

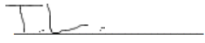
# PA3 report

Tyler Lehrfeld

November 2024

## Programming Assignments 3 and 4 – 601.455/655 Fall 2024

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	Tyler Lehrfeld
Email	tlehrfe2@jhu.edu
Other contact information (optional)	
Name 2	
Email	
Other contact information (optional)	
Signature (required)	I (we) have followed the rules in completing this assignment 

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	

600.455/655 Fall 2024

1 of 18

## 1 Problem statement

The problem I am trying to solve is essentially how do I know where I am pointing on a virtual mesh based on optical measurements. To do this I have optical of a reference body which lets me understand where the bone is relative to the optical tracker, and with all that information I can find where the tip of a pointer is because it is also tracked by the optical tracker. Then all I need to do is map the tip onto the mesh using a search.

## 2 Description of Mathematical Approach

### 2.1 3D point-cloud to point-cloud registration

For the 3D Point Cloud Registration, I used the Kabsch algorithm [1].

### 3 Closest point on triangle

I chose to use the method of least squares to estimate the parameters  $\lambda$  and  $u$ , then project on to a line if the parameters are outside of the triangle. The method is taken from Dr. Taylor's slides 11-15 on "Finding Point pairs" [2]. We have a point  $\mathbf{a}$  and a triangle  $\{\mathbf{p}, \mathbf{q}, \mathbf{r}\}$  where we need to find the closest point to on the triangle to  $\mathbf{a}$ . We first solve this equation using least squares:

$$\mathbf{a} - \mathbf{p} \approx \lambda(\mathbf{q} - \mathbf{p}) - u(\mathbf{r} - \mathbf{p})$$

$$\mathbf{A} = [\mathbf{q} - \mathbf{p} \quad \mathbf{r} - \mathbf{p}]$$

$$\begin{bmatrix} \lambda \\ u \end{bmatrix} = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T [\mathbf{a} - \mathbf{p}]$$

We then calculate the closest point of the triangle as

$$\mathbf{c} = \mathbf{p} + \lambda(\mathbf{q} - \mathbf{p}) - u(\mathbf{r} - \mathbf{p})$$

Now we must check if the point is inside the triangle. If it isn't, we project it onto the lines of the triangle. If  $\lambda < 0$ , we project  $\mathbf{c}$  on to the line  $\mathbf{rp}$ . If  $u < 0$ , we project onto  $\mathbf{pq}$ , and if  $\lambda + u > 1$ , we project onto the edge  $\mathbf{qr}$ . This ensures we find a closest point ON the triangle. We can project  $\mathbf{c}$  onto line  $\mathbf{ab}$  like so:

$$\lambda = \frac{(\mathbf{c} - \mathbf{a}) \cdot (\mathbf{b} - \mathbf{a})}{(\mathbf{b} - \mathbf{a}) \cdot (\mathbf{b} - \mathbf{a})}$$

Lambda must be between 0 and 1, so if it is below 0, it will be 0, and if it is above one, it will be brought down to one. Then we calculate the new, projected  $\mathbf{c}$  like so:

$$\mathbf{c} = \mathbf{a} + \lambda(\mathbf{b} - \mathbf{a})$$

### 3.1 Calculating $\mathbf{F}_d$

$\mathbf{F}_d$  is simply the transformation to the reference body frame from frame A. That means  $\mathbf{F}_d = \mathbf{F}_B^{-1} \mathbf{F}_A$ .

$\mathbf{F}_A$  and  $\mathbf{F}_B$  can be calculated using point cloud registration.

### 3.2 Bounding Sphere

I used Dr. Taylor's slides 17-19 to implement bounding sphere's on triangles [2].

Given a triangle  $\vec{a}, \vec{b}, \vec{c}$  and long side of a triangle  $(\vec{a}, \vec{b})$ , we need to find a  $\mathbf{q}$  such that a sphere will inscribe the triangle so that all of its corners are within one radius from the center. The minimum possible radius is  $\|\vec{a} - \vec{b}\|/2$ . If that radius contains  $\vec{c}$ , from  $(\vec{a} + \vec{b})/2$  then the center can be at  $(\vec{a} + \vec{b})/2$ . If not, we need to move the center perpendicular to  $\mathbf{AB}$  and on the triangle's plane so that the circle is expanded until the circle encloses  $\vec{c}$ . If we take the initial center to be  $\vec{f} = (\vec{a} + \vec{b})/2$

