivot_cell(1,2));
emPivot = readmatrix('debug/pa1-debug-g-empivot.txt')';
emPivot(:,1) = [];

G0 = 1/num_G*sum(emPivot(:,1:num_G),2);
g_j = emPivot-G0;

R = zeros(3*num_frame,3);
P = zeros(3*num_frame,1);

for i = 1:num_frame
    G_j = emPivot(:,(i-1)*num_G+1:(i-1)*num_G+num_G);
    g = g_j(:,(i-1)*num_G+1:(i-1)*num_G+num_G);
    g = [g;ones(1,6)];
    F_G = G_j*pinv(g);
    R((i-1)*3+1:(i-1)*3+3,:) = F_G(1:3,1:3);
    P((i-1)*3+1:(i-1)*3+3,:) = F_G(1:3,4);
end

F = [R P];

[b_post, ~] = pivot_calibration(F);
```

```
disp('Pivot calibration of the EM tracking probe. Dimple location is:')
b_post

%% Goal 5
optPivot_cell = readcell('debug/pa1-debug-g-optpivot.txt');
num_D = cell2mat(optPivot_cell(1,1));
num_H = cell2mat(optPivot_cell(1,2));
num_frame = cell2mat(optPivot_cell(1,3));
optPivot = readmatrix('debug/pa1-debug-g-optpivot.txt')';

calBody_cell = readcell('debug/pa1-debug-g-calbody.txt');
num_d = cell2mat(calBody_cell(1,1));
num_a = cell2mat(calBody_cell(1,2));
num_c = cell2mat(calBody_cell(1,3));
calBody = readmatrix('debug/pa1-debug-g-calbody.txt')';

% calculate F_D
D_all = zeros(3,8);
d_all = zeros(4,8);
for i = 1:num_frame
    D = optPivot(:,(i-1)*(num_D+num_H)+1:(i-1)*(num_D+num_H)+num_D);
    d = [calBody(:,1:num_d);ones(1,num_d)];
    D_all = [D_all D];
    d_all = [d_all d];
end
D_all(:,1:8) = [];
d_all(:,1:8) = [];
F_D = D_all*pinv(d_all);
F_D = [F_D;0 0 0 1];
F_D = inv(F_D);

% Transfer from opt frame to em frame
optPivot_em = F_D(1:3,1:3)*optPivot+F_D(1:3,4);

% Calibration
H0 = 1/num_G*sum(optPivot_em(:,1:num_H),2);
h_j = optPivot_em-H0;

R = zeros(3*num_frame,3);
P = zeros(3*num_frame,1);

for i = 1:num_frame
    H_j = optPivot_em(:,(i-1)*num_H+1:(i-1)*num_H+num_H);
    h = h_j(:,(i-1)*num_H+1:(i-1)*num_H+num_H);
    h = [h;ones(1,6)];
    F_H = H_j*pinv(h);
    R((i-1)*3+1:(i-1)*3+3,:) = F_H(1:3,1:3);
    P((i-1)*3+1:(i-1)*3+3,:) = F_H(1:3,4);
end

F = [R P];

[b_post, ~] = pivot_calibration(F);
```

```
disp('Pivot calibration of the optical tracking probe. Dimple location is:')
b_post
```

PA 2

Part 1b - Quaternion analytical approach

The Eye in Hand calibration comes down to solving the equation:

$$AX = XB$$

for *X*, which is the transformation between frames *A* and *B*.

We implemented a quaternion based, Eye in Hand calibration algorithm using collected data. Solving the quaternion based algorithm comes down to solving a least squares problem,

$$min \; || M \; q_x ||$$

Where:

$$M = \left[M(q_{A,1}, \; q_{B,1}) \ldots \; M(q_{A,n}, \; q_{B,n}) \right]^T$$

And each matrix

$$M(q_A, q_B) = \left[\begin{array}{c|c} s_A - s_B & -(v_A - v_B)^T \\ \hline (v_A - v_B) & (s_A - s_B) I_3 + sk(v_A + v_B) \end{array} \right]$$

Subject to:

$$||q_x|| = 1$$

Using the Singular Value Decomposition,

$$[U, \; \Sigma, \; V] = \; svd(M)$$

The last column of *V* produces the quaternion, q_x. With q_x, we can find the Rotation Matrix for .

$$R_x$$

Then finally, the displacement vector, p_x, is found through least squares by:

$$(R_{A,k} - \; I) p_x = R_x p_{B,k} - p_{A,k}$$

Part 2 - Noisy Data

After processing the noisy data, we recognize that the outputs are nearly identical. This is most likely because the least squares algorithm finds the same solution when the noise is normally distributed.

When we only use half the data set, interestingly we get the same results.

	R_x	p_x
Clean Data	-0.0032 1.0000 -0.0001 -0.0008 0.0001 1.0000 1.0000 0.0032 0.0008	-0.4154 -0.0886 0.0628
Noisy Data	-0.0034 1.0000 0.0000 -0.0009 -0.0000 1.0000 1.0000 0.0034 0.0009	-0.4154 -0.0885 0.0629
Noisy Data - first 5 points	-0.0037 1.0000 0.0036 -0.0031 -0.0036 1.0000 1.0000 0.0037 0.0031	-0.3229 -0.0429 -0.0834
Noisy Data - last 5 points	-0.0043 1.0000 0.0010 0.0030 -0.0010 1.0000 1.0000 0.0043 -0.0030	0.4278 0.1250 1.6217

test.m

