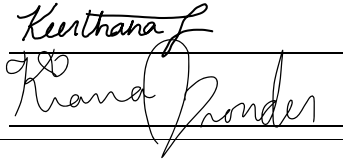


Programming Assignments 3 and 4 – 601.455/655 Fall 2023

Score Sheet (hand in with report) Also, PLEASE INDICATE WHETHER YOU ARE IN 601.455 or 601.655

(one in each section is OK)

Name 1	Keerthana Thammanna
Email	lthamma1@jhu.edu
Other contact information (optional)	
Name 2	Kiana Bronder
Email	kbronde1@jhu.edu
Other contact information (optional)	
Signature (required)	<p>I (we) have followed the rules in completing this assignment</p> <div style="text-align: center;">  </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	

1 Mathematical Approach

1.1 Registration and DK Calculation

We opted to implement a closed form solution for registration rather than an iterative method to have a quicker method. We used Arun's registration, which we describe below.

$$\begin{aligned}\bar{a} &= \frac{1}{N} \sum_{i=1}^N \vec{a}_i & \tilde{a}_i &= \vec{a}_i - \bar{a} \\ \bar{b} &= \frac{1}{N} \sum_{i=1}^N \vec{b}_i & \tilde{b}_i &= \vec{b}_i - \bar{b} \\ \mathbf{H} &= \sum_i \begin{bmatrix} \tilde{a}_{i,x} \tilde{b}_{i,x} & \tilde{a}_{i,x} \tilde{b}_{i,y} & \tilde{a}_{i,x} \tilde{b}_{i,z} \\ \tilde{a}_{i,y} \tilde{b}_{i,x} & \tilde{a}_{i,y} \tilde{b}_{i,y} & \tilde{a}_{i,y} \tilde{b}_{i,z} \\ \tilde{a}_{i,z} \tilde{b}_{i,x} & \tilde{a}_{i,z} \tilde{b}_{i,y} & \tilde{a}_{i,z} \tilde{b}_{i,z} \end{bmatrix}\end{aligned}$$

We then computed the singular value decomposition to get $\mathbf{H} = \mathbf{U}\mathbf{S}\mathbf{V}^t$, where the rotation matrix $\mathbf{R} = \mathbf{V}\mathbf{U}^t$. The translation was calculated using $T = \bar{b} - \mathbf{R}\bar{a}$. If the determinant of the rotation matrix is -1, the last column of \mathbf{V} was multiplied by -1 to make sure that the resulting rotation matrix has a determinant of 1. A determinant of -1 could be caused by having a very small rotation.

For each sample frame k of the marker bodies, the values $a_{i,k}$ and $b_{i,k}$, the positions of the LED markers with respect to the optical tracker, were registered using Arun's method with A_i and B_i , the positions of the LED markers, giving $F_{A,k}$ and $F_{B,k}$. The position of the pointer tip with respect to rigid body B was calculated using $d_k = F_{B,k}^{-1} F_{A,k} A_{tip}$.

1.2 Closest Point on Triangle

Since we are working with a surface mesh defined by triangles and we need to find the closest point on the surface to another point, we implemented a function that finds the closest point on a triangle to a given point.

If the vertices of a triangle are defined by p , q , and r , and the given point is a , the closest point on the triangle to a can be found by the following steps:

$$\begin{aligned}a - p &= \lambda(q - p) + \mu(r - p) \\ \begin{bmatrix} q - p & r - p \end{bmatrix} \begin{bmatrix} \lambda \\ \mu \end{bmatrix} &= [a - p] \longrightarrow Ax = b \longrightarrow x = (A^T A)^{-1} A^T b\end{aligned}$$

After calculating λ and μ ,

if $\lambda \geq 0, \mu \geq 0, \lambda + \mu \leq 1$: $c = p + \lambda(q - p) + \mu(r - p)$

if $\lambda < 0$: $\text{ProjectOnSegment}(c, r, p)$

if $\mu < 0$: $\text{ProjectOnSegment}(c, p, q)$

if $\lambda + \mu > 1$: $\text{ProjectOnSegment}(c, q, r)$

To project on a segment given c , p , and q :

$$\begin{aligned}\lambda &= \frac{(c - p) \cdot (q - p)}{(q - p) \cdot (q - p)} \\ \lambda^{(seg)} &= \text{Max}(0, \text{Min}(\lambda, 1)) \\ c^* &= p + \lambda^{(seg)} \times (q - p)\end{aligned}$$

1.3 Iterative Closest Point

1.3.1 Linear Search

After calculating d_k for each sample frame k and using $F_{reg} = 1$, the sample points s_k were found using $s_k = F_{reg} \cdot d_k$. To find the points c_k on the surface mesh, we initially did a linear search between each d_k and each triangle, finding the closest point on the triangle to the d_k and storing the shortest distance. This algorithm uses an $O(n)$ runtime, which is quite slow. As a result, we also made another optimized search with faster runtimes.

