PA4-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 ChongZun Yang.

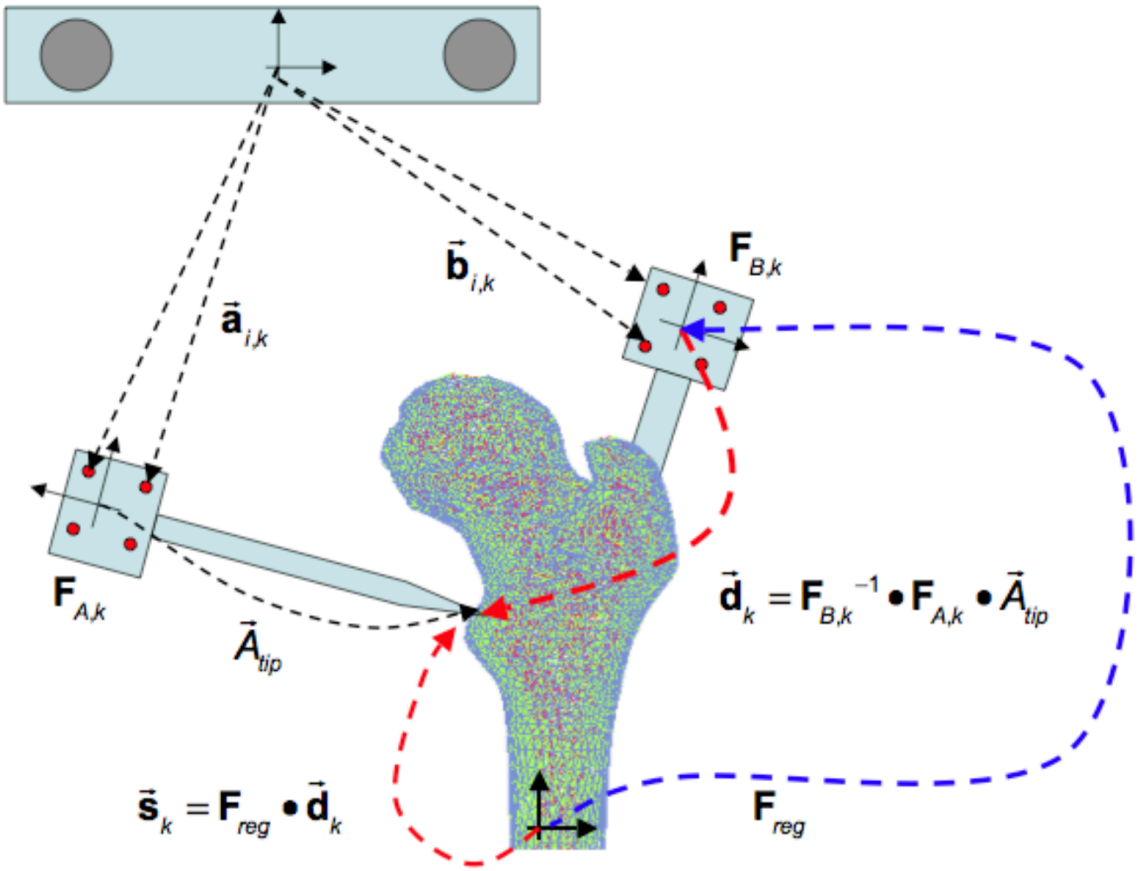
Grade Factor		
Program (40)		
Design and overall program structure	20	
Reusability and modularity	10	
Clarity of documentation and programming	10	
Results (20)		
Correctness and completeness	20	
Report (40)		
Description of formulation and algorithmic approach	15	
Overview of program	10	
Discussion of validation approach	5	
Discussion of results	10	
TOTAL	100	

I. Mathematics & Algorithms Implementation

This section introduces the mathematical principles and implemented algorithm for locating the closest point to a given mesh file described in Programming Assignment 4.