$\vec{u} = \vec{a} - \vec{f}$   
 $\vec{v} = \vec{c} - \vec{f}$   
 $\vec{d} = (\vec{u} \times \vec{v}) \times \vec{u}$  We get a  $\lambda = \frac{\vec{v}^2 - \vec{u}^2}{2\vec{d} \cdot (\vec{v} - \vec{u})}$  such that center =  $\min(0, \lambda)\vec{d} + \vec{f}$  The radius is just  $\|\vec{a} - \text{center}\|$

## 4 Algorithmic approach

### 4.1 Coding language

I used C++ for all of my coding

### 4.2 Closest Point on Triangle algorithm

The algorithm is described above in the mathematical approach.

### 4.3 Bounding Sphere

---

---

**Require:** Triangle triangle( $\vec{a}, \vec{b}, \vec{c}$ )

$\vec{a}, \vec{b} \leftarrow \text{getLongestSide}(\text{triangle})$

$\vec{c} \leftarrow \text{getOtherCorner}(\text{triangle})$

$\vec{f} \leftarrow (\vec{a} + \vec{b})/2$

$\vec{u} \leftarrow \vec{a} - \vec{f}$

$\vec{v} \leftarrow \vec{c} - \vec{f}$

$\vec{d} \leftarrow (\vec{u} \times \vec{v}) \times \vec{u}$

$\lambda \leftarrow \frac{\vec{v}^2 - \vec{u}^2}{2\vec{d} \cdot (\vec{v} - \vec{u})}$

center  $\leftarrow \min(0, \lambda)\vec{d} + \vec{f}$

radius  $\leftarrow \|\vec{a} - \text{center}\|$

**return** center

**return** radius

---

## 4.4 Fast and Slow closest point search

### 4.4.1 Slow Search

---

**Require:** point  $a$   
**Require:** list[point]  $mesh$   
**Require:** list[triangle]  $triangles$   
**Require:** int  $number\_of\_triangles$

```
while  $i \neq number\_of\_triangles$  do
    if  $get\_closest(triangles[i])$  is closer than  $closest$  to point  $a$  then
         $closest = get\_closest(triangles[i])$ 
    end if
     $i++$ 
end while
return  $closest$ 
```

---

### 4.4.2 Fast Search

---

**Algorithm 1**  $init\_oct\_tree\_node$

---

**Require:** array of bounding\_sphere pointers  $spheres$ , int  $num\_spheres$   
Assign  $spheres$  and  $num\_spheres$   
Call  $get\_bounding\_info$   
Call  $construct\_subtrees$

---

---

**Algorithm 2**  $get\_bounding\_info$

---

Initialize  $centroid$  to origin and  $max\_radius$  to radius of first sphere  
Set  $upper\_bound\_point$  and  $lower\_bound\_point$  to first sphere center  
**while** there is a next sphere **do**  
     $sphere \leftarrow next\ sphere$   
    Update upper and lower bound points if sphere center is beyond current bounds  
    Update  $max\_radius$  if sphere radius is larger  
    Accumulate sphere center into  $centroid$   
**end while**  
Calculate final center by dividing  $centroid$  by number of spheres

---

---

**Algorithm 3** construct\_subtrees

---

```
if num_spheres  $\leq$  min_num_spheres or bounds are within min_diag then
    has_subtrees  $\leftarrow$  false
    return
end if
Initialize vector num_spheres_in_each_box
Call Splitsort to sort spheres into boxes
Count viable boxes with non-zero spheres
Set num_subtrees to number of viable boxes
for each box with spheres do
    Create new subtree for spheres in box
end for
has_subtrees  $\leftarrow$  true
```

---

---

**Algorithm 4** Splitsort

---

```
Require: point center, list[bounding_sphere*] spheres, list[int]
num_spheres_in_each_box, int num_spheres
for i  $\leftarrow$  each box identifier (0 to 7) do
    for each sphere in list do
        Calculate difference from bounding box center and check if sphere
        falls in current box
        if sphere belongs in box then
            Swap sphere to current box position
        end if
    end for
    num_spheres_in_each_box[i]++
end for
```

