Programming Assignments 1 & 2 (circle one)
601.455 and 601/655 (circle one) Fall 2018

Score Sheet (hand in with report)

Name 1	Tianyu Song
Email	tsong11@jhu.edu
Other contact information (optional)	
Name 2	Huixiang Li
Email	lhuixia1@jhu.edu
Other contact information (optional)	
Signature (required)	I (we) have followed the rules in completing this assignment Tianyu Song Huixiang Li

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	

a) Methods and Algorithms in Our Programs

1. Rotations

Rotating around xyz axis has the following equations for building rotation matrixes

$$R_x(\emptyset) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\emptyset & -\sin\emptyset \\ 0 & \sin\emptyset & \cos\emptyset \end{bmatrix}$$

$$R_y(\emptyset) = \begin{bmatrix} \cos\emptyset & 0 & \sin\emptyset \\ 0 & 1 & 0 \\ -\sin\emptyset & 0 & \cos\emptyset \end{bmatrix}$$

$$R_z(\emptyset) = \begin{bmatrix} \cos\emptyset & -\sin\emptyset & 0 \\ \sin\emptyset & \cos\emptyset & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

When performing the rotation in the order of roll, pitch, then yaw, the rotation matrix should be

$$R\,(\alpha,\beta,\gamma) = R_z(\alpha)R_y(\beta)R_x(\gamma)$$

[http://planning.cs.uiuc.edu/node102.html]

2. Skew matrix

$$\mathrm{skew}([\mathrm{x},\mathrm{y},\mathrm{z}]) = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix}$$

3. Rigid body transformation

Homogeneous representation for the rigid body transformation

$$F_1 = [R_1, p_1] = \begin{bmatrix} R_1 & p_1 \\ 0 & 1 \end{bmatrix}$$

Inverse of the rigid body transformation

$$F_1^{-1} = [R_1^{-1}, -R_1^{-1}p_1] = \begin{bmatrix} R_1^{-1} & -R_1^{-1}p_1 \\ 0 & 1 \end{bmatrix}$$

Multiplication of two homogeneous representation of the rigid body transformation

$$F_1F_2 = [R_1, p_1][R_2, p_2] = [R_1R_2, R_1p_2 + p_1] = \begin{bmatrix} R_1R_2 & R_1p_2 + p_1 \\ 0 & 1 \end{bmatrix}$$

4. **Point cloud to point cloud registration**

Step 1: compute the center of the two point clouds

$$\bar{a} = \frac{1}{N}\sum_{i=1}^{N}\overrightarrow{a_i} \qquad \widetilde{a_i} = \overrightarrow{a_i} - \bar{a}$$

$$\bar{b} = \frac{1}{N}\sum_{i=1}^{N}\overrightarrow{b_i} \qquad \widetilde{b_i} = \overrightarrow{b_i} - \bar{b}$$

Step 2: use direct techniques to solve R that minimizes, the details are shown in **5**

$$\sum_i (R\widetilde{a_i} - \widetilde{b_i})^2$$

Step 3: calculate p base on the solved R

$$\mathrm{p} = \bar{b} - R\bar{a}$$

Step 4: obtain the rigid body transformation

$$F = [R\,, p\,]$$

5. **Direct techniques to solve R [1]:**

Step1: compute H matrix

$$H = \sum_i \begin{bmatrix} \tilde{a}_{i,x}\tilde{b}_{i,x} & \tilde{a}_{i,x}\tilde{b}_{i,y} & \tilde{a}_{i,x}\tilde{b}_{i,z} \\ \tilde{a}_{i,y}\tilde{b}_{i,x} & \tilde{a}_{i,y}\tilde{b}_{i,y} & \tilde{a}_{i,y}\tilde{b}_{i,z} \\ \tilde{a}_{i,z}\tilde{b}_{i,x} & \tilde{a}_{i,z}\tilde{b}_{i,y} & \tilde{a}_{i,z}\tilde{b}_{i,z} \end{bmatrix}$$

Step2: calculate the SVD decomposition of H

$$H = USV^t$$

Step3: solve the rotation matrix based on the eigenvectors obtained

$$R = VU^t$$

Step4: verify that

if $Det(R) = 1$, then the R is the rotation we desired

if $Det(R) = -1$, there’s a reflection exists

(a) if one singular value of H is zero,

$$R' = V'U^t$$

where V' is obtained from V by changing the sign of the third column

(b) if none of singular value of H is zero, this approach is not supposed to be appropriate

6. Pivot Calibration

The known is a set of transformations from the frames of the tracker device to the different probe frames. The unknown is the coordinates of the dimple relative to the tracker device and those of the tip relative to probe frames, which are constant.