0. Scenario

In this programming assignment, we are required to compute the registration transformation F_{reg} using Iterative Closest Point (ICP). We use LED trackers to detect 2 rigid bodies on the bone and obtain the point cloud from intraoperative reality. The CT system could obtain information of the 3D surface model. By finding the transformation matrix between the preoperative surface mesh and intraoperative point cloud we can derive the frame transformation between the CT frame and Bone frame. While finding the closest triangle in the ICP algorithm, we implemented Brute-Force Search and Octree Search to determine the closest points. We also compared their performance in terms of searching efficiency.



1. Iterative Closest Point Method

1) Mathematical Method

1. Find Corresponding Point Pairs

   Since we are finding the transformation between a point cloud and a surface mesh, the algorithm needs to identify the closest pair between the points cloud and surface mesh in the first place.

   Then we use the search method to search for the closest point on the surface mesh for each element in the point cloud, i.e., find the closest points on the surface corresponding to $F_{reg}q_k$

   As introduced in PA3, we implemented Brute Force Searching and Octree Searching methods to solve this problem. Here we use F_n to denote the F_{reg} updated in the previous iteration, $\vec{q}_k$ is the k-th point in the point cloud and $\vec{c}_k$ is the corresponding closest needlepoint on the surface mesh in the previous corresponding process.

   Note that we assume that $Freg$ is identity matrix as the initial guess.

2. Find Best Transformation

   After we found the updated closest corresponding point pair. We can utilize the registration function that we implemented in the PA1 to find the transformation that minimizes the total errors between the given point cloud and the closest cloud as Eq 1 shows. After updating the registration transformation, we calculate residuals for each transformed point with the corresponding closest needlepoint. Then we can compute the residual error based on the values shown as Eq 2-5: And update the termination conditions according to the following functions:

$$\hat{R} = \underset{R}{\operatorname{argmax}} \sum ||p'_{ic} - Rp_{ic}||^2 \tag{1}$$

$$\vec{e}_k = F_{reg}\vec{q}_k - \vec{c}_k \tag{2}$$

$$\sigma_n = \frac{\sqrt{\Sigma \vec{e}_k \cdot \vec{e}_k}}{NumElts(A)} \tag{3}$$

$$\epsilon_{max} = \max_k \sqrt{\vec{e}_k \cdot \vec{e}_k} \tag{4}$$

$$\bar{\epsilon}_n = \frac{\Sigma \sqrt{\vec{e}_k \cdot \vec{e}_k}}{NumElts(A)} \tag{5}$$

3. Adjusting the Boundary for Outliers

   As the iteration proceeds, we can leverage the distance values of each point pair and their residual errors serving as thresholds to filter out some clearly wrong matches.

4. Check Termination Condition

   Check for the error metrics. If the average and variance of the errors are small enough, and the max errors are also small enough, stop the iteration and return the current F_{reg}. If the termination condition isn't satisfied, go to step 1 and revisit the whole process.

   Remember, in the next iteration, using $F_{reg}q_k$ to find the closest points c_k; using q_k and c_k to update F_{reg}

2) Code Implementation

The code is implemented in **"/cispa/IterClosestPoint.py"**. The basic steps are:

Algorithm 1	**Compute F_{reg}**
Input:	Point Cloud Q and Surface Mesh M
Output:	F_{reg}: The frame transformation between Q and M
Initialize:	$F_{reg} = I_{4\times4}$; n = 0
STEP 1:	A = $\emptyset$; B = $\emptyset$
	FOR each point q_k in Q
	CALL [c_k, d_k] = **CorrespondPoints**(q_k , M , search_method)
	ENDFOR
	IF $d_k < \eta_n$
	A.pushback(q_k) ; B.pushback(c_k)
STEP 2:	n += 1 #**Comment**: Iteration Count
	CALL F_{reg} = **Registration**(A , B)
	CALL Update Termination Conditions
STEP 3:	**COMPUTE** η_n
STEP 4:	**IF** Termination Condition == False
	GO TO STEP 1
	ELSE
	RETURN F_{reg}