---

---

**Algorithm 5** find\_closest (public)

---

```
Require: Matrix point, vector of Matrices vertices
Initialize closest to closest point on triangle using first sphere
Set bound as distance from point to closest
Call find_closest(private) with updated parameters
return closest
```

---

---

**Algorithm 6** find\_closest

---

**Require:** Point point, double& bound, Point& closest, list[points] vertices

```
    Calculate distance bound + max_radius
    if point exceeds any bound limits by distance then
        return
    end if
    if node has subtrees then
        for each subtree do
            Recursively call find_closest on subtree
        end for
    else
        for each sphere do
            update_closest(closest, sphere)
        end for
    end if
```

---

---

**Algorithm 7** update\_closest

---

**Require:** bounding\_sphere bsphere.i, Point point, double& bound, Point closest, list[Point] vertices

```
    Calculate distance from point to bsphere.i.center
    if distance - bsphere.i.radius is greater than bound then return
    end if
    Find_closest_point(bsphere.i.triangle, point)
    Recalculate distance from new closest point to point
    if distance < bound then
        Update bound and closest point
    end if
```

---

## 5 Overview of Program Structure

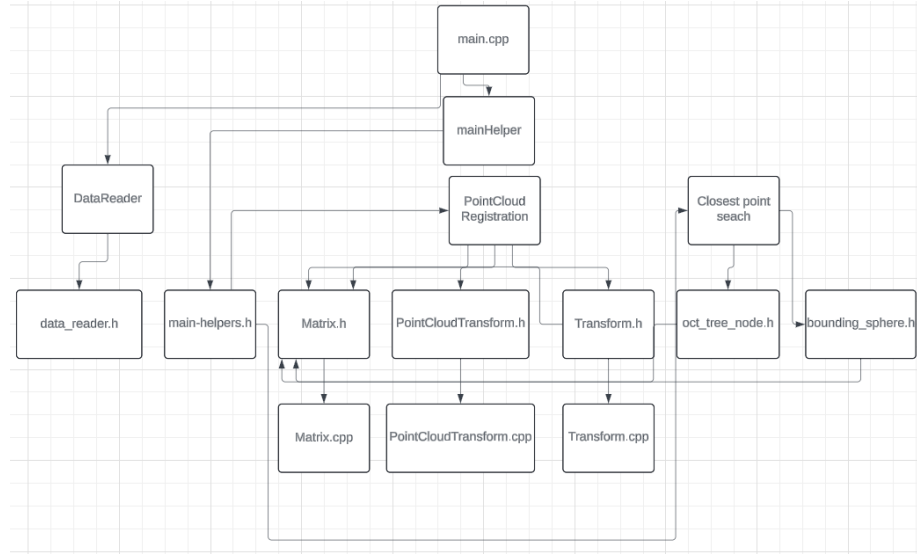


Figure 1: Hierarchy Chart

The main file is the file that pulls everything together to read data, and call helper functions in main-helpers.h. These helper functions call the algorithms described above which are located mostly located in oct\_tree\_node.h and bounding\_sphere.h. The data reading and data outputting functions are split up in data-reader.h and they read data as needed in from the main file. There is also a main-helper function that calculates the error on output. There are also two helper modules, one called helperFunctions.h for miscellaneous helper functions and the other called triangle-functions.h which just implements finding the closest point on a triangle to another point.

### 5.1 Dependencies

I used the C++ standard libraries and Eigen/Dense[3] For SVD in point-cloud registration.

## 6 Validation Approach

I implemented unit testing for base level operations like matrix, transform, pointcloud, bounding-sphere, and closest point on triangle. See PA2 for discussion of matrix, transform, and pointcloud testing. For bounding sphere and closest point, I designed my own unit tests by coming up with triangles by hand

and tested them that way. Higher level testing was done on the search algorithms. First I did a linear search and validated my approach with the debug data. Then, I was able to develop my oct\_tree and use the linear search data to verify that there was no difference in results. See figures 2 and 3 to see that the outputs are the same.

## 7 Results

My results were very similar to the debug results as shown by figure 2. A lot or all of the differences were simply due to rounding errors. It was clear that my octree was significantly faster. Across all test cases, it was 25 times faster than the simple linear search.