```
clc;
clear all;
close all;
[q_Robot, q_camera, t_Robot, t_camera]  = data_quaternion();
[q_Robot_noisy, q_camera_noisy, t_Robot_noisy, t_camera_noisy]  =
data_quaternion_noisy();
[Rx, px, Tx] = handeye_transform(q_Robot, t_Robot, q_camera, t_camera)
[Rx_noisy, px_noisy, Tx_noisy] = handeye_transform(q_Robot_noisy, ...
                                                   t_Robot_noisy, ...
                                                   q_camera_noisy, ...
                                                   t_camera_noisy)
[Rx_half, px_half, Tx_half] = handeye_transform(q_Robot_noisy(1:5,:), ...
                                                t_Robot_noisy(1:5,:), ...
                                                q_camera_noisy(1:5,:), ...
                                                t_camera_noisy(1:5,:))
```

handeye_transform.m

```
function [R_x, p_x, T_x] = handeye_transform(qA, pA, qB, pB)
%handeye_transform Perform Eye In Hand Calibration, AX=XB Algorithm
   % param: qA (nx4 quaternion measurements)
   % param: pA (nx3 translation vector measurements)
   % param: qB (nx4 quaternion measurements)
```

```
% param: pB (nx3 translation vector measurements)
% return: Rx (Optimal 3x3 Rotation Matrix)
% return: px (Optimal 1x3 translation vector)
% return: Tx (Optimal Transformation Matrix)
% reference: ASBR W12L1 Course Notes
% Check the data for proper dimensions
if size(qA, 2) ~= 4 || size(qB, 2) ~= 4 ...          % proper quaternion
        || size(pA, 2) ~= 3 || size(pA, 2) ~= 3 ... % proper trans vec
        || size(qA, 1) ~= size(pA, 1) ...            % consistent n_obs
        || size(qB, 1) ~= size(pB, 1) ...            % consistent n_obs
        || size(qB, 1) ~= size(qA, 1)                % consistent n_obs
    assert
end
n_observations = size(qA, 1);
% build S1*inv(S2) inv(E1)*E2
qE12 = [];
qS12 = [];
for n = 1:n_observations-1
    R_E1 = quat2rotm(qA(n, :));
    R_E2 = quat2rotm(qA(n+1, :));
    qE12 = [qE12;
            rotm2quat(R_E1'*R_E2)];
    R_S1 = quat2rotm(qB(n, :));
    R_S2 = quat2rotm(qB(n+1, :));
    qS12 = [qS12;
            rotm2quat(R_S1*R_S2')];

end

M = [];
for n = 1:n_observations-1
    % grab info
    sA = qE12(n, 1);
    vA = qE12(n, 2:end)';     % transpose to column vector
    sB = qS12(n, 1);
    vB = qS12(n, 2:end)';     % transpose to column vector
    % build M_qAqB
    M_qAqB = zeros(4);
    M_qAqB(1, 1) = sA-sB;
    M_qAqB(1, 2:end) = -(vA-vB)';
    M_qAqB(2:end, 1) = vA-vB;
    M_qAqB(2:end, 2:end) = (sA-sB)*eye(3) + vec_2_skew_mat(vA+vB);
    M = [M;
         M_qAqB];
end
% quaternion is 4th column of e-vector
[U, S, V] = svd(M);
q_x = transpose(V(:,4));
R_x = quat2rotm(q_x);
% least squares, solving Ax = b
A = zeros(3*n_observations, 3);V
b = zeros(3*n_observations, 1);
for n = 1:n_observations
    idx = 3*(n-1)+1;
    A(idx:(idx+2), 1:3) = quat2rotm(qA(n,:))-eye(3);
```

```
        vA = qA(n, 2:end)';     % transpose to column vector
        vB = qB(n, 2:end)';     % transpose to column vector
        b(idx:(idx+2), 1) = R_x*vB - vA;
    end
    p_x = lsqr(A, b);
    T_x = zeros(4);
    T_x(4,4) = 1;
    T_x(1:3, 1:3) = R_x;
    T_x(1:3, 4) = p_x;
end
```

vec_2_skew_matrix.m

```
function [S] = vec_2_skew_matrix(v)
%vec_2_skew_matrix Convers screw axis vector to skew symmetric
%matrix
%   param: (v) 1x3 screw axis vector
%   return: (S) 3x3 skew symmetrix matrix
%   reference: MR Ch4 Intro paragraph
    if length(v) ~= 3
        assert
    end
    S = [   0       -v(3)    v(2) ;
          v(3)        0     -v(1) ;
         -v(2)       v(1)     0   ];
end
```