The **CorrespondPoints** method is done by calling find_closest_point function developed in previous PA. The code is implemented in **"/cispa/IterClosestPoint.py"**. The basic steps are:

Algorithm 2	**Find Corresponding Point Pairs**
Input:	Source Point q_k , Surface Mesh M , Search Method = {'Brute', 'Octree'}
Output:	Closest point , Minimum Distance
	IF Search Method == 'Brute'
	Closest point , Minimum Distance = BruteForceSolver(q_k)
	ElSE IF Search Method == 'Octree'
	Closest point, Minimum Distance = OctreeSolver(q_k)

With the matched point pairs, we adopted the 3D point-cloud-to-point-cloud registration method introduced in PA1 and PA2 to compute the best transformation.

There might be some outlier points in the source point cloud, so we need a threshold η_n to filter out the wrong pairs. Here we adopt $3\varepsilon_n$ as this threshold. In practice, the setting of η_n is very tricky. On the one hand, a strict threshold might delete some matching pairs that are actually correct. On the other hand, a loose threshold might not filter out the wrong pairs.

The criterion of termination we adopted is to stop the iteration when σ_n , ε_n, $(\varepsilon_m ax)_n$ are less than desired thresholds (all take 0.01) and $\gamma \leq \varepsilon_n/\varepsilon_{n-1} \leq 1$. Here we assume that $\gamma = 0.95$.

II. Overall Structure

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

```
└── PROGRAMS
    ├── PA4              # 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
    |   ├── pa4_main.py  # This DIR contains all the result of our program
    |   └── pa4_icp_test.py  # Test the ICP Solver
    └── cispa                          # Utility Functions are in this directory
├── CarteFrame.py
    ├── ComputeExpectValue.py
    ├── CorrectDistortion.py
    ├── DataProcess.py
    ├── FindBoundingSphere.py
    ├── FindClosestPoint2Mesh.py       # Find the closest point on the mesh to point
    ├── FindClosestPoint2Triangle.py
    ├── HomoRepresentation.py
    ├── Octree.py
    ├── PivotCalibration.py
    ├── IterClosestPoint.py            # ICP Implementation

    └── Registration.py
```

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
PA4   cispa
```

Dear Graders, Before starting to run our code, please remember to install **matplotlib** in your virtual environment.

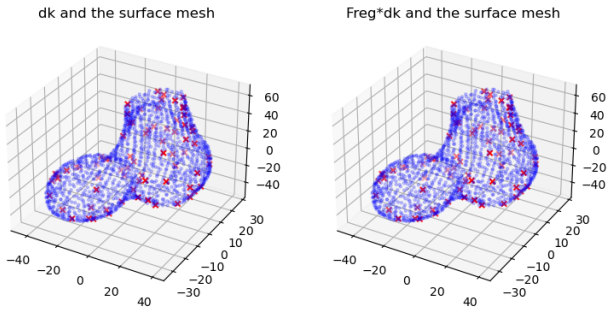
1. ICP Unit Test

This script is implemented to run and test the Iterative Closest Point Algorithm. In the ../PROGRAMS directory, run test script "/PA4/pa4_icp_test.py". In the terminal, using the following command to test the ICP with given debug data based on BruteForce Search.

```
$ python3 /PA4/pa4_icp_test.py -n PA4-A-Debug
------------------------------
Iteration:  0
Max Error:  0.007296588755025472
Sigma:  6.418087247959794e-05
Eps_bar:  0.0026738785954926376
Convergence Flag:  False
Termination Condition Reached!
Checking Convergence...
------------------------------
Iteration:  1
Max Error:  0.007324601783688765
Sigma:  5.82115039929643e-05
Eps_bar:  0.002302747183746499
Convergence Flag:  False
Termination Condition Reached!
Checking Convergence...
------------------------------
Iteration:  2
Max Error:  0.006206855125657278
Sigma:  5.374036809133574e-05
Eps_bar:  0.0020148950330007493
Convergence Flag:  False
Termination Condition Reached!
Checking Convergence...
------------------------------
Iteration:  3
Max Error:  0.004451084757710672
```

