PA1-REPORT

Score Sheet

Name 1	
Email	
Other contact information (optional)	
Name 2	
Email	
Other contact information (optional)	
Signature (required)	I (we) have followed the rules in completing this assignment Jiaming Zhang ChongZhe Yang.

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	

I. Mathematics & Algorithms Implementation

This section introduces the mathematical principles and implemented algorithm for the 6 problems in Programming Assignment 1.

0. Scenario

The goal is to perform registration and calibration between the sensors, the layout is shown as follows:

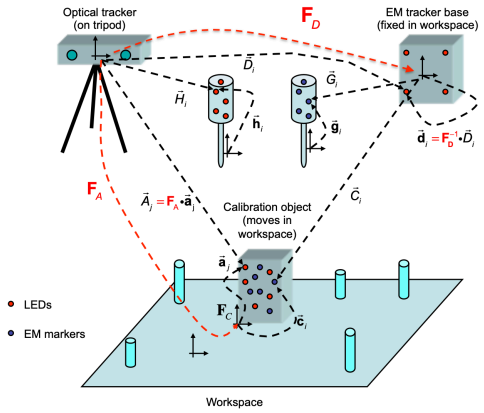

Figure 1

1. Cartesian Math Packages

1) Mathematical Method

Typical rigid body transformations are represented by matrix multiplcations. Suppose there are two Cartesian coordinate frames, A and B. A position vector in frame A and frame B is denoted as $\vec{p}_a$ and $\vec{p}_b$, respectively. $F_{AB} = \{R_{AB}, t_{AB}\}$ represents the rigid body transformation from frame B to frame A. The relationship between $\vec{p}_a$ and $\vec{p}_b$ is $\vec{p}_b = R_{AB}\vec{p}_b + t_{AB}$.

To do this in a more compact way, we can expand the transformation and position vector to their homogeneous form, i.e.

$$F_{AB} = \begin{bmatrix} R_{AB} & t_{AB} \\ \vec{0} & 1 \end{bmatrix}, \quad p = \begin{bmatrix} \vec{p} \\ 1 \end{bmatrix} \tag{1}$$

Hence, the rigid body can be simply represented as $F_1 \cdot F_2$ and $F \cdot p$

2) Code Implementation

A function that computes the homogeneous form of the given transformation frame of position vector is provided in the **"../cispa/DataProcess.py"** . The function "homovec" take a set of 3x1 position vector as input and return the homogeneous form of the set. And the function "dehomovec" is the inverse operation to "homovec".

2. Registration

A 3D point-cloud to point-cloud registration is introduced in this section.

1) Mathematical Method

Our implementation is refferred to Arun's Method [1]. The paper titled “Least-Squares Fitting of Two 3-D Point Sets” describes a non-iterative least-squares approach to match two sets of 3D points.

Suppose we have two point sets and denote them as $P = \{p_i, i \in 0, 1, 2, \ldots, N\}$ and $P' = \{p'_i, i \in 0, 1, 2, \ldots, N\}$, and they are representing the same rigid body described at different poses. Hence, they should follows: $p'_i \approx Rp_i + t$, hence the error is stated as:

$$ei = p'_i - Rp_i - t \tag{2}$$

Arun's method applied Singular Value Decomposition to find the R and t that **Minimize** the following cost function:

$$\Sigma^2 = \sum_{i=1}^{N} ||p'_i - Rp_i - t||^2 \tag{3}$$

And to solve the problem, we took a two step method, i.e. Solve R first and find the corresponding t. To do this, we have to centralize the point sets to annihilate the coupling between rotation and translation. Therefore, the problem reduces to:

$$\hat{R} = \underset{R}{\operatorname{argmax}} \sum ||p'_{ic} - Rp_{ic}||^2 \tag{4}$$

where $p_{ic} = p_i - \bar{p}$, $p'_{ic} = p'_i - \bar{p}'$

And t can be found simply by:

$$\hat{t} = \bar{p}'_i - \hat{R}\bar{p}_i \tag{5}$$

2) Code Implementation

The code is implemented in the **"../cispa/Registration.py"** and the function is defined as **regist_matched_points(X,Y)**. Here X and Y represent two point sets.

