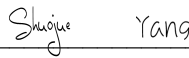



## Programming Assignments 1 & 2 601.455 and 601/655 Fall 2021

Please also indicate which section(s) you are in (one of each is OK)

### Score Sheet

Name 1	Shuo j u e Yang
Email	syang131@j h. edu
Other contact information (optional)	
Name 2	Zi j i an Wu
Email	zwu52@j hu. edu
Other contact information (optional)	
Signature (required)	<p>I (we) have followed the rules in completing this assignment</p> <div style="text-align: center; margin-top: 20px;">               _____         </div> <div style="text-align: center; margin-top: 20px;">               _____         </div>

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	

## Overview

A. In this section, we developed a **Cartesian math package** to realize rigid body transformation and some other useful mathematical operations.

### 1. Method overview

Suppose there are two Cartesian coordinate frames, A and B. One position vector in frame A and frame B is denoted as  $\vec{p}_a$  and  $\vec{p}_b$ , respectively.  $F_{AB} = [R_{AB}, \vec{t}_{AB}]$  represents the rigid body from frame B to frame A. The relationship between  $\vec{p}_a$  and  $\vec{p}_b$  is

$$\vec{p}_a = F_{AB}\vec{p}_b = R_{AB}\vec{p}_b + \vec{t}_{AB} \quad (1)$$

Note that the inverse of rigid body transformation can be denoted as

$$F_{AB}^{-1} = F_{BA} = [R_{AB}^T, -R_{AB}^T\vec{t}_{AB}] \quad (2)$$

When there is another Cartesian coordinate frame C. The transformation from frame C to frame A is

$$F_{AC} = F_{AB}F_{BC} = [R_{AB}R_{BC}, \vec{t}_{AB} + R_{AB}\vec{t}_{BC}] \quad (3)$$

### 2. Implementation

We developed the *Cartesian\_transformation* class, which has two member variables, R and t. R and t represent the rotation part and translation part of the rigid body transformation. This class can realize the following functions:

- Create a class object

To create the class object, we need a dictionary as the parameter. This dictionary contains 2 key-value pairs which store rotation matrix and translation vector respectively. When calling this class object, the input parameter should be a  $3 \times 1$  vector.

- Inverse operation of rigid body transformation

*Cartesian\_transformation* class contains a member function *.inverse()* to calculate the inverse of rigid body transformation.

- Multiple rigid body transformation

In order to facilitate the calculation of multiple rigid body transformations, we overload the multiplication operator. When the left and right sides of the multiplication sign are both *Cartesian\_transformation* class objects, the result of multiplication is a class object, whose rotation and translation are calculated according to equation(3).

B. In this section, we developed a **point-cloud-to-point-cloud registration algorithm** to compute the transformation between two coordinates.

### 1. Method overview

- Registration error

To tackle the registration task, two given point sets  $\{a_i\}$  and  $\{b_i\}$  are fed into the algorithm to compute the transformation minimizing the registration error between these two point sets. Typically, the transformations are represented by Cartesian Coordinate Transformation group defined in Sec. Overview. A and the goal can be formulated as following:

$$\arg \min_{F=(R, \vec{p})} \sum_i^N ||R \vec{a}_i + \vec{p} - \vec{b}_i||^2 \quad (4)$$

- Transferring to least-squares problem

To minimize the registration error between the two point sets, we transfer Eq. (4) into a least-squares problem. Specifically,  $\{a_i\}$  and  $\{b_i\}$  subtract their midpoints' coordinates and generate  $\{\tilde{a}_i\}$  and  $\{\tilde{b}_i\}$ , respectively. Thus, the rotation and translation parts are decoupled and the minimization problem defined by Eq. (4) is reduced in two subsequent problems:

$$\hat{R} = \arg \min_R \sum_i^N ||R \tilde{a}_i - \tilde{b}_i||^2 \quad (5)$$

$$\hat{\vec{p}} = \hat{R} \bar{a} - \bar{b} \quad (6)$$

where  $\bar{a}$  and  $\bar{b}$  denote the midpoint of  $\{a_i\}$  and  $\{b_i\}$ , respectively. We use least-squares method to compute the  $\hat{R}$  and then further obtain the  $\hat{\vec{p}}$  based on the estimated rotation. In this assignment, we used Singular Value Decomposition (SVD) based method [1] to solve Eq. (5)

