PA2-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 Chongzhen Yang.

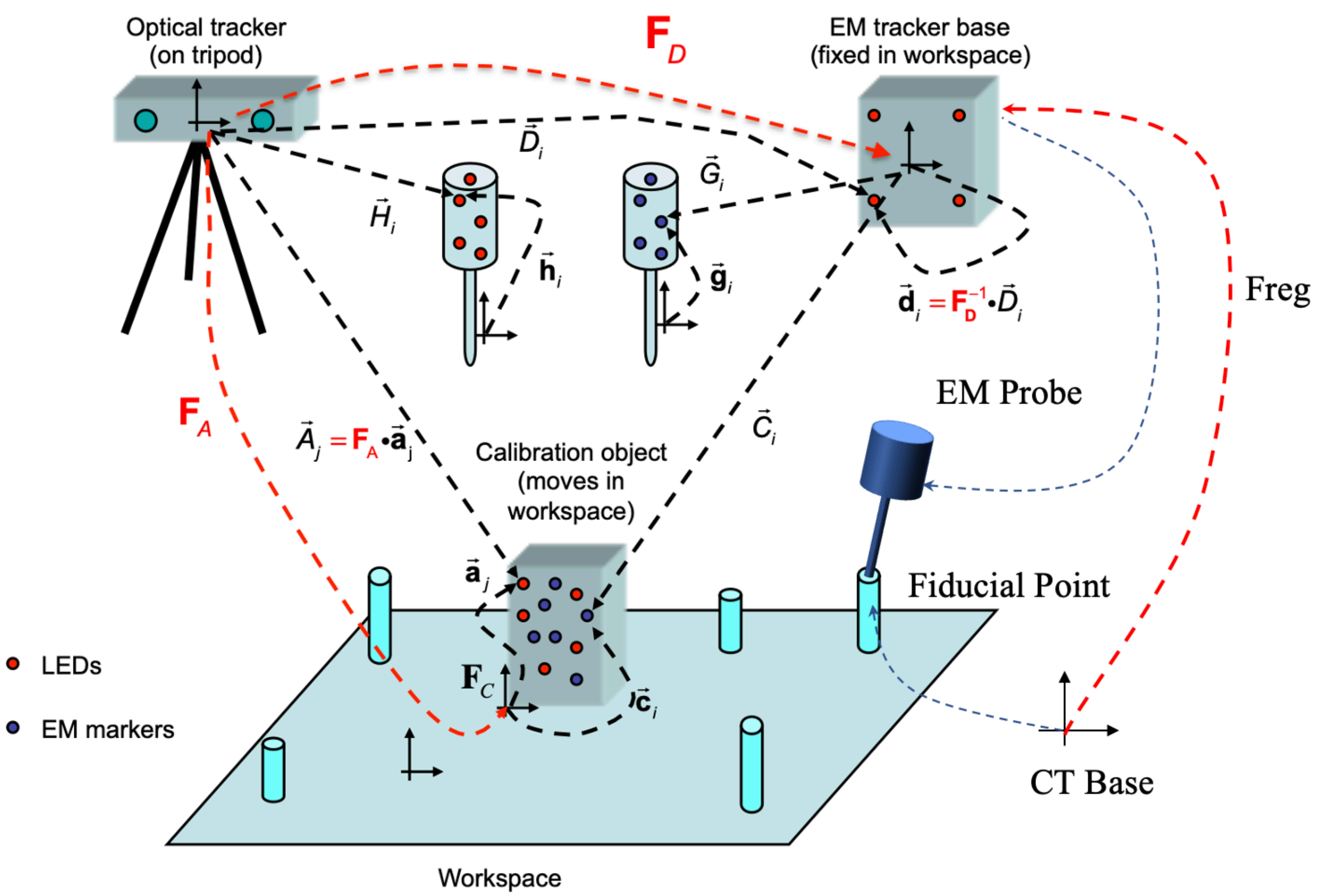
Grade Factor		
Program (40)		
Design and overall program structure	20	
Reusability and modularity	10	
Clarity of documentation and programming	10	
Results (20)		
Correctness and completeness	20	
Report (40)		
Description of formulation and algorithmic approach	15	
Overview of program	10	
Discussion of 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 2.

Scenario

The goal is to correct the distortion of the electromagnetic tracker and apply the corrected data to locate the fiducial markers in the CT coordinate system.