```
Sigma:  5.047461309804469e-05
Eps_bar:  0.0018138042593038159
Convergence Flag:  False
Termination Condition Reached!
Checking Convergence...
----------------------------
Iteration:  4
Max Error:  0.004312592212429139
Sigma:  4.760076615548475e-05
Eps_bar:  0.001734653467734899
Convergence Flag:  True
Termination Condition Reached!
Checking Convergence...
Converged!
[05:24:39] INFO     ICP time =
                    39.031960010528564 seconds
           INFO     ICP transformation matrix :
                    R =
                    [[ 1.00000000e+00 -1.10774285e-05  1.77951869e-05]
                     [ 1.10786612e-05  9.99999998e-01 -6.92754954e-05]
                     [-1.77944194e-05  6.92756925e-05  9.99999997e-01]]


                     t =[[-0.00200915] [-0.00194022] [-0.00070645]]
```



Compare the result to given answer, we can see our code is correct.

2. Discussion for the results for different cases

1) Debug case

In Table 1, compared with the Brute Force Search, the Octree Search has largely reduced running time. Besides the octree generation time consumption, the search process only takes 1/8 of the time for the brute force search. This could be caused by the performance in the implementation of python. During the tree construction process, there would be a series of calling and assignment operations for List type limiting the efficiency of Octree.

Table 1: Comparison of various methods in running time for Debug case A-F

Method	**Time(s)**					
	A	B	C	D	E	F
Brutal Force Solver	39.0319	720.4946	700.1229	745.9486	419.5355	362.8294
Octree Solver	5.0123	79.18321	82.04423	83.2465	47.2459	39.3456

We notice that E and F data cannot reach the threshold we set for max-error and average error. Therefore, we exit the iteration according to the convergence criterion. A reasonable hypothesis is that there are noises in datasets E and F which cannot be eliminated through the outlier reduction process.

2) Unknown case

For Unknown cases, we save the results in ./output. In the section, we also give a time comparison, which is shown in Table 2. For all 4 cases, the Octree-based Search method performs better than the Brute Force Method.

Table 2: Comparison of various methods in running time for Debug case G-K

Method	**Time(s)**			
	G	H	J	K
Brute Force Solver	682.0382	847.2487	410.4638	249.9082
Octree Solver	72.3959	89.0580	33.7900	29.0014