1.3.2 Optimized ("Binary") Search

We created a KD Tree* from Nodes that store point, triangle, index, right child, and left child information. We used starter code from Geeksforgeeks.com (contributed by Prajwal Kandekar) and modified it as necessary for this assignment. We kept only the insert and search functions but modified search so that it returns the nearest node rather than check the tree for if that specific point already exists in the tree. We used our own Euclidean distance function to calculate distances, and updated this with each recursive search throughout the tree. Each depth level in the KD tree is oriented with respect to a specific dimension, looping from 0 to 2 and back to 0 again. This allows for as close to a binary search as is possible with 3-dimensional information.

The tree is constructed with respect to the information stored at point, which is the centroid of the triangle's vertices. This was calculated as:

$$x_c = \frac{x_1 + x_2 + x_3}{3}$$

Where x_c is the x-coordinate of the centroid, and x_i is the x-coordinate of vertex i . The same calculations were used for the y and z coordinates. This was necessary because it seemed the easiest way to reference each individual triangle, since the alternative required Nodes for each individual vertex (3 times as many Nodes!) and storing the information of all triangles it contributes to. The search(...) function traverses these centroid points to find the closest Node to the given query point, and this returned Node's triangle information is used to calculate the correct c_k value using Euclidean distance.

* K = number of dimensions = 3

2 Algorithms

2.1 Registration

Transpose vectors if necessary. Calculate average and detract from vectors. Implement Arun's method by creating the H matrix using the calculated vectors. Calculate registration frame using SVD. After reading in $a_{i,k}$ and $b_{i,k}$, use a for loop for each k to find a transformation to A_i and B_i using Arun's method, and use those to calculate d_k .

2.2 Closest Point on Triangle

Transpose the point vectors as necessary to make sure that the matrices are the correct dimensions. Create the 2 known matrices shown in the mathematical approach and use least squares to find the unknowns. Using if-else statements to see if the closest point is on the edge of the triangle, calculate the final center point based on the conditions of the location.

2.3 Iterative Closest Point

2.3.1 Linear Search

Have one for-loop that iterates through all the d_k s, and inside, iterate through all the triangles. Find the closest point to the triangle for each d_k and save the point and the distance when distance was minimized.

2.3.2 Optimized ("Binary") Search

Iterate over all triangle vertices in the mesh and calculate their centroids. Create Nodes from these centroids and store their corresponding triangle vertices and triangle indices. Add each Node to the same tree. Iterate over each d_k and search the tree for its nearest Node. Use the Node's triangle information to find the closest point on the

triangle. Calculate the distance between this c_k and d_k . Calculate and add to the c_k and $\|d_k - c_k\|$ errors with each iteration.

3 Overview of Program

Important Files for PA3:

File Directory	
File	Description
FileIO.py	functions to read input and output1 files
GenerateOutput3.py	functions to read the unknown dataset files and generate an output3 txt file
Point3d.py	class to create 3d point objects and perform cartesian math functions
Frame.py	class to do frame transformations, save frame data, and transform points.
Registration.py	an implementation of the Arun's registration method
ClosestPointOnTriangle.py	functions to find the closest location of a triangle to a given point
Mesh.py	class to store the surface mesh, and functions for KD-tree
testing.py	testing functions we used to test and debug our functions

3.1 Packages

Packages used: NumPy==1.26.0 (important functions include matrix functions and np.linalg.svd), Math (used math.comb, comes with python==3.11.0), Time (comes with python).

3.2 Code Structure

We created files for the different parts of the project.

GenerateOutput3.py is the main files that should be run on input data to get output 3 files.