0. Registration and Calibration Recap

In Programming Assignment 1, we developped a math package to describe frame transformation in Cartesian Space. We also implemented point cloud to point cloud registration and pivot calibration methods. The mathematical methods and code implementation are explained and described in Ref 1. Here we'll go through a quick recap for what we included in PA1.

1) Cartesian Math Package

Rigid body transformations can be represented by matrix multiplcations. Suppose there are two Cartesian coordinate frames, A and B. Position vectors 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 $F \cdot p$. A function that computes the homogeneous form of the given transformation frame of position vector is provided in the **"../cispa/CarteFrame.py"** .

2) Registration

Our implementation uses a non-iterative least-squares approach to match two sets of 3D points. Suppose we have two point sets and denote them as $P = \{p_i, i \in 1, 2, \ldots, N\}$ and $P' = \{p'_i, i \in 1, 2, \ldots, N\}$, and they are representing the same rigid body described at different poses. Hence, they should follows: $p'_i \approx Rp_i + t$, in which the error is stated as: $ei = p'_i - Rp_i - t$. We applied Singular Value Decomposition to find the R and t that minimize the following cost function:

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

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 as $p_{ic} = p_i - \bar{p}$, $p'_{ic} = p'_i - \bar{p}'$. Therefore, the problem reduces to:

$$\hat{R} = \underset{R}{\text{argmax}} \sum ||p'_{ic} - Rp_{ic}||^2 \tag{3}$$

And t can be found simply by:

$$\hat{t} = \bar{p}'_i - \hat{R}\bar{p}_i \tag{4}$$

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.

3) Pivot Calibration

The model of pivot calibration is described as: $\vec{p}_{pivot} = F_i \vec{p}_t = R_i \vec{p}_t + \vec{p}_i$
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 (5).

$$\begin{bmatrix} \cdots & \cdots \\ R_i & -I \\ \cdots & \cdots \end{bmatrix} \begin{bmatrix} p_t \\ p_{pivot} \end{bmatrix} = \begin{bmatrix} \cdots \\ -p_i \\ \cdots \end{bmatrix} \tag{5}$$

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

2. Compute $C^{expected}$

We compute the expected value of calibration object poistion w.r.t. the EM tracker coordinate system at each frame k in this section.

1) Mathematical Method

As described in the Ref 1, $\vec{C}^{expected}$ is computed based on the Optical tracker data and arguments of the rigid body (i.e. calibration object). The Optical tracker readings is recorded at each frame k. Here the frame k denotes the position of the calibration object. Therefore, the steps taken can be stated as:

First, we compute the transformation for EM tracker base and calibration object w.r.t. the Optical tracker.

$$\begin{aligned} \vec{D}_i[k] &= F_D[k]\vec{d}_i \\ \vec{A}_i[k] &= F_A[k]\vec{a}_i \end{aligned} \tag{6}$$

And then, through registration, F_D and F_A can be found, thereby the $\vec{C}^{expected}$ can be computed by:

$$\vec{C}_i^{expected}[k] = F_D^{-1}[k] F_A[k] \vec{c}_i \tag{7}$$

Here, $\vec{a}_i$, $\vec{b}_i$ and $\vec{c}_i$ are the position of markers w.r.t. the calibration object reference frame. They are constant values since the calibration object is a rigid body.

2) Code Implementation

The code is implemented in **"../cispa/ComputeExpectValue.py"** and the function is **C_expected()**. The steps of our algorithm is shown as follows:

Algorithm 1 ComputeExpectValue.py

INPUT: Point cloud $\{\vec{A}_i\ \vec{D}_i\}$ Optical marker position w.r.t. Optical tracker

Point cloud $\{\vec{a}_i\ \vec{d}_i\ \vec{c}_i\}$ Optical & EM marker position w.r.t. the calibration object

OUTPUT: Expected position of EM marker w.r.t. the EM tracker

FOR: each frame of data in $\{\vec{A}_i\ \vec{D}_i\}$

  CALL: $F_D[k] = \text{Registration}(D[k], d)$, $F_A[k] = \text{Registration}(A[k], a)$

  FOR: each point in set $\{c\}$

    COMPUTE: $\vec{c}_i^{\,expected}[k] = F_D^{-1}[k] F_A[k] \vec{c}_i$

  ENDFOR

ENDFOR

RETURN: $C^{expected} = \{\vec{c}_i^{\,expected}[k]\}$

3. Distortion Correction

We developped an approach for correcting distorted data. The distortion is invoked by the EM tracker and is manifested as a slight change in the relative position between markers on the calibration object.