Similarly, We notice that J and K data cannot reach the threshold we set for max-error and average error. We presume it has the same reason as Debug datasets E and F.

IV. Result and Discussions

1. Evaluation Metrics

The evaluation methods are introduced in Eq 2-5. Besides, we consider the convergence condition to be another termination condition. That is because there exist some cases that the source data is "polluted" by noises so that max_error cannot reach to termination threshold even if the conditions have already converged.

2. ICP Result for Debug Dataset

In this section, we first give a quantitative evaluation of the implemented ICP algorithm on various cases shown in Table 1. In addition, for both the Debug and Unknown cases, we compared the running time of the three made searching methods, which are Brute-Force Search, and Octree Search, to demonstrate the efficiency improvement.

To demonstrate our result, we use L2-Norm to compare the given output and ours, the comparison is shown in Table 3

TABLE 3 L2-Norm for our result and given output

Dataset	s_k error	c_k error
A	0.0072	0.0065
B	0.0088	0.0079
C	0.0091	0.0080
D	0.0112	0.0099
E	0.0298	0.0231
F	0.0303	0.0317

2.1. A-Debug case

The result is compared with the "PA4-A-Debug-own-output.txt" and a segment of the result is shown in the table below.

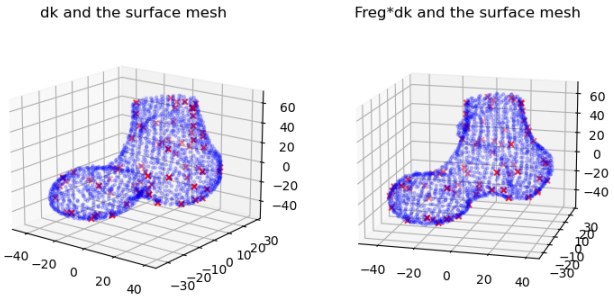
TABLE 4 PA3-A-Debug-own-output

Our Results

$d_k.x$	$d_k.y$	$d_k.z$	$c_k.x$	$c_k.y$	$c_k.z$	$\|\vec{d}_k - \vec{c}_k\|$
-4.76523	20.38775	13.36044	-4.76683	20.38423	13.36185	0.00411
-6.13158	17.36735	12.29557	-6.13419	17.36472	12.29568	0.00370
-0.25092	4.91662	-21.87844	-0.2546	4.91456	-21.87736	0.00443

Given Results

$d_k.x$	$d_k.y$	$d_k.z$	$c_k.x$	$c_k.y$	$c_k.z$	$\|\vec{d}_k - \vec{c}_k\|$
-4.77	20.38	13.36	-4.77	20.38	13.36	0.00
-6.13	17.36	12.30	-6.13	17.36	12.30	0.00
-0.25	4.91	-21.88	-0.25	4.91	-21.88	0.00



2.2. B-Debug

The result is compared with the "PA4-B-Debug-own-output.txt" and a segment of the result is shown in the table below.

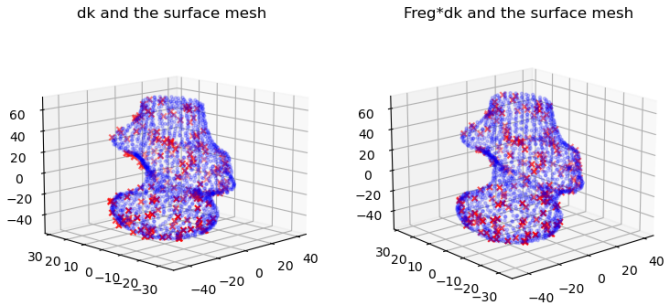
TABLE 5 PA4-B-Debug-own-output

Our Results

$d_k.x$	$d_k.y$	$d_k.z$	$c_k.x$	$c_k.y$	$c_k.z$	$\|\vec{d}_k - \vec{c}_k\|$
6.67632	-4.78724	63.41865	6.68189	-4.80027	63.41594	0.00050
-37.19274	-17.17340	-41.06712	-37.19187	-17.17339	-41.06642	0.00121
28.11550	19.27105	12.82410	28.11558	19.27111	12.82416	0.00012

Given Results

$d_k.x$	$d_k.y$	$d_k.z$	$c_k.x$	$c_k.y$	$c_k.z$	$\|\vec{d}_k - \vec{c}_k\|$
6.68	-4.80	63.42	6.68	-4.80	63.42	0.00