We apply numpy.linalg.svd() to solve the Singular Value Decomposition problem. The steps of our algorithm is shown as follows:

Algorithm 1	**Point Cloud to Point Cloud Registration**
INPUT	Two point sets $A = \{a_i\}$ and $B = \{b_i\}$, **DataType**: numpy.array
OUTPUT	Registered transformation $\{\hat{R}$, $\hat{p}\}$, **DataType**: numpy.array
STEP1:	Centralize the point sets $A_c = \{a_{ic} - mean(A)\}$ and $B_c = \{b_{ic} - mean(B)\}$
STEP2:	Sum up the outer product of a_{ic} and b_{ic} , i.e. $H = \sum a_{ic} b_{ic}^T$
STEP3:	Find SVD of H , $H = U\Sigma V^T$
STEP4:	Compute $\hat{R} = V * diag(1, 1, \|V\|\|U^T\|) * U^T$ // Ensure det($\hat{R}$)==1
STEP5:	Compute $\hat{t} = mean(B) - \hat{R}mean(A)$
RETURN:	F = $\{\hat{R}$, $\hat{t}\}$

3. Pivot Calibration

A pivot calibration is introduced in this section.

1) Mathematical Method

The pivot calibration problem is shown as Fig 2, (Image cited from [2])

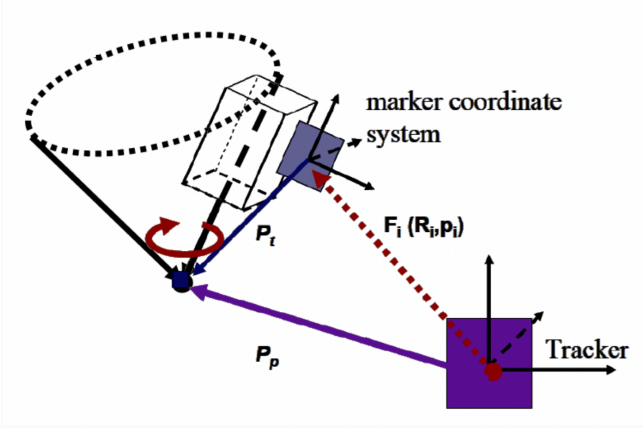

Figure 2

The model of pivot calibration is described as:

$$\vec{p}_{pivot} = F_i \vec{p}_t = R_i \vec{p}_t + \vec{p}_i \tag{6}$$

It holds for all point pairs on the tool. F_i is the rigid transformation from tool coordinate frame to the EM tracker coordinate frame. The calculation of pivot vector can be described as a least square problem like Eq (6).

$$\begin{bmatrix} \cdots & \cdots \\ R_i & -I \\ \cdots & \cdots \end{bmatrix} \begin{bmatrix} p_t \\ p_{pivot} \end{bmatrix} = \begin{bmatrix} \cdots \\ -p_i \\ \cdots \end{bmatrix} \tag{7}$$

To solve Eq (6), we can consider it as a argmin problem as follows:

$$x = \underset{x}{\operatorname{argmax}} ||Ax - b||^2 \tag{8}$$

where $x = \begin{bmatrix} p_t \\ p_{pivot} \end{bmatrix}$, $A = \begin{bmatrix} \cdots & \cdots \\ R_i & -I \\ \cdots & \cdots \end{bmatrix}$ and $B = \begin{bmatrix} \cdots \\ -p_i \\ \cdots \end{bmatrix}$

By solving this least square problem, we can get the pivot vector p_{pivot} .

2) Code Implementation

The code is implemented in the **"../cispa/PivotCalibration.py"**. The function is defined as **calib_pivot_points(F)**. Here F is a list of homogeneous transformations.