1) Mathematical Method

The distortion is corrected by applying Bernstein Polynomials to fit the given data. For a scalar u Bernstein Polynomials are defined by the Bernstein basis:

$$B_{N,k}(u) = \binom{N}{k} (1-u)^{N-k} u^k \tag{8}$$

Where N denotes the highest order of the Bernstein basis. For 3D case, the Berstein basis will become the production of 3 independent basis of scalar.

$$B_{N,ijk}(\vec{p}) = [\binom{N}{i}(1-p_x)^{N-i}p_x^i] * [\binom{N}{j}(1-p_y)^{N-j}p_y^j] * [\binom{N}{k}(1-p_z)^{N-k}p_z^k] \tag{9}$$

And thereby the Bernstein Polynomials can be established as a linear combination of the basis in Eq 9. For any distorted point $\vec{p} = [p_x, p_y, p_z]$ in 3D space, if the order of Bernstein Polynomials N is known, we can have the polynomials $\vec{F}(\vec{p})$ arranged into a vector form:

$$\vec{F}(\vec{p}) = \begin{bmatrix} F_{000}(\vec{p}) & F_{001}(\vec{p}) & \dots & F_{ijk}(\vec{p}) & \dots & F_{NNN}(\vec{p}) \end{bmatrix} \tag{10}$$

It's easy to show that $\vec{F}(\vec{p})$ is a $1 \times (N+1)^3$ row vector, where $i, j, k \in \{0, 1, 2 \dots N\}$ and together with the "ground truth" $\vec{p}_g$ they follows:

$$\vec{F}(\vec{p}) \begin{bmatrix} c^x_{000} & c^y_{000} & c^z_{000} \\ c^x_{001} & c^y_{001} & c^z_{001} \\ \dots & \dots & \dots \\ c^x_{ijk} & c^y_{ijk} & c^z_{ijk} \\ \dots & \dots & \dots \\ c^x_{NNN} & c^y_{NNN} & c^z_{NNN} \end{bmatrix}_{(N+1)\times 3} = \begin{bmatrix} p^x_g & p^y_g & p^z_g \end{bmatrix} \tag{11}$$

Obviously, the position vectors need to be scaled to a bounding box with the range of $[0, 1]$ to maintain numerical stability of the Berntein basis. That is, $\forall \vec{p}$ in distorted dataset or the ground truth dataset, $\vec{p}_s = ScaleToBox(\vec{p})$. Note that here we consider the data to be anisotropy, which means we independently scale the data on every dimension instead of simply normalize the data. As we can see, Eq 11 is a underdetermined equation. If we have multiple points to be corrected, Eq 11 can become a least square problem, i.e.

$$\begin{bmatrix} \vec{F}(\vec{p}_1) \\ \vec{F}(\vec{p}_2) \\ \vec{F}(\vec{p}_3) \\ \dots \end{bmatrix} \begin{bmatrix} \dots & \dots & \dots \\ c^x_{ijk} & c^y_{ijk} & c^z_{ijk} \\ \dots & \dots & \dots \end{bmatrix} = \begin{bmatrix} p^x_{1g} & p^y_{1g} & p^z_{1g} \\ p^x_{2g} & p^y_{2g} & p^z_{2g} \\ p^x_{3g} & p^y_{3g} & p^z_{3g} \\ \dots & \dots & \dots \end{bmatrix} \tag{12}$$

The "distortion correction coefficient" can be easily found by solving the least square problem described in Eq 12. And we can rectify any point set that follows the same distortion pattern by the distortion correction coefficient afterwards.

2) Code Implementation

The code contains four separate modules, namely **Scale2Box(p)**, **Bernstein(p, order)**, **fit(p,p_g)** and **predict(p, coeff)**. These functions are implemeted in **"../cispa/CorrectDistortion.py"**. The steps taken are described as the following pseudocode.

Function1	**Bernstein(p, ORDER)**
INPUT:	A Point Cloud P
OUTPUT:	Corresponding bernstein polynomial matrix $F(P)$
DEFINE:	bern_basis(i,u) = Combination$(ORDER, i)(1-u)^{N-i}u^i$
	FOR each point $\vec{p} \in P$
	FOR each order i,j,k in ORDER
	CALL: $F_{ijk}(\vec{p})$ = bern_basis(i, p_x)bern_basis(j, p_y)bern_basis(k, p_z)
	$F(\vec{p}) = [F(\vec{p})\ ,\ F_{ijk}(\vec{p})\ ]$
	ENDFOR
	$F(P) = \begin{bmatrix} F(P) \\ F(\vec{p}) \end{bmatrix}$
	ENDFOR

As we discussed in the mathematics part, the data need to be scaled to the range of [0,1].