### 2. Algorithm and implementation

The algorithm is shown in Alg. 1, in which we input two point sets and obtain the estimated coordinate transformation of these two points.

---

**Algorithm 1:** Point-cloud-to-point-cloud registration

---

**Input:** two point sets  $\{a_i\}$  and  $\{b_i\}$ ;  
**Output:**  $F = [\hat{R}, \hat{p}]$ ;  
1 **Initialization:** extremely small value  $\varepsilon = 10^{-6}$ ;  
2 **Compute**  $\{\tilde{a}_i\}$  and  $\{\tilde{b}_i\}$  by  $\{a_i\} - \bar{a}$  and  $\{b_i\} - \bar{b}$   
3 **Compute**  $H = \sum_i \tilde{a}_i (\tilde{b}_i)^t$   
4 **Compute** SVD of  $H$ :  $H = U\Lambda V^t$   
5 **if**  $|\det(H) - 1| < \varepsilon$  **then**  
6      $\hat{R} = VU^t$   
7 **else if**  $|\det(H) + 1| < \varepsilon$  **then**  
8     **if** 0 in  $\Lambda$  **then**  
9          $U.col[-1] * = -1$ ;                     // Inverse signs of the last column of  $H$   
10          $\hat{R} = VU^t$   
11     **else**  
12         Skip this data point  
13     **end**  
14 **end**  
15 **Compute**  $\hat{p} = \hat{R} \bar{a} - \bar{b}$

---

C. In this section, we develop a package to **calibrate the pivot vector relative to tracker coordinate frame**.

1. Method overview

As shown in Fig.1, the calculation of pivot vector can be described as a least square problem. By solving this least square problem, we can get the pivot vector. The model of pivot calibration is:

$$\vec{p}_{pivot} = F_i \vec{p}_t, \quad (7)$$

in which  $F_i$  is the rigid transformation from tool coordinate frame to the EM tracker coordinate frame. Eq. (7) can be rewritten as:

$$\begin{bmatrix} \vdots & \vdots \\ R_i & -I \\ \vdots & \vdots \end{bmatrix} \cdot \begin{bmatrix} \vec{p}_t \\ p_{pivot} \end{bmatrix} = \begin{bmatrix} \vdots \\ -\vec{p}_i \\ \vdots \end{bmatrix} \quad (8)$$

Eq. (8) can be transferred to the following least square problem:

$$\vec{q}^* = \arg \min_{\vec{q}} \sum_k \left\| A \vec{q} - B \right\|^2 \quad (9)$$

in which  $\vec{q} = \begin{bmatrix} \vec{p}_t \\ \vec{p}_{pivot} \end{bmatrix}$ ,  $A = \begin{bmatrix} \vdots & \vdots \\ R_i & -I \\ \vdots & \vdots \end{bmatrix}_{3k \times 6}$ ,  $B = \begin{bmatrix} \vdots \\ -\vec{p}_i \\ \vdots \end{bmatrix}_{3k \times 1}$ ,  $k$  is the frame number for pivot calibration.

## 2. Algorithm and implementation

To solve the least square problem mentioned above, we utilize the function of the linear algebra module of NumPy - `numpy.linalg.lstsq(a, b[, rcond])`<sup>1</sup>

As shown in Alg. 2, when calling *Pivot\_calib* class, we input a sequence of *Cartesian\_transformation* class objects representing the rigid transformation in each frame.