### 7.0.1 A

$\vec{s}$	$\vec{c}$	$  \vec{c} - \vec{s}  $
7.0344 14.5817 63.1612	7.03444 14.5818 63.1597	0.00144708
-7.00515 6.32865 41.5312	-7.01006 6.32838 41.5319	0.0049701
-11.5026 -16.3335 -46.0608	-11.5019 -16.334 -46.0619	0.00134009
-20.8844 10.6556 -35.2927	-20.8844 10.6563 -35.2932	0.000873166
26.1727 6.27154 31.0583	26.1721 6.2715 31.0582	0.000565403
-9.27779 8.20705 -15.9969	-9.2773 8.20067 -15.9973	0.00640643
-7.28723 5.53074 35.9481	-7.2918 5.52908 35.9486	0.00488551
-21.7418 -10.7874 -48.8965	-21.743 -10.7863 -48.8992	0.00309401
18.8955 -16.7724 -13.387	18.8943 -16.7798 -13.3898	0.00801197
-0.149698 8.29734 -15.8521	-0.150487 8.29169 -15.8506	0.00589344
-31.8942 -28.4852 -34.0071	-31.8969 -28.4899 -34.009	0.00570137
17.4032 1.48844 -22.5215	17.403 1.48844 -22.5216	0.000211808
-14.1194 -18.4423 -7.00074	-14.119 -18.4432 -6.99972	0.00139207
-4.89501 -2.34918 55.1882	-4.89727 -2.35075 55.188	0.00275415
-44.0747 -7.51524 -24.3199	-44.076 -7.51495 -24.3195	0.00145625



### 7.0.2 B

$\vec{s}$	$\vec{c}$	$  \vec{c} - \vec{s}  $
-37.6483 -9.48581 -12.7668	-38.8213 -9.64166 -11.907	1.46267
-18.6051 -9.65676 -4.29505	-19.117 -9.69631 -2.94045	1.44865
-2.14009 -7.91887 52.4488	-1.18696 -5.72672 52.6506	2.3989
17.8724 -12.5778 1.66919	17.8617 -12.5609 1.66221	0.0211932
-13.9544 21.5028 25.2969	-14.604 22.2213 25.2789	0.968745
28.9643 19.055 -13.7396	30.8069 20.1181 -13.9724	2.13998
24.2615 22.1227 -1.99782	25.4584 24.2645 -2.07189	2.4547
-27.7546 -23.8005 -48.3674	-27.0512 -21.4194 -45.7041	3.64111
-28.464 7.90253 -35.8924	-28.5501 8.08152 -36.0005	0.226145
-12.6558 9.56599 12.9006	-11.2033 8.78557 13.3039	1.69752
-27.1181 -28.6558 -14.3557	-27.0801 -28.508 -14.4163	0.164209
37.255 5.2578 4.71165	39.6482 4.99394 4.85205	2.41176
20.1763 -14.8724 -4.87099	20.5783 -15.3305 -4.36846	0.789907
-4.25822 12.8388 54.0807	-4.78726 13.0851 54.1533	0.588045
32.0833 15.8541 -2.09465	33.3062 16.9563 -2.28073	1.65677

### 7.0.3 C

$\vec{s}$	$\vec{c}$	$  \vec{c} - \vec{s}  $
-19.432 -30.6717 -20.324	-19.2411 -31.8896 -20.1887	1.24012
-28.2395 -17.0734 -9.53419	-28.4928 -16.8924 -7.38914	2.16754
8.57426 -10.057 -23.9407	8.14667 -9.30294 -22.8081	1.42625
37.658 10.1092 -2.8328	38.5951 10.6388 -2.97727	1.08605
9.24236 23.9493 -6.16654	9.62985 23.5518 -5.92196	0.606647
34.3888 2.48236 -22.7729	34.0949 2.4934 -22.5682	0.358308
-42.0588 -0.0219358 -26.3231	-41.5494 -0.376152 -26.304	0.62075
-25.4719 -15.8521 -50.2971	-24.8748 -15.5861 -48.0403	2.34952
-2.0873 -20.9073 -32.9965	-1.79264 -21.0146 -33.1302	0.340901
-15.1458 2.45182 1.83221	-14.9219 2.31438 1.76358	0.271528
13.5193 -8.10307 13.6702	13.585 -8.49032 13.7754	0.406634
-4.10646 -2.48831 -36.0349	-3.83761 -2.39648 -36.0274	0.284197
-10.2152 9.13799 -26.0151	-10.4264 8.82392 -25.9581	0.382736
25.453 -4.24763 16.8053 25.7167	-4.74349 17.0082	0.597134
-7.78233 8.31799 -17.0035	-7.78313 8.34361 -17.0021	0.0256718