Function2	**Scale2Box**
INPUT:	Original Point Cloud P
OUTPUT:	Scaled Point Cloud P^{scaled}
STEP1:	Find the minimum and maximum value on every dimension
STEP2:	$x_i^{scaled} = \frac{x_i - x_{min}}{x_{max} - x_{min}}$, $y_i^{scaled} = \frac{y_i - y_{min}}{y_{max} - y_{min}}$, $z_i^{scaled} = \frac{z_i - z_{min}}{z_{max} - z_{min}}$
RETURN:	$\vec{p}^{scaled} = \{\ldots, [x_i^{scaled} y_i^{scaled} z_i^{scaled}], \ldots\}$

Now we need to prepare a distorted data and its corresponding standard data to "train the model" through fit function.

Function3	**fit**
INPUT:	Distorted Point Cloud P and Expected Point Cloud $P^{expected}$
OUTPUT:	Correction coefficient matrix C
STEP1:	Scale both point cloud to box $\vec{p}_s = Scale2Box(\vec{p})$ and $\vec{p}_s^{expected} = Scale2Box(\vec{p}^{expected})$
STEP2:	Find Berstein Polynomial Matrix $F(\vec{p})$ = Bernstein$(p_s$, order=5)
STEP3:	Solve $C = leastsquare(F(\vec{p}), \vec{p}_s^{expected})$
RERURN:	C

Note that here we always need to scale both the distorted data and the expected/corrected data. After we find the C through fitting process, we can rectify or, in other words, predict the distortion for other point cloud.

Function4	Predict
INPUT:	Point Cloud $\vec{p}$ that needs to dewlap, Correction coefficient matrix C
OUTPUT:	Corrected Point Cloud $\vec{p}^{corrected}$
STEP1:	Scale the input point cloud to box $\vec{p}_s = Scale2Box(\vec{p})$
STEP2:	$\vec{p}_s^{corrected} = Bernstein(\vec{p}_s) \cdot C$
STEP3:	Scale back $\vec{p}_s^{corrected}$ according to the max and min value of input point cloud
RETURN:	$\vec{p}^{corrected}$

4. Pivot Calibration

1) Mathematical Method

The EM pivot calibaration procesure is the same as the method in Programming Assignment 1. The main method is shown recap section. The only difference here is we need to correct the distortion of the EM marker point cloud before performing the pivot calibration.

We choose the first frame as the start frame and then establish a local coordinate system attached on the tool. We define the corrected local and EM system coordinates at the k-th frame as $g^{corrected}[k]$ and $G^{corrected}[k]$, respectively. Since the local reference frame is rigidly connected to the calibration probe. $g^{corrected}[k]$ remains the same when k varies. Hence $g^{corrected}[0]$ will be taken as the local reference frame to perform pivot calibration. And thereby the transformation between different poses of the probe can be found by solving the registration problem:

$$\vec{G}_j^{corrected}[k] = F_G[k] \cdot \vec{g}_j^{corrected}[0] \tag{13}$$

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

2) Code Implementation

The code is implemented in **"../PA2/pa2_problem3_test.py"**.

Algorithm 1	**Dewraped EM probe calibration**
INPUT:	"../*-empivot.txt" , Distortion Correction Coefficient Matrix C
OUTPUT:	Calibrated post position p_{dimple} relative to EM tracker frame
INITIALIZE:	Empty sequence F
STEP1:	Perform distortion correction on $G_j[k]$ and get $G_j^{corrected}[k]$
STEP2:	Compute the local coordinate $g_j^{corrected}[0]$
STEP3:	For $k \in (0, N_{frames})$ perform registration for $F_G[k] = registration(g_j^c[0], G_j^c[k])$
STEP4:	Push back $F_G[k]$ to the end of F
STEP5:	Perform pivot calibration for $\{\vec{p}_{dimple}, t_G\} = calibration(F)$
RETURN:	$\vec{p}_{dimple}$

5. Find Fiducials w.r.t. EM Coordinate System

To find the fiducial positions described in the EM coordinate system, we need to derive the corresponding $\vec{p}_{dimple}$. The basic steps are: use the probe to touch the fiducial marker and collect a series of point cloud relative to the EM tracker, and then move the probe to another fiducial and repeat the data collection process.