We apply numpy.linalg.lstsq() to sovle the least square problem. The steps of our algorithm is shown as follows:

Algorithm 2 Pivot Calibration

INPUT:	A set of Homogeneous Transformations $F = [F_i] = [\{R_i, p_i\}]$, **DataType**: List
OUTPUT:	$x = \begin{bmatrix} p_t \\ p_{pivot} \end{bmatrix}$
STEP1:	Extract R_i and p_i from the input F
STEP2:	Construct $A = \begin{bmatrix} \dots & \dots \\ R_i & -I \\ \dots & \dots \end{bmatrix}$ and $B = \begin{bmatrix} \dots \\ -p_i \\ \dots \end{bmatrix}$
STEP3:	Compute x = numpy.linalg.lstsq(A, B)
RETURN:	$p_t = x[0:3]$ and $p_{pivot} = x[3:6]$

4. Problem 4

Problem Description: Compute the $\vec{C}_i^{expected}$ for distortion calibration data set

1) Mathematical Method

First, we need to perform registration between point sets $D = \{D_j\}$ and $d = \{d_j\}$, where D denotes the optical markers described in the optical tracker frame, d denotes the optical markers described in the EM tracker base frame. They are connected by a transformation : $\vec{D}_j = F_D \cdot \vec{d}_j$. The transformation from optical tracker frame to EM tarcker frame is therefore computed by: $F_D = registration(d, D)$

> Note: "Registration" is the function "regist_matched_points(X,Y)" introduced in the Subsection 2 .

Likewise, we can find the F_A using the same process. F_A denotes the transformation from calibration object and optical tracker frame.

Then the expected position of the markers on the calibration object w.r.t. EM tracker frame can be found by:

$$\vec{C}_i^{expected} = F_D^{-1} F_A \vec{c}_i \tag{9}$$

2) Code Implementation

The code is implemented in **"../PA1/pa1_problem4.py"**.

Algorithm 3	**distortion data expectation**
INPUT:	"../*-calbody.txt" "../*-calreading.txt"
OUTPUT:	$C = \{\vec{C}_i^{expected}\}$
STEP1:	Extract D and A from ../*-calreading.txt
	Extract d and a from ../*-calbody.txt and c from the input data
STEP2:	Compute the corresponding registration for $\{D, d\}$ and $\{A, a\}$
STEP3:	Construct the homogeneous form of c
STEP4:	$\vec{C}_i^{expected} = F_D^{-1} F_A \vec{c}_i$
RETURN:	$C = \{\vec{C}_i^{expected}\}$

5. Problem 5

Problem Description: determine the position relative to the EM tracker base coordinate system of the dimple in the calibration post

1) Mathematical Method

Because the reference frame is unspecified for the calibration post, we have to manually set an local reference frame for it. The EM system coordinates at the k-th frame is $G_i[k]$. Marker coordinates relative to the calibration probe coordinate system are defined as the centralized coordinates of $G_i[k]$, i.e.

$$g_i[k] = G_i[k] - \frac{1}{N_G} \sum_{i}^{N_G} G_i[k] \tag{10}$$

By centralize the point cloud at k=0, the $g_i[0]$ **will be automatically "moved" to the origin of the EM tracker coordinate system**. Since the local reference frame is rigidly connected to the calibration probe. $g_i[k]$ remains the same when k varies. Hence $g_i[0]$ will be taken as the local reference frame to perform pivot calibration.

Now, we gain two representations for one point set w.r.t. local and tracker coordinate system. The transformation from tracker coordinate system to calibration probe coordinate system can be therefore computed as:

$$\vec{G}_j[k] = F_G[k] \cdot \vec{g}_j[0] \tag{11}$$

As we can see, this problem now becomes a point set to point set registration problem. By applying the registration algorithm we developed in previous Subsection 2 , for every pose k, we can find a corresponding transformation $F_G[k]$. Moreover, we have $F_G = \{F_G[k]\}$

p_{dimple} and t_G represents the tip position w.r.t. tracker frame and local frame, respectively. For every pose k, we have:

$$\vec{p}_{dimple} = F_G[k] \cdot \vec{t}_G \tag{12}$$

By applying the pivot calibration algorithm we developed in previous Subsection 3 , we can find the position of calibration probe relative to EM tracker base frame.

2) Code Implementation

The code is implemented in **"../PA1/pa1_problem5.py"**.