This file includes functions to read the unknown dataset files and generate the desired output files. The output includes our calculated d_k , c_k , and $\|d_k - c_k\|$ values.

testing.py is a script that takes an input debugging data set and outputs the error of our calculated variables compared to those given in out "-output1.txt" and "-output2.txt" files. It also includes a few component tests that we did in addition to the debugging tests.

testKDTree() checks to make sure that the insert and search functions work correctly given a tree that we designed.

testClosestPointToTriangle() checks to make sure that the correct closest point to a triangle is found given different conditions.

printPA3OutputErrorsLinearICP() prints the average d_k error, c_k error, and magnitude error, and the runtime of execution when a linear search is used to find the closest points.

printPA1OutputErrorsOptimizedICP() prints the average d_k error, c_k error, and magnitude error, and the runtime of execution when a KD-tree is used to find the closest points.

FileIO.py includes functions to read the body, mesh, samplereadings, and output3 files.

For each type of file, the exact dimensions of the output arrays are specified in the python file, so that users can look at the description and understand how the array is formatted.

Point3d.py and Frame.py include classes to create 3d point objects, do transformations, save frame data, and other Cartesian math functions.

Each Point3d object stores the name of the frame it resides in in addition to its x, y, z coordinates. The base code for this class was taken off Github (cited in code) though we added several additional features (e.g., frame variable and error() function) as necessary for this scenario. Each Frame object stores its 3D rotation matrix \mathbf{R} and 3D position vector \vec{p} , as well as its neighboring Frames so that the whole system may be traversed from one Frame object.

Registration.py has an implementation of the Arun's registration, following the algorithm detailed above.

registrationArunMethod() takes in 2 point clouds and finds a transformation \mathbf{F} such that $\mathbf{F} \cdot a = b$

Mesh.py creates a mesh object given the points of the vertices, indices of the vertices for each triangles, and each triangles' neighbors, along with functions to easy get the centroid of triangles and the vertices of triangles in array form. This files also contains the implementation of the KD-tree.

insert() inserts new nodes into the KD-tree and search() returns the triangle closest to a point based on Euclidean distance.

ClosestPointOnTriangle.py includes functions that take in a point and a triangle's vertices points, and returns the closest point on the triangle to the given point.

findClosestPointOnTriangle() is the main function that is called.

4 Verification of Code

4.1 Registration

We verified our Arun registration function by testing \mathbf{F}_D against the first frame's D coordinates (using debugging file "a") and confirmed that $\mathbf{F}_D^{-1} \cdot \vec{D}_0 = \vec{d}_0$ on MATLAB.

4.2 Closest Point on Triangle

Since there are multiple checks in the function that calculates the closest point on triangle, multiple tests that satisfy the different conditions were written. There are tests where the closest point should lie in the middle of the triangle, on the first edge, on the second edge, and on the third edge. These tests compared against expected results. An example of a test is shown below:

4.3 KD Tree

Following the example given on the Geeksforgeeks page, We changed it to include a third dimension since it was given for a 2D tree rather than a 3D one. testKDTree() in testing.py checks both the insert and search function by looking for the closest node to a given point. Comparison of the two outputs reveals that the correct answer is reached, validating our KD tree implementation.

4.4 Iterative Closest Point

Since d_k is calculated before ICP is called, these values are the same regardless of the type of search implemented. However, there were slight differences in the c_k and $||d_k - c_k||$ between the linear and optimized search methods. These results are summarized below, with the errors calculated by comparing our outputs to the debugging datasets A-F to the sample outputs provided.

Table 1: Differences in Errors

		c_k Error		Magnitude Error	
Dataset	d_k Error	Linear	Optimized	Linear	Optimized
A	0.012711	0.011038	0.032143	0.004924	0.025722
B	0.008872	0.006602	0.091757	0.004307	0.019176
C	0.008347	0.005995	0.069917	0.003712	0.009833
D	0.010232	0.009151	0.011820	0.002801	0.003407
E	0.585882	0.387412	0.376950	0.336209	0.330929
F	0.674781	0.516378	0.482767	0.298989	0.301194

The slight differences between the errors might be due to different triangles being found as the closest between the 2 search algorithms. For example, for debug dataset A, the following triangle indices were returned by the search algorithms:

As shown, a few of the d_k s had different triangle indices. One way to reduce error to make sure surrounding triangles don't have a closer point is to look at the neighbors of the triangle that is returned by the KD-tree search and check if there are any that are closer. Overall, due to the low amount of error, we verified that our KD-tree works well.