Given

$$F_i = \begin{bmatrix} R_i & P_i \\ 0 & 1 \end{bmatrix}$$

Calculate

$$p_{tip}$$

$$p_{pivot}$$

The formulation:

$$F_i p_{tip} = p_{pivot} = R_i p_{tip} + P_i$$

Thus,

$$R_i p_{tip} - p_{pivot} = -P_i$$

Stack all the equations together, we can form:

$$\begin{bmatrix} \cdots & \cdots \\ R_i & -I \\ \cdots & \cdots \end{bmatrix} \begin{bmatrix} p_{tip} \\ p_{pivot} \end{bmatrix} = \begin{bmatrix} \cdots \\ -P_i \\ \cdots \end{bmatrix}$$

The steps taken to solve this:

Step1: Initial guess by Moore–Penrose inverse [2]:

Let A = $\begin{bmatrix} \cdots & \cdots \\ R_i & -I \\ \cdots & \cdots \end{bmatrix}$ and b = $\begin{bmatrix} \cdots \\ -P_i \\ \cdots \end{bmatrix}$, and x = $\begin{bmatrix} p_{tip} \\ p_{pivot} \end{bmatrix}$, then $x = (A^T A)^{-1} A^T b$.

Step2. Use least square method to solve dx= $(A^T A)^{-1} A^T (b - Ax)$

Step3. Calculate x = x+dx and evaluate dx.

If dx <1e-5, then return x as the result, else go to Step 2.

b) Description of Functions

Functions	**Input variable**	**Output variable**
__init__.py	/	/
# Initialize the environment		
angle_2_rot.py	***R_angle*** angle array(3*1 or 1*3) in rad	***R_rot*** rotation matrix(3*3)
# Generate the rotation matrix from angle matrix using left multiplication		
rot_2_angle.py	***R_rot*** rotation matrix(3*3)	***R_angle*** angle array(3*1 or 1*3) in rad
# Generate the angle matrix from rotation matrix		
F_homo_multiple.py	***F1 F2*** rigid transform matrix (4*4)	***F3*** result rigid transform matrix (4*4)
# Generate the rigid body transformation matrix between two homogenous 4*4 matrices		
F_Rp_multiple.py	***R1 R2*** rotation matrix(3*3), ***p1 p2*** translation matrix(3*1)	***F3*** result rigid transform matrix (4*4)
# Generate the rigid body transformation matrix between two rotation matrix and translation matrices		
invtrans.py	***F*** rigid transform matrix (4*4)	***inv(F)*** inverse transform of F matrix(4*4)
# Inverse of rigid body transformation, from F to inv(F)		

rot.py	***angle_value*** anlge in rad	***Rot*** a rotation matrix (3*3) around specific xyz axis
# Generate rotation matrix for specific xyz axis		
trans.py	***R*** rotation matrix (3*3) and ***p*** translation (3*1)	***F*** rigid body transformation matrix (4*4)
# Generate the rigid body transformation matrix from rotation matrix and translation matrix using homogeneous representation		
trans_angle.py	***angle*** array (1*3) and ***p*** translation (3*1)	***F*** rigid body transformation matrix (4*4)
# Generate the rigid body transformation matrix from angle array and translation matrix using homogeneous representation		
skew_build.py	***x*** array (3*1 or 1*3)	***skew(x)*** skew matrix (3*3)
# Build up a 3*3 skew matrix with an array (3*1 or 1*3)		
point_cloud_registration.py	***c1 c2*** two 3D point clouds(n*3)	***F*** rigid transform matrix (4*4)
# Point cloud registration, compute the rigid transform matrix with two input point clouds		
pivot_calibration.py	***NF*** n rigid transformations matrix(4*4*n)	***P*** two translations in one array(6*1)
# Pivot calibration, compute the two translations using n input transformations matrix		
Q4.py	Provided files in the PA data folder	***C_expected, F_D, F_A*** the calculated results
# Read in the data from the provided folder, and calculate C_i_expected and F_D		
Q5.py	Provided files in the PA data folder	***P_EM_dimple***
# Read in the data from the provided folder, and calculate P_dimple		