---

### Algorithm 2: Pivot calibration algorithm

---

**Input:** *Cartesian\_transformation* class object set  $\{F_i\}$ ;

**Output:**  $\vec{q}^* = \begin{bmatrix} \vec{p}_t \\ \vec{p}_{pivot} \end{bmatrix}$ ;

**1 Obtain**  $\{R_i\}$  and  $\{\vec{p}_i\}$  from  $\{F_i\}$

**2 Construct**  $A = \begin{bmatrix} \vdots & \vdots \\ R_i & -I \\ \vdots & \vdots \end{bmatrix}_{3k \times 6}$ ,  $B = \begin{bmatrix} \vdots \\ -\vec{p}_i \\ \vdots \end{bmatrix}_{3k \times 1}$

**3 Compute**  $\vec{q}^* = \text{numpy.linalg.lstsq}(A, B)$

---

D. In this section, we follow the procedures given by instructions to **compute the expected C coordinates**.

### 1. Method overview

According to the procedures, we have mathematics conclusions as following:

$$\vec{D}_i = F_D \vec{d}_i \quad (10)$$

---

<sup>1</sup><https://numpy.org/doc/stable/reference/generated/numpy.linalg.lstsq.html>

$$\vec{A}_i = F_A \vec{a}_i \quad (11)$$

$$\vec{C}_i = F_D^{-1} F_A \vec{d}_i \quad (12)$$

## 2. Algorithm and implementation

We have four point sets in this task,  $a_i$  corresponding to  $A_i$  and  $d_i$  corresponding to  $D_i$ . To obtain the Cartesian Coordinate Transformations ( $F_D$  and  $F_A$ ) that satisfy Eq. (10) and Eq. (11), respectively. Then we can use the point-cloud-to-point-cloud method described in Alg. 1 to calculate the transformation  $F_D$  and  $F_A$ . Finally, we can calculate the expected  $C_i$  coordinates relative to the EM system by Eq. (12) for each frame.

To verify the implemented modules, we have computed the expected  $C_i$  for cases ‘-debug-a’ to ‘-debug-f’ in given data files and calculated the L2-Norm with corresponding ground-truths (‘NAME-OUTPUT1.txt’ files of ‘-debug’ cases). The results are shown in Table 1, in which we can observe that our implemented modules achieve the average L2-Norm of all the cases as 0.7859.

E. In this section, we perform **pivot calibration utilizing the *Pivot\_calib* class developed previously and the marker data relative to EM tracker system.**

### 1. Method overview

We choose the first frame as the start frame and then establish a local coordinate system attached on the tool. We define the local and EM system coordinates at the  $k$ -th frame as  $g_i[k]$  and  $G_i[k]$ , respectively. Marker coordinates relative to the tool coordinate system are defined as following:

$$g_i[0] = G_i[0] - \frac{1}{N_G} \sum_i^{N_G} G_i[0] \quad (13)$$

where  $g_i[0]$  and  $G_i[0]$  respectively denote coordinates relative to the local and EM coordinate system in the first frame. The local coordinate system is attached on the tool, which means that the markers’ local coordinates are fixed in each frame:

$$g_i[k] = g_i[0], \quad k \in \{0, 1, \dots, N_G\} \quad (14)$$

Driven by this observation, the transformation registering the  $g_i[0]$  to  $G_i[k]$  is the  $F_G[k]$ , which gives the position and orientation of the pointer body at frame  $k$  with respect to tracker coordinate system. Thus by using the registration package developed in Sec.B., we obtain a sequence of rigid transformations  $\{F_G[0], F_G[1], F_G[2], \dots, F_G[N_{Frame}]\}$ . Specifically, we can formulate the problem into computing a transformation minimizing the registration error as following:

$$F_G[k] = \underset{F=(R, \vec{p})}{argmin} \sum_i^{N_G} ||R \vec{g}_i[0] + \vec{p} - \vec{G}_i[k]||^2 \quad (15)$$

Finally, using the pivot calibration package developed in Sec.C., we get the pivot vector relative to the EM tracker system.