Algorithm 4	**EM probe calibration**
INPUT:	"../*-empivot.txt"
OUTPUT:	Calibrated post position p_{dimple} relative to EM tracker frame
INITIALIZE:	Empty sequence F
STEP1:	Extract $G_j[k]$ from ../*-empivot.txt
STEP2:	Compute the local coordinate $g_j[0]$
STEP3:	For $k \in (0, N_{frames})$ perform registration for $F_G[k] = registration(g_j[0], G_j[k])$
STEP4:	Push back $F_G[k]$ to the end of F
STEP5:	Perform pivot calibration for $\{\vec{p}_{dimple}, t_G\} = calibration(F)$
RETURN:	$\vec{p}_{dimple}$

> NOTE: "registration()" is the function "regist_matched_points(X,Y)" introduced in the Subsection 2 ; "calibration()" is the function "calib_pivot_points(F)" introduced in the Subsection 3

6. Problem 6

Problem Description: Perform a pivot calibration of the optical tracking probe

1) Mathematical Method

As shown in the Fig 1 , the transformation from optical tracker to optical probe can be found by registering H_i and h_i. However, to calibrate the optical tracking probe w.r.t. EM tracker, we have to change the base frame from the optical tracker frame to EM tracker frame. Note that frame-change is equivalent to the transform F_D we derived in the previous sections. The frame-change operation is as follows:

$$H_i^{EM} = F_D^{-1} \cdot H_i \tag{13}$$

The remaining steps are essentially the same as what we developed for Problem 5 .

And for pose k, the local frame h_i is derived by:

$$h_i[k] = H_i^{EM}[k] - \frac{1}{N_H} \sum_{i}^{N_H} H_i^{EM}[k] \tag{14}$$

Similarly, $h_i[k] = h_i[0]$ since the points on optical probe remian relative static in the local frame.

Register these two point sets $\{H_i^{EM}[k]\}$ and $\{h_i[k]\}$, and derive the transformation $F_H[k]$ for pose k. And perform pivot calibration for $F_H = \{F_H[k]\}$ to get the position of optical probe described in EM tracker frame p_{dimple}

2) Code Implementation

The code is implemented in **"../PA1/pa1_problem6.py"**.

Algorithm 5	**Optical probe calibration**
INPUT:	"../*-optpivot.txt" & F_D
OUTPUT:	Calibrated post position p_{dimple} relative to EM tracker frame
INITIALIZE:	Empty sequence F
STEP1:	Extract $H_j[k]$ from ../*-optpivot.txt
STEP2:	Compute $H_i^{EM}[k] = F_D^{-1} H_i[k]$
STEP3:	Compute the local coordinate $h_j[0]$
STEP4:	For $k \in (0, N_{frames})$ perform registration for $F_H[k] = registration(h_j[0], H_j^{EM}[k])$
STEP5:	Push back $F_H[k]$ to the end of F
STEP6:	Perform pivot calibration for $\{\vec{p}_{dimple}, t_H\} = calibration(F)$
RETURN:	$\vec{p}_{dimple}$

> NOTE: "registration()" is the function "regist_matched_points(X,Y)" introduced in the Subsection 2 ; "calibration()" is the function "calib_pivot_points(F)" introduced in the Subsection 3

II. Overall Structure

The overall structure for the ../PROGRAMS folder is described as follows:

```
└── PROGRAMS
    ├── PA1              # Test scripts are contained in this directory
    │   ├── Data         # This DIR contains all the provided data
    │   ├── output       # This DIR contains all the result of our program
    │   ├── pa1_main.py              # This is the main process that output the result
    │   ├── pa1_problem2_test.py     # Unit test for registration
    │   ├── pa1_problem3_test.py     # Unit test for pivot calibration
    │   ├── pa1_problem4_test.py     # Unit test for expected distortion
    │   ├── pa1_problem5_test.py     # Unit test for EM probe calibration
    │   └── pa1_problem6_test.py     # Unit test for Optical probe calibration
    └── cispa            # Functions are contained in this directory
        ├── DataProcess.py           # Contains useful functions like skew operation
        ├── HomoRepresentation.py
        ├── LoadData.py              # Load data from text file
        ├── PivotCalibration.py      # Contains function : calib_pivot_points(F)
        └── Registration.py          # Contains function : regist_matched_points(X,Y)
```