-37.20	-17.16	-41.07	-37.20	-17.16	-41.07	0.00
28.12	19.26	12.82	28.12	19.26	12.82	0.00



2.3. C-Debug

The result is compared with the "PA4-C-Debug-own-output.txt" and shown in the table below.

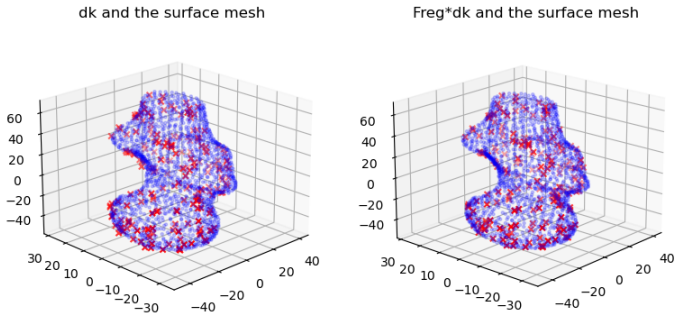
TABLE 6 PA4-C-Debug-own-output

Our Results

$d_k.x$	$d_k.y$	$d_k.z$	$c_k.x$	$c_k.y$	$c_k.z$	$\|\vec{d}_k - \vec{c}_k\|$
3.14217	-7.01464	52.60118	3.14176	-7.01778	52.60090	0.00318
30.04987	12.93950	16.46012	30.04923	12.93932	16.45975	0.00076
-33.86310	-23.65617	-13.48330	-33.86272	-23.65572	-13.48364	0.00068

Given Results

$d_k.x$	$d_k.y$	$d_k.z$	$c_k.x$	$c_k.y$	$c_k.z$	$\|\vec{d}_k - \vec{c}_k\|$
3.15	-7.02	52.60	3.15	-7.02	52.60	0.00
30.05	12.94	16.46	30.05	12.94	16.46	0.00
-33.86	-23.66	-13.48	-33.86	-23.66	-13.48	0.00



2.4. D-Debug

The result is compared with the "PA4-D-Debug-own-output.txt" and shown in the table below.

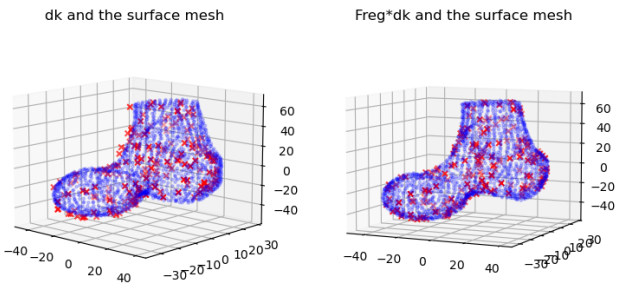
TABLE 7 PA4-D-Debug-own-output

Our Results

$d_k.x$	$d_k.y$	$d_k.z$	$c_k.x$	$c_k.y$	$c_k.z$	$\|\vec{d}_k - \vec{c}_k\|$
4.64148	10.64674	-11.33561	4.64150	10.64569	-11.33496	0.00123
-33.66358	-26.57982	-17.16295	-33.66550	-26.57981	-17.16183	0.00303
3.95660	19.38075	0.62465	3.95714	19.38058	0.62492	0.00063

Given Results

$d_k.x$	$d_k.y$	$d_k.z$	$c_k.x$	$c_k.y$	$c_k.z$	$\|\vec{d}_k - \vec{c}_k\|$
4.64	10.64	-11.34	4.64	10.64	-11.34	0.00
-33.65	-26.59	-17.16	-33.65	-26.59	-17.16	0.00
3.95	19.38	0.62	3.95	19.38	0.62	0.00



2.5. E-Debug

The result is compared with the "PA4-E-Debug-own-output.txt" and shown in the table below.

TABLE 8 PA4-E-Debug-own-output

Our Results

$d_k.x$	$d_k.y$	$d_k.z$	$c_k.x$	$c_k.y$	$c_k.z$	$\|\vec{d}_k - \vec{c}_k\|$
-1.52454	23.96212	21.64005	-1.51920	24.11006	21.63451	0.14814
-16.94467	6.20677	-13.18442	-16.93070	6.16768	-13.22228	0.05618
6.56171	16.61250	-5.98440	6.53039	16.63523	-6.00568	0.04388

Given Results

$d_k.x$	$d_k.y$	$d_k.z$	$c_k.x$	$c_k.y$	$c_k.z$	$\|\vec{d}_k - \vec{c}_k\|$
-1.55	23.95	21.67	-1.54	24.11	21.67	0.158
-16.94	6.21	-13.17	-16.92	6.16	-13.21	0.064
6.56	16.61	-5.95	6.52	16.65	-5.98	0.060

2.6. F-Debug

The result is compared with the "PA4-F-Debug-own-output.txt" and shown in the table below.

TABLE 9 PA4-F-Debug-own-output

Our Results

$d_k.x$	$d_k.y$	$d_k.z$	$c_k.x$	$c_k.y$	$c_k.z$	$\|\vec{d}_k - \vec{c}_k\|$
-3.41479	4.72143	-23.91917	-3.42619	4.70791	-23.9109	0.01952
-16.3325	10.6650	-34.0154	-16.3425	10.6370	-34.0076	0.03068
5.53749	20.4962	43.43669	5.54148	20.4639	43.43246	0.02747