## 2. Algorithm and implementation

The step by step procedure is shown in Alg. 3.

---

### Algorithm 3: EM prob calibration algorithm

---

**Input:** coordinates relative to EM tracker system in different frames  $\{\{G_i[0]\}, \{G_i[1]\}, \dots, \{G_i[k]\}, \dots\}$ ;  
**Output:** dimple position relative to EM tracker system  $\vec{p}_{dim}$ ;  
**1 Initialization:** number of frames  $N_{Frame}$ ; transformation sequence  $\mathbb{T} = \emptyset$ ;  
**2 Compute** local coordinates  $\{g_i[0]\}$  by Eq. (14) and define the local coordinate system attached on pointer body  
**3 for**  $k = 1, \dots, N_{Frame}$  **do**  
**4**     **Compute**  $F_G[k]$  by registering  $\{g_i[0]\}$  to  $\{G_i[k]\}$  (Eq. (15))  
**5**     Push  $F_G[k]$  at the back of the transformation sequence  $\mathbb{T}$   
**6 end**  
**7 Compute**  $\vec{p}_{dim}$  by using Alg. 2 with the transformation sequence  $\mathbb{T}$  as input

---

F. In this section, we perform **pivot calibration** utilizing the *Pivot\_calib* class developed previously and the marker data **relative to optical tracker system**.

## 1. Method overview

The solution to this section is similar to the method for solving Sec.E.. One difference is that the optical tracker is not fixed in workspace. Therefore, the coordinates of the markers relative to the optical tracker system should first be transformed to relative to the EM tracker system. We define the coordinates of markers on EM tracker base relative to the optical tracker system and relative to the EM tracker system as  $\vec{D}_i$  and  $\vec{d}_i$ , respectively. According to Sec.D., the relationship between  $\vec{D}_i$  and  $\vec{d}_i$  can be expressed as Eq. (10). We first extract the set of  $\vec{D}_i$  from the marker data relative to optical tracker system. Then using the method introduced in Sec.D., we can calculate the  $F_D$ . The other coordinates we can obtain is the set of  $H_i$ , which is the markers' coordinates on the tool relative to the optical tracker system. According to the previous definition, there is:

$$H\_EM_i = F_D^{-1}H_i, \quad (16)$$

where  $H\_EM_i$  is the markers' coordinates on the tool relative to the EM tracker system. The remaining steps are exactly the same as the procedure mentioned in Sec.E..

We also choose the first frame as the start frame and then establish a local coordinate system attached on the optical tool. We define the local and EM system coordinates at the  $k$ -th frame as  $h_i[k]$  and  $H\_EM_i[k]$ , respectively. Marker coordinates relative to the tool coordinate system are defined as following:

$$h_i[0] = H\_EM_i[0] - \frac{1}{N_H} \sum_i^{N_H} H\_EM_i[0] \quad (17)$$

where  $h_i[0]$  and  $H\_EM_i[0]$  respectively denote coordinates relative to the local and EM coordinate system in the first frame. The local coordinate system is attached on the tool, which means that the markers' local coordinates are fixed in each frame:

$$h_i[k] = h_i[0], \quad k \in \{0, 1, \dots, N_H\} \quad (18)$$

Driven by this observation, the transformation registering the  $h_i[0]$  to  $H\_EM_i[k]$  is the  $F_H[k]$ , which gives the position and orientation of the pointer body at frame  $k$  with respect to tracker coordinate system. Thus by using the registration package developed in Sec.B., we obtain a sequence of rigid transformations  $\{F_H[0], F_H[1], F_H[2], \dots, F_H[N_{Frame}]\}$ . Finally, using the pivot calibration package developed in Sec.C., we get the pivot vector relative to the EM tracker system.