1) Mathematical Methods

The first step we need to take is to correct the data collected by the em tracker. Then we perform pivot calibration to find the tip position w.r.t. the probe local frame $\vec{p}_{tip}$. By registration between the point clouds collected at different fiducials, we can calculate F_T , which is the rigid transformation between probe local frame and EM tracker frame. Finally, the pivot position of the probe w.r.t. the EM tracker can be calculated through:

$$\vec{p}_{pivot} = F_T \vec{p}_{tip} \tag{14}$$

And since the probe pricks at the fiducial marker, the positions of the fiducials are therefore equals to the pivot position of the probe. Hence, we have:

$$\vec{B}_i = \vec{p}_i^{pivot} \tag{15}$$

2) Code Implementation

The code is implemented in **"../PA2/pa2_problem4_test.py"**. This program first perform the exact same process as problem3, i.e. perform distortion corrected pivot calibration to find the tip position in the probe local frame. And then perform point cloud registration between k-th frame of EM tracker readings and the "centralized" point cloud used in problem3. Eventually, the fiducial positions w.r.t. the EM tracker frame are found through Eq 15. The pseudocode can be found at Section 7 Step 1-3.

6. Find Transformation EM-CT (F_{reg})

The transformation between two coordinate systems can be derived through locating the same point cloud in these two coordinate systems.

1) Mathematical Method

Through the prevoius section, we locate the fiducials in the EM coordinate system, denote as $\vec{B}_i$ and the fiducials in the CT coordinate system are recorded as $\vec{b}_i$. Therefore they can be connected through the following equation:

$$\vec{b}_i = F_{reg}\vec{B}_i \tag{16}$$

2) Code Implementation

The code is implemented in **"../PA2/pa2_problem5_test.py"**. Note the EM-related data must be dewraped in the beginning. This program first perform the exact same process as problem4. Thus the fiducials are located in the EM tracker coordinate system, and the same fiducial positions can be read directly from the CT coordinate system. By performing point cloud to point cloud registration, the transformation F_{reg} will be found. The pseudocode can be found at Section 7 Step 1-4.

7. Locate probe tip w.r.t. CT Coordinate System

The probe tip position is directly read from EM tracker data. With the known frame transformation from EM tracker frame to the CT frame F_{reg}, we can translate the probe tip position into the CT coordinate system.

1) Mathematical Method

Since there are EM markers sticked on the probe, we can always get the position of the probe tip w.r.t. EM tracker by F_G , which is derived from registration between the current point cloud and the centralized reference point cloud, and the tip position in probe local frame $\vec{p}_{tip}$, which is determined in the pivot calibration process.

$$\vec{B}_i = \vec{p}^{EM}_{pivot} = F_G \vec{p}_{tip} \tag{17}$$

Through the previous steps, the frame transformation matrix F_{reg} and the tip position in probe local frame $\vec{p}_{tip}$ have been determined. Therefore, the tip position described in CT frame is stated as:

$$\vec{b}_i = F_{reg} B_i = F_{reg} F_G \vec{p}_{tip} \tag{18}$$

2) Code Implementation

The code is implemented in **"../PA2/pa2_problem6_test.py"**. Note the EM-related data must be dewraped in the beginning.

Algorithm 2	**Locate the probe tip in CT frame**
INPUT:	"/..-empivot.txt" "/..-em-NAV.txt" "/..-em-fiducials.txt" "/..-ct-fiducials.txt"
OUTPUT:	Probe tip location $\vec{p}_{CT}$w.r.t. CT coordinate system
STEP1:	Perform distortion correction on em related data
STEP2:	Perform pivot calibration to find the probe tip position relative to its local frame
STEP3:	Perform point cloud registration and pivot calibration to find the fiducials w.r.t. the EM coordinate system
STEP4:	Perform point cloud registration to find the frame transformation between EM frame and CT frame
STEP5:	Compute the probe tip position w.r.t EM tracker frame and transfer it to CT frame
RETURN:	probe tip position w.r.t. CT coordinate system $\vec{b}_i$

III. Unit Test and Debug

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. Distortion Correction Unit Test

In the ../PROGRAMS directory, run test script "/PA2/pa2_problem2_test.py". This script is desinged to test the distortion correction functions, in the terminal, run the following command:

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