#### 7.0.4 D

$\vec{s}$	$\vec{c}$	$  \vec{c} - \vec{s}  $
6.17715 20.9916 51.9871	6.45998 20.203 51.9056	0.841727
16.4082 -1.82436 44.7326	18.1104 -3.72568 45.2716	2.60827
-8.81133 -1.9083 62.1202	-5.65383 -1.4653 62.0443	3.18933
-11.5773 8.68655 58.8383	-6.95995 7.71017 58.8698	4.71956
8.53758 -4.6315 -22.5139	8.50268 -4.63894 -22.4046	0.114992
17.3113 -4.20451 32.7469	17.7039 -5.16886 32.9167	1.05497
12.9232 21.7853 28.5303	12.9227 22.8373 28.7403	1.07272
13.3107 -13.6875 3.59161	13.2651 -13.4721 3.43315	0.27122
-29.2511 -31.3695 -36.298	-28.683 -29.6044 -35.2521	2.12894
1.6608 -18.5669 -22.878	1.59388 -18.5261 -22.8509	0.0829704
-5.59817 18.3119 43.9874	-5.12929 17.4698 43.6646	1.01643
-6.30747 10.6441 -0.379266	-5.87867 10.0385 -0.293494	0.746972
-28.741 -14.5304 -8.36026	-29.0036 -15.2247 -6.8593	1.6745
-21.4636 -1.50633 -7.99164	-21.7424 -1.23065 -7.68382	0.498476
15.5378 -11.3045 -20.7811	15.4989 -10.5093 -20.0299	1.09464

#### 7.0.5 E

$\vec{s}$	$\vec{c}$	$  \vec{c} - \vec{s}  $
15.5885 1.11343 60.8587	15.6395 1.11555 62.9719	2.11383
12.4079 21.8459 -23.9166	12.7801 21.5255 -23.7692	0.512762
-38.7206 -22.2042 -14.3139	-37.2343 -21.4389 -15.1524	1.87023
15.8182 14.5838 48.3588	18.5534 17.4422 48.7247	3.97318
-40.4482 -12.7824 -13.0746	-39.6233 -12.662 -13.6432	1.00908
26.775 -10.876 -0.576283	26.7454 -10.607 -0.69617	0.295992
10.644 -14.969 3.36536	10.6104 -13.8171 2.9197	1.23559
-45.6387 -6.21911 -30.8499	-43.0756 -6.23732 -30.0571	2.68301
2.5743 -17.5709 -22.2519	2.47408 -17.5096 -22.2113	0.124261
1.59527 19.0659 -0.249124	3.2051 17.9795 0.405693	2.04954
8.50447 23.3375 13.3808	8.23649 25.1651 13.663	1.8686
9.93149 17.9739 -26.2947	11.3115 18.5375 -25.9883	1.52184
15.7158 -2.9724 56.4365	16.5735 -3.79958 56.5121	1.19403
18.4788 23.2116 -9.08711	18.4938 26.072 -9.54976	2.89762
-37.0116 3.70228 -20.9424	-37.2361 3.94413 -20.8363	0.346662

### 7.0.6 F

$\vec{s}$	$\vec{c}$	$  \vec{c} - \vec{s}  $
-6.16966 6.7737 -6.61955	-6.73559 8.39393 -6.74575	1.72086
33.7335 13.6227 6.41961	35.0617 14.8394 7.05045	1.90844
-9.21859 0.129841 -41.5746	-8.96903 0.276977 -41.8635	0.409161
-39.5429 -13.1877 -15.5553	-40.414 -13.3414 -14.9562	1.06835
-8.54938 -6.89252 25.0189	-6.89298 -5.51744 24.1132	2.33554
-11.3143 -16.4233 -47.9523	-12.108 -15.8477 -46.643	1.6357
6.12082 -7.71516 46.6315	6.04838 -6.98547 46.6617	0.733892
24.4003 -7.60242 15.5082	24.0135 -5.87983 15.1966	1.79277
5.43131 19.5873 43.1111	5.30145 20.4626 43.2491	0.895607
18.1968 6.55043 -26.7921	18.8564 6.92261 -26.2671	0.921586
-8.06384 3.74911 41.1006	-6.83661 3.94132 41.0451	1.24343
18.5869 12.4125 51.0231	21.0833 13.3662 51.3866	2.69708
26.9974 -4.36923 18.882	26.6286 -3.65742 18.6653	0.830458
1.85838 -13.6415 -28.8466	0.945 -13.4956 -28.4465	1.00779
-12.2277 5.44571 6.26588	-12.1612 5.39785 6.25795	0.0823096