---

**Algorithm 4:** Optical prob calibration algorithm

---

- Input:** coordinates of markers on the tool relative to optical tracker system in different frames  $\{\{H_i[0]\}, \{H_i[1]\}, \dots, \{H_i[k]\}, \dots\}$ , coordinates of markers on the EM tracker base relative to optical tracker system in different frames  $\{\{\vec{D}_i[0]\}, \{\vec{D}_i[1]\}, \dots, \{\vec{D}_i[k]\}, \dots\}$ , coordinates of markers on the EM tracker base relative to EM tracker system in different frames  $\{\vec{d}_i\}$ ;
- Output:** dimple position relative to optical tracker system  $\vec{p}_{dim}$ ;
- 1 **Initialization:** number of frames  $N_{Frame}$ ; transformation list  $\mathbb{T} = \emptyset$ ;
  - 2 **Compute**  $F_D$  by Eq. (10)
  - 3 **Compute**  $H\_EM_i$ , the coordinates relative to EM tracker system by Eq. (16)
  - 4 **Compute** local coordinates  $\{h_i[0]\}$  by Eq. (18) and define the local coordinate system attached on pointer body
  - 5 **for**  $k = 1, \dots, N_{Frame}$  **do**
  - 6     **Compute**  $F_H[k]$  by registering  $\{h_i[0]\}$  to  $\{H_i[k]\}$
  - 7     Push  $F_H[k]$  at the back of the transformation list  $\mathbb{T}$
  - 8 **end**
  - 9 **Compute**  $\vec{p}_{dim}$  by using Alg. 2 with the transformation list  $\mathbb{T}$  as input
-



Table 1: Quantitative evaluation of our implemented modules. We compare the outputs of the implemented methods by calculating the average L2-Norm with the given ground-truth.

Case	L2-Norm		
	$\vec{C}_i$	$\vec{p}_{dim}(\text{EM})$	$\vec{p}_{dim}(\text{Opt})$
a	0.0048	0.0037	0.0075
b	0.4711	0.0044	0.0055
c	0.3453	0.5551	0.0065
d	0.0101	0.0068	0.0051
e	1.5046	0.3103	0.0099
f	1.6031	0.5462	0.0062
g	1.5621	0.2854	0.0075
Average	0.7859	0.2446	0.0069

Table 2: Dimple coordinates of unknown cases with respect to EM tracker system and optical tracker system.

case	$\vec{p}_{dim}(\text{EM})$	$\vec{p}_{dim}(\text{Opt})$
h	(210.7992, 195.5491, 218.6504)	(394.5904, 399.9720, 192.8282)
i	(206.1700, 201.6335, 193.2549)	(404.0700, 398.2292, 203.9080)
j	(191.2612, 190.1500, 210.1792)	(397.6587, 408.1820, 202.7904)
k	(191.1497, 201.2274, 187.1436)	(402.1888, 403.1132, 197.8957)

## 2. Algorithm and implementation

The step by step procedure is shown in Alg. 4.

# Results and discussion

## 1. Accuracy evaluation method

In this assignment, we adopt L2-Norm to evaluate the implemented programs. 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 L2-Norm for all the output position vectors in each case.

## 2. Discussion about the results

According to Table 1, it is obvious that the calibration result attained by optical tracker  $\vec{p}_{dim}(\text{Opt})$ , shows a higher accuracy than its EM counterpart  $\vec{p}_{dim}(\text{EM})$ .

During the 3D-3D point sets registration, since we utilize the SVD method depending on some prerequisites mentioned in Alg. 1, there could be some observations being discarded and failing to provide valid information to the registration procedure, which may impair the performance of registration.

## Statement of teamwork

Zijian Wu developed the A, C, F parts, and Shuojue Yang developed the B, D, E parts of this assignment.

## References

- [1] K. S. Arun, T. S. Huang, and S. D. Blostein, “Least-squares fitting of two 3-d point sets,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-9, no. 5, pp. 698–700, 1987.