Table 2: Differences in Closest Triangles Found

d_k idx	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Linear	2425	1561	1187	1456	2285	3053	1677	2330	932	2552	2001	845	636	772	5
KD-Tree	2425	1561	1187	1456	2285	3053	1677	2327	932	2552	2004	845	636	816	4

We also tracked the runtime of the two search methods to validate that using a KD tree is indeed quicker. These results are displayed in the table below.

Table 3: Differences in Runtimes

Dataset	Execution Time (s)	
	Linear	Optimized
A	3.92902	0.09568
B	4.41851	0.09834
C	3.56319	0.09412
D	4.14773	0.12804
E	4.32264	0.09894
F	4.40827	0.12028

5 Results

5.1 Discussion

We believe that due to the error in d_k , the error may have propagated to c_k and the magnitude error, because different points on the surface mesh would be closer to our calculated d_k . We believe that this error is from our registration method, Arun’s registration method, which is not as robust to noise. Possibly adding an iterative component to registration would reduce the calculated error.

Additionally, the difference in c_k and index values between the two search methods is likely due to how the KD tree is implemented. The centroid of each triangle’s 3 vertices was used as the Node’s point, which is the variable of sorting and comparison for the tree algorithm. Therefore, the closest node is always with respect to the centroid, whereas some triangles may have larger surface areas (i.e., vertices are farther away from the centroid) yet further centroids that are overlooked despite their being the better solution.

Despite these slight differences in c_k and $\|d_k - c_k\|$, the optimized search is still the preferred choice for two main reasons. Firstly is the runtime, as shown in Table 3. There is a significant decrease to the runtime for the optimized search compared to the linear search—approximately 0.1 seconds (on average) compared 4.2 seconds (on average). The second reason is the range of error magnitude. Although the optimized search’s c_k and $\|d_k - c_k\|$ errors are higher for datasets A-D, sometimes by as large of a difference as a 16.67% increase (dataset C c_k error), this same magnitude increase is not maintained in noisier datasets. Datasets E and F actually have *lower* c_k errors for the optimized search, thereby leading to near identical magnitude errors between the two search methods. This means that the c_k and $\|d_k - c_k\|$ values have the same upper end error ranges between both search methods. Combined with the aforementioned improvement to runtime, the optimized search is much better than the linear search.

5.2 Unknown outputs

5.2.1 Dataset G

20 PA3-G-Unknown-Output.txt
-6.77 6.29 52.88 -7.08 6.36 52.89 0.316
-6.31 4.73 -22.89 -5.16 6.21 -23.77 2.074
20.44 -1.62 48.10 20.17 -1.32 48.08 0.401
-20.22 -21.64 -8.38 -20.10 -21.99 -7.83 0.663
7.53 19.42 45.81 7.34 20.53 45.89 1.126
17.72 19.68 32.81 18.75 20.92 32.96 1.616
17.31 -17.37 -5.52 17.30 -17.14 -5.56 0.238

-0.46 9.59 -8.72 -0.64 10.18 -8.84 0.625
11.76 16.95 59.84 10.61 19.26 61.57 3.102
-26.98 -33.68 -21.47 -26.59 -31.92 -22.04 1.894
19.06 15.45 47.66 19.83 15.95 47.96 0.961
30.20 12.29 10.48 32.15 13.64 12.63 3.202
-11.26 6.87 -25.23 -10.14 8.69 -25.65 2.183
-18.18 4.97 -42.08 -18.21 5.08 -42.22 0.179
20.08 -5.50 30.79 19.76 -4.73 30.65 0.843
22.53 -6.20 20.19 22.30 -5.46 20.11 0.772
-27.76 -21.84 -48.37 -27.19 -20.67 -46.05 2.660
26.89 19.94 3.67 28.39 22.07 3.73 2.608
17.27 20.26 -29.73 17.25 20.23 -29.68 0.057
39.03 0.21 -5.69 38.82 0.36 -5.72 0.259