III. Unit Testing and Debugging

NOTE: Please change your directory to the ${PROGRAMS} directory before you start the unit test. In our case, the command is:

```
$ cd ~/*PARENT DIR*/PROGRAMS
```

And list all files to check whether you are in the correct directory.

```
$ ls
PA1   cispa
```

1. Registration Unit Test

In the ../PROGRAMS directory, run test script "/PA1/pa1_problem2_test.py". This script is desinged to test the registration function **regist_matched_points**, in the terminal, run the following command:

```
../PROGRAMS $ python PA1/pa1_problem2_test.py -d PA1/Data
```

It will load "/Data/pa1-debug-a-calbody.txt" data and apply an artificial rigid transformation F to the imported point set A , so the transformed point set $B = FA$.

The script will first import the **regist_matched_points(A,B)** function and get a registration result F_{regist} and then compare the registration result $F_{register}$ with the original F and print them out to the terminal. The result of our test case is:

```
../PROGRAMS $ python PA1/pa1_problem2_test.py -d PA1/Data
[11:09:26] DEBUG    Suppose we apply a transformation
                    R = [[-1.0000000e+00 -1.2246468e-16  0.0000000e+00]

                     [ 1.2246468e-16 -1.0000000e+00  0.0000000e+00]

                     [ 0.0000000e+00  0.0000000e+00  1.0000000e+00]]

                    p=[[10.]

                     [10.]

                     [10.]]

           DEBUG    The registered transformation
                    R = [[-1.0000000e+00 -1.2126596e-16  0.0000000e+00]

                     [ 1.2126596e-16 -1.0000000e+00  0.0000000e+00]

                     [ 0.0000000e+00  0.0000000e+00  1.0000000e+00]]

                    p=[10. 10. 10.]

           INFO     Registered rotation matrix is correct
           INFO     Registered translation vector is correct
```

2. Pivot Calibration Unit Test

In the ../PROGRAMS directory, run test script "/PA1/pa1_problem3_test.py". This script is desinged to test the calibration function **calib_pivot_points**, input the following command in the terminal:

```
../PROGRAMS $ python PA1/pa1_problem3_test.py -d PA1/Data
```

It will load "/Data/pa1-debug-a-empivot.txt" data and assume an imaginary probe with a certain $\{p_{tip}, p_{pivot}\}$. For a sequence of rotations $R = \{R_i\}$ derived from the data, its corresponding translations $P = \{p_i\}$ can be found by $p_i = p_{pivot} - R_i p_{tip}$. Combine R and P, we will "made" a sequence of homogeneous transformation F .

The script will first import the **calib_pivot_points(F)** function and get a calibration result $\{p_{tip}^{cal}, p_{pivot}^{cal}\}$; and then compare the calibration result with the original $\{p_{tip}, p_{pivot}\}$ and print them out to the terminal. The result of our test case is:

```
../PROGRAMS $ python PA1/pa1_problem3_test.py -d PA1/Data
[11:12:10] DEBUG    Suppose we have a probe
                    p_t = [[ 1. ]
                     [ 0.5]
                     [-2. ]]
                    p_pivot=[[ 9.8]
                     [10.2]
                     [ 3.4]]
           DEBUG    The calibration result
                    p_t = [[ 1. ]
                     [ 0.5]
                     [-2. ]]
                    p_pivot=[[ 9.8]
                     [10.2]
                     [ 3.4]]
           INFO     Calibration result is correct!
```

3. Problem3 Debug

In the ../PROGRAMS directory, run test script "/PA1/pa1_problem4_test.py". This script is desinged to debug the algorithm 3 to find the expected C using "pa1-debug-a-calbody.txt" and "pa1-debug-a-calreadings.txt"

```
../PROGRAMS $ python PA1/pa1_problem4_test.py
```

The result is compared with the "pa1-debug-a-output1.txt" and shown in the table below.