### 7.0.7 G

$\vec{s}$	$\vec{c}$	$  \vec{c} - \vec{s}  $
-34.0176 -23.9366 -13.6179	-33.8993 -23.7985 -13.7197	0.208464
11.4175 21.5514 -18.2673	12.3454 21.2511 -18.2306	0.976019
18.0893 1.80043 -28.6281	20.9426 3.27337 -27.1902	3.51834
15.7018 13.111 47.581	19.8022 15.9859 47.9916	5.02468
-37.6891 -12.3268 -11.4111	-37.8886 -12.4029 -11.2105	0.293024
3.59752 16.6953 -6.85859	4.87163 15.1097 -5.39421	2.50639
-9.31201 -4.57336 48.8155	-5.06231 -2.14044 48.9353	4.89831
11.291 -7.89054 51.2031	10.9479 -6.49254 51.2931	1.4423
32.8038 -6.21482 -17.3954	33.3263 -6.5882 -17.5399	0.658195
17.0843 18.7871 -30.3374	17.1943 18.814 -30.6117	0.296757
35.2573 -2.79224 -13.9877	35.8853 -3.05892 -13.9476	0.683481
-3.86239 -17.8535 -39.0902	-3.80274 -17.8905 -39.1536	0.0945548
-38.4617 -4.44644 -45.5121	-36.0492 -5.41318 -42.774	3.7752
-29.1316 10.0279 -32.1395	-28.9483 9.42954 -31.884	0.675929
6.97165 -10.3198 16.4779	6.71625 -9.42579 16.2523	0.956783
29.0079 16.505 -18.0267	31.2065 17.7269 -18.8631	2.65077
14.7525 -19.2374 -13.9028	15.5915 -16.3813 -12.8041	3.17301
18.5992 21.2742 10.0458	19.584 24.5923 10.7988	3.5421
-14.8054 3.90116 13.5093	-12.2235 4.09884 13.1401	2.61564
16.8228 -8.43088 -26.0754	18.4257 -7.98362 -23.3202	3.21876

### 7.0.8 H

$\vec{s}$	$\vec{c}$	$  \vec{c} - \vec{s}  $
36.2539 -7.32629 -17.7121	34.2027 -5.65606 -16.9929	2.74124
8.36422 -17.9488 -17.0846	7.31041 -15.8225 -16.4096	2.46721
2.18975 17.8335 5.37797	1.17987 18.4381 4.57829	1.42298
40.2847 1.02203 3.84716	39.1168 1.57734 3.87517	1.29354
15.7359 22.167 39.4777	15.5567 21.7222 39.3596	0.493912
9.45925 -15.4004 -20.634	8.42642 -13.5658 -19.1803	2.55845
39.6902 6.08729 -6.69379	39.7347 6.09617 -6.70505	0.0468139
-37.9955 -8.68622 -38.9227	-39.7211 -8.6381 -39.3789	1.78555
28.6333 17.9362 -22.7862	28.6985 17.9883 -22.8465	0.102935
-7.57811 1.12992 -36.1892	-6.48089 2.0731 -36.7882	1.56598
-0.619359 -0.597982 62.8166	-0.605237 -0.60032 63.3982	0.581779
22.0834 12.8748 54.9591	20.9479 12.4809 54.8981	1.20341
-4.45353 15.1717 9.92322	-4.93273 15.7513 9.4336	0.897386
-29.4116 -11.6324 -47.0341	-29.5174 -11.6734 -47.4125	0.39506
19.4184 18.2862 51.4612	18.417 17.2397 51.3272	1.45468
-25.965 0.824633 -41.8101	-26.6164 2.55798 -43.6696	2.62425
-10.6462 -14.8724 -46.6241	-10.8164 -14.7489 -46.3432	0.350926
-4.02669 3.99888 61.8114	-4.2424 3.83405 63.2248	1.43927
-41.0429 -14.0345 -14.2165	-40.2053 -13.8866 -14.7926	1.02732
17.157 -20.0392 -10.1533	17.401 -17.715 -9.67415	2.38559