Given Results

$d_k.x$	$d_k.y$	$d_k.z$	$c_k.x$	$c_k.y$	$c_k.z$	$\|\vec{d}_k - \vec{c}_k\|$
-3.42	4.75	-23.90	-3.43	4.73	-23.89	0.028
-16.34	10.67	-34.00	-16.35	10.65	-33.99	0.029
5.50	20.48	43.47	5.50	20.46	43.47	0.022

By inspection, our hypothesis about datasets E and F are noisier has been validated.

3. ICP Result for Unknown Data

3.1 G-Unknown

The result is compared with the "PA4-G-Unknown-own-output.txt" and shown in the table below.

TABLE 10 PA4-G-Unknown-own-output

Our Results

$d_k.x$	$d_k.y$	$d_k.z$	$c_k.x$	$c_k.y$	$c_k.z$	$\|\vec{d}_k - \vec{c}_k\|$
-31.39652	2.95585	-13.76602	-31.39741	2.95613	-13.76382	0.00239
-41.15667	-3.37876	-17.37140	-41.15464	-3.37961	-17.37294	0.00269
-8.87246	-20.46155	-43.13026	-8.87106	-20.46289	-43.14208	0.00265

3.2 H-Unknown

The result is compared with the "PA4-H-Unknown-own-output.txt" and shown in the table below.

TABLE 11 PA4-H-Unknown-own-output

Our Results

$d_k.x$	$d_k.y$	$d_k.z$	$c_k.x$	$c_k.y$	$c_k.z$	$\|\vec{d}_k - \vec{c}_k\|$
-43.06506	-6.11796	-30.09367	-43.06355	-6.11797	-30.09320	0.00158
5.44425	23.97245	20.27309	5.44446	23.97840	20.27387	0.00590
-37.60270	-14.73096	-11.31146	-37.60239	-14.73089	-11.31167	0.00038

3.3 J-Unknown

The result is compared with the "PA4-J-Unknown-own-output.txt" and shown in the table below.

TABLE 12 PA4-J-Unknown-own-output

Our Results

$d_k.x$	$d_k.y$	$d_k.z$	$c_k.x$	$c_k.y$	$c_k.z$	$\|\vec{d}_k - \vec{c}_k\|$
3.09453	23.84074	11.16172	3.14715	23.76177	11.18160	0.09695
8.93881	20.81370	52.50778	8.87499	20.99284	52.53015	0.19758
-1.34118	-11.0236	11.4822	-1.3285	-10.9256	11.47266	0.09924

3.4 K-Unknown

The result is compared with the "PA4-K-Unknown-own-output.txt" and shown in the table below.

TABLE 13 PA4-K-Unknown-own-output

Our Results

$d_k.x$	$d_k.y$	$d_k.z$	$c_k.x$	$c_k.y$	$c_k.z$	$\|\vec{d}_k - \vec{c}_k\|$
-43.06506	-6.11796	-30.09367	-43.06355	-6.11797	-30.09320	0.00158
5.44425	23.97245	20.27309	5.44446	23.97840	20.27387	0.00590
-37.60270	-14.73096	-11.31146	-37.60239	-14.73089	-11.31167	0.00038

Contributions

Jiaming Zhang completed the coding part and Section 1.1, Section 2, Section 3, and Section 4 of the report. Chongjun Yang did the rest part of this report.

References

[1] Jiaming Zhang, Chongjun Yang. PA1-REPORT

[2] Jiaming Zhang, Chongjun Yang. PA2-REPORT

[3] Jiaming Zhang, Chongjun Yang. PA3-REPORT

[4] https://kartiksonu.github.io/project/icp/slides.pdf

[5] https://en.wikipedia.org/wiki/Octree