TABLE 1: pa1_problem4_test.py result for pa1-debug data

	OUR RESULT	GIVEN RESULT
$\vec{C}_i^{expected}$	(208.53810 208.11373 208.77357) (206.48697 211.65187 333.70665) (204.43583 215.19000 458.63973) ...	(209.02, 209.18, 209.56) (209.60, 205.60, 334.51) (210.17, 202.01, 459.45) ...

4. Problem4 Debug

In the ../PROGRAMS directory, run test script "/PA1/pa1_problem5_test.py". This script is desinged to debug the algorithm 4 using "pa1-debug-a-empivot.txt"

```
../PROGRAMS $ python PA1/pa1_problem5_test.py
```

The result is compared with the "pa1-debug-a-output1.txt" and shown in the table below.

TABLE 2: pa1_problem5_test.py result for pa1-debug data

	OUR RESULT	GIVEN RESULT
$\vec{p}_{dimple}^{OP}$	(403.49890 405.26775 208.63184)	(403.50, 405.27, 208.63)

5.Problem5 Debug

In the ../PROGRAMS directory, run test script "/PA1/pa1_problem6_test.py". This script is desinged to debug the algorithm 5 using "pa1-debug-a-empivot.txt", "pa1-debug-a-calbody.txt" and "pa1-debug-a-calreadings.txt"

```
../PROGRAMS $ python PA1/pa1_problem6_test.py
```

The result is compared with the "pa1-debug-a-output1.txt" and shown in the table below.

TABLE 3: pa1_problem6_test.py result for pa1-debug data

	OUR RESULT	GIVEN RESULT
$\vec{p}_{dimple}^{EM}$	(200.02611 199.51553 195.06558)	(200.02, 199.52, 195.07)

6. Discussion for the results

Table 4: 2-Norm of the debug datasets

DataSet	$C_i^{expected}$	p_{dimple}^{EM}	p_{dimple}^{OP}
a	0.0048	0.0037	0.0075
b	0.2424	0.0018	0.0060
c	0.3449	0.5538	0.0059
d	0.0100	0.0068	0.0058
e	1.5046	0.3100	0.0100
f	1.5998	0.5632	0.0062
g	1.5747	0.3029	0.0068

In this assignment, we adopt 2-Norm to evaluate the implemented programs. And the results are shown in the TABLE 4. Since the final outputs of these modules are position vectors in 3D Euclidean space, it is reasonable and intuitive that Euclidean Distance between predictions and ground-truths can serve as error and evaluate the performance. Driven by this, we compute the average 2-Norm for all the output position vectors in each case.

Datasets e,f,g are somehow noisy and the expected error is greater than a,b,c. The pivot calibration on dimple for EM tracking system is much noisier than the calibration result derived from Optical tracking system.

IV. Result Tabulars

Perform similar steps for Group h to Group k by running the command in terminal

```
$ python PA1/pa1_main.py -n pa1-unknown-h
$ python PA1/pa1_main.py -n pa1-unknown-i
$ python PA1/pa1_main.py -n pa1-unknown-j
$ python PA1/pa1_main.py -n pa1-unknown-k
```

The results for calibrated probe position relative to EM tracker system $\vec{p}^{EM}$ and $\vec{p}^{OP}$are shown in the TABLE 5.

Table 5: result for unknown data

DataSet	p^{EM}	p^{OP}
h	(210.7992, 195.5491, 193.2549)	(394.5904, 399.9720, 192.8282)
i	(206.1700, 201.6335, 210.1792)	(404.0700, 398.2292, 203.9080)
j	(191.2612, 190.1500, 218.6504)	(397.6587, 408.1820, 202.7904)
k	(191.1497, 201.2274, 187.1436)	(402.1888, 403.1132, 197.8957)

Contributions

Jiaming Zhang developed problem 1,2,3,4 ; Chongjun Yang developed problem 5,6 of this PA. Both team member complete a part of this report.

References

[1] K. S. Arun, T. S. Huang and S. D. Blostein, "Least Squares Fitting of Two 3D Point Sets", in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.PAMI-9, no.5

[2] https://cas-assignment.readthedocs.io/en/latest/assignment.toolcalibration.html