5.2.2 Dataset H

20 PA3-H-Unknown-Output.txt
0.03 -11.68 54.72 1.67 -7.10 55.06 4.879
35.51 11.91 -8.23 36.85 12.70 -8.30 1.554
3.28 18.83 1.62 3.28 18.83 1.62 0.006
-1.87 -17.18 -4.43 -1.96 -16.06 -5.08 1.297
17.60 25.27 -19.84 17.59 25.32 -19.84 0.049
-4.50 -20.14 -10.77 -4.59 -20.01 -10.96 0.244
-18.67 -10.68 -49.16 -18.67 -10.68 -49.18 0.016
-4.45 -7.36 63.94 -2.43 -5.12 62.51 3.338
31.79 -8.77 -12.88 31.74 -8.72 -12.89 0.069
7.93 -9.87 63.49 7.69 -7.25 62.62 2.779
-35.36 -13.65 -9.14 -35.37 -13.88 -8.84 0.381
-0.94 -12.24 13.17 -1.05 -10.82 12.92 1.449
13.51 12.77 -26.21 13.75 13.41 -25.84 0.781
16.16 21.45 -26.96 16.75 22.68 -27.81 1.600
19.53 5.15 -27.65 20.08 5.31 -27.38 0.636
-34.46 7.02 -28.74 -34.52 7.33 -28.73 0.314
14.52 18.14 -29.07 13.96 17.64 -30.46 1.583
9.82 -10.45 -21.73 9.78 -10.35 -21.58 0.176
-26.60 -4.97 -8.81 -26.67 -4.88 -8.64 0.199
2.12 17.13 3.80 1.46 17.61 3.38 0.916

5.2.3 Dataset J

20 PA3-J-Unknown-Output.txt
-2.68 18.01 9.58 -2.81 18.12 9.45 0.219
15.46 12.36 53.92 19.27 15.18 54.55 4.784
-1.82 17.37 9.84 -2.60 17.99 9.11 1.232
-23.64 -28.75 -42.21 -23.24 -27.14 -40.77 2.197
-9.81 -0.42 52.21 -6.29 0.87 52.36 3.751
20.92 -5.21 32.55 20.24 -4.22 32.41 1.201
-32.05 -25.22 -43.01 -31.01 -23.86 -41.69 2.166
-10.24 -7.61 19.15 -7.79 -5.84 18.37 3.123
9.63 21.99 -8.86 9.76 21.88 -8.81 0.175
8.09 -11.05 22.42 7.24 -7.87 21.99 3.319
7.22 18.36 42.36 6.86 20.77 42.74 2.474
6.83 -9.61 29.02 6.54 -7.24 28.94 2.389
-3.15 -10.40 48.56 -0.98 -5.76 48.17 5.137
-2.34 -22.58 -16.50 -2.87 -21.99 -16.87 0.873

-0.68 18.44 9.95 -1.63 19.18 9.07 1.486
1.53 16.52 44.26 0.84 19.40 45.12 3.082
-12.49 10.21 14.90 -11.72 10.02 15.39 0.936
24.53 -3.99 25.63 23.83 -3.25 25.40 1.041
-42.49 -22.65 -21.36 -40.23 -21.10 -21.68 2.756
-4.38 -6.15 -29.64 -1.98 -5.08 -30.66 2.815

5.2.4 Dataset K

20 PA3-K-Unknown-Output.txt
1.92 -13.27 8.68 2.19 -11.84 8.53 1.467
-3.93 -12.59 -37.41 -2.10 -12.83 -38.21 2.013
-36.01 -8.18 -42.24 -36.73 -7.89 -43.05 1.114
-12.40 -5.65 -43.30 -10.50 -4.29 -45.61 3.286
-37.65 -22.93 -10.99 -35.87 -21.30 -12.94 3.098
17.03 -12.11 7.10 16.67 -10.86 6.45 1.459
-5.11 6.53 -9.89 -5.62 8.76 -10.02 2.290
-2.34 -4.84 43.43 -2.82 -5.38 43.37 0.726
20.19 -8.94 14.92 19.81 -7.12 14.45 1.919
28.02 19.49 -1.63 29.46 21.37 -1.74 2.367
11.61 22.11 59.96 11.41 20.40 59.75 1.737
10.32 21.85 27.40 10.32 23.05 27.65 1.231
-30.71 -29.71 -12.03 -29.35 -27.35 -14.01 3.376
22.31 11.93 39.34 23.36 12.25 39.48 1.104
-31.49 -12.40 -5.47 -31.10 -12.53 -6.28 0.908
-20.39 -14.48 -2.68 -19.93 -13.30 -4.87 2.527
-1.56 -21.15 -33.01 -1.73 -21.09 -32.93 0.195
-33.50 4.47 -31.62 -34.85 6.08 -32.21 2.185
16.83 -4.52 43.59 16.97 -4.76 43.60 0.286
-41.09 -19.93 -32.77 -40.11 -19.80 -32.48 1.028

6 Partner Work

Both Kiana and Keerthana worked on all the python files as well as the report.