### 7.0.9 J

$\vec{s}$	$\vec{c}$	$  \vec{c} - \vec{s}  $
1.28312 -0.111278 -25.4958	0.763222 -0.156101 -24.6733	0.974108
22.6059 -8.38463 18.9811	21.7591 -5.78913 18.6976	2.74485
29.2108 1.57526 25.9234	27.5251 2.21012 25.313	1.90182
38.9774 -5.39068 -14.7104	35.5718 -3.94456 -14.9278	3.70626
15.0618 -19.1186 -2.43495	14.8961 -16.5797 -3.27229	2.67854
18.6898 9.21077 62.2811	18.7456 9.18967 63.0581	0.779328
15.2358 -11.9986 10.4704	14.5496 -9.82204 9.74125	2.3958
-22.0755 -23.79 -47.6569	-21.3891 -22.3655 -45.8001	2.43884
27.5605 1.31014 -30.211	27.6165 1.39522 -30.0266	0.210593
8.7969 -9.8474 28.8896	8.33881 -6.97988 28.6861	2.91101
1.70181 16.8794 5.33899	0.815449 17.9323 4.55583	1.58355
24.7706 -4.26275 32.7925	22.7874 -2.30517 32.2454	2.83983
31.9905 -10.2045 3.87308	30.6552 -8.04243 3.70059	2.54701
-25.9547 -16.4716 -48.8074	-25.6723 -16.3457 -47.7398	1.11151
29.9594 -3.85591 19.0692	28.4739 -2.43718 18.1595	2.24651
-9.75394 -18.9402 -48.227	-10.8324 -19.0231 -45.2349	3.18165
-33.1353 -26.5286 -38.7554	-32.5644 -25.7691 -38.1788	1.11141
3.0442 20.5928 10.3603	1.76521 22.5124 9.87683	2.35675
-6.13884 -32.6482 -25.0526	-8.84659 -29.1033 -25.7012	4.50766
-6.71113 1.22391 27.4363	-7.98931 1.03221 27.7062	1.32035

```
mean squared error on my outputs: 1.65596e-05
mean squared error on given outputs: 0
mean squared error on my outputs: 3.2742
mean squared error on given outputs: 3.16691
mean squared error on my outputs: 1.11946
mean squared error on given outputs: 1.12194
mean squared error on my outputs: 3.51259
mean squared error on given outputs: 3.51201
mean squared error on my outputs: 3.5874
mean squared error on given outputs: 3.57752
mean squared error on my outputs: 2.13191
mean squared error on given outputs: 2.12919
```

Figure 3: Mean Squared Error: fast

```
mean squared error on my outputs: 1.65596e-05
mean squared error on given outputs: 0
mean squared error on my outputs: 3.2742
mean squared error on given outputs: 3.16691
mean squared error on my outputs: 1.11946
mean squared error on given outputs: 1.12194
mean squared error on my outputs: 3.51259
mean squared error on given outputs: 3.51201
mean squared error on my outputs: 3.5874
mean squared error on given outputs: 3.57752
mean squared error on my outputs: 2.13191
mean squared error on given outputs: 2.12919
```

Figure 2: Mean Squared Error: slow

```
total time using slow algorithm in ms: 19551
total time using fast algorithm in ms: 776
```

Figure 4: Enter Caption

## References

- [1] Lawrence, Jim, et al. “A Purely Algebraic Justification of the Kabsch-Umeyama Algorithm.” *Journal of Research of the National Institute of Standards and Technology*, vol. 124, 9 Oct. 2019, <https://doi.org/10.6028/jres.124.028>. Accessed 7 Dec. 2022.
- [2] Taylor, R. H. “Finding point-pairs.” CIS I (601.455/655) Fall 2024 Schedule, JHU, 2010-2022, [https://ciis.lcsr.jhu.edu/lib/exe/fetch.php?media=courses:455-655:lectures:finding\\_point-pairs.pdf](https://ciis.lcsr.jhu.edu/lib/exe/fetch.php?media=courses:455-655:lectures:finding_point-pairs.pdf).
- [3] Guennebaud, G., Jacob, B., et al. (2010). \*Eigen v3.\* Retrieved from <http://eigen.tuxfamily.org>