Q6.py	Provided files in the PA data folder	***P_opt_dimple***
# Read in the data from the provided folder to perform a pivot calibration of the optical tracking probe		

c) Results of Functions. This section should include all the test functions you have written for your package together with their results. You do not need to write a page for each function. A couple of sentences with some results showing the function has reasonably been tested and works. In addition, show the results of your main function that the assignment asks. Based on the questions you can summarize the results in a table or demonstrate them in appropriate figures.

We build a test_pkg including functions for testing. The main method for testing is that we randomly generate data and apply our functions on the generated data, then calculated the residual error between the calculated one and the ground truth.

(1)

test_point_cloud_registration.py	***num_rounds*** a value for automatic test times	***residual*** total residual value

To test the point_cloud_registration.py, we build a test_point_cloud_registration.py, in this function, we first generate a rigid body transformation matrix and a point cloud data, then apply this transformation to the point cloud data to get a new point cloud. In this situation, we have the ground truths of two point clouds and the real transformation matrix. Then we use the point_cloud_registration.py to calculate the transformation matrix, and get the residual error between two transformation matrices. If the error is in the acceptance range, like e-10, which is close to zero, then it means our function is good and with small enough error.

```
the total residual matrix
[[  2.39100189e-14   2.27456942e-14   1.88217497e-14   2.25167107e-14]
 [  2.76723089e-14   2.58074812e-14   2.10451231e-14   2.72004641e-14]
 [  2.13579154e-14   1.47525221e-14   2.15383267e-14   2.35089725e-14]
 [  0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00]]


the sum of all element in the total residual matrix
2.70877287614e-13
[Finished in 0.8s]
```

After running this automatic test program, we find even sum all errors in 100 rounds, it is still less than e-10, in order words, our point_cloud_registration.py function has a good accuracy when calculating the rigid transformation between two point clouds.

(2)

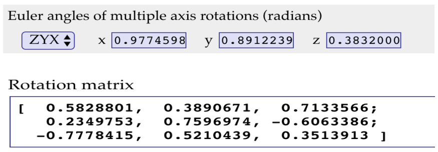
test_angle_rot.py	***num_rounds*** a value for automatic test times	***residual*** total residual value
# implement an automatic test for angle_2_rot.py and rot_2_angle, return total residual error between calculated angle and the generated angle		

To test the angle_2_rot.py and the rot_2_angle.py, we build a test_angle_rot.py. In this function, we first generate random angle array, then apply angle_2_rot.py function to obtain the rotation matrix, then apply the rot_2_angle.py to transform the rotation matrix back to angle array. Compare the two angle arrays to get the residual error. If the error is in the acceptance range, like e-10, which is close to zero, then it means our function is good and with small enough error. In addition, we also compare print out the angle array and rotation matrix, compare it with some online calculator.

```
the total residual array
[[  2.48119667e-15   2.62984079e-15   2.85925797e-15]]


the sum of all residual error
7.97029543059e-15
[Finished in 0.7s]
```

```
[[ 0.97745988  0.89122393  0.38320008]]
[[ 0.58288006  0.38906705  0.71335661]
 [ 0.23497532  0.75969738 -0.6063386 ]
 [-0.77784152  0.52104388  0.3513913 ]]
```



[website source: https://www.andre-gaschler.com/rotationconverter/]

After comparing the rotation matrix we get in our function with the one in the online calculator, we find they are the same. Also, the residual error is almost zero. At all, our angle_2_rot.py and rot_2_angle.py functions are good to use.

d) Discussion of the Results and Analysis

We compare our results with the debug results and calculated the residual between them in each dataset. Here we only show the first nine lines, where the first line corresponds to the EM pivot calibration result, i.e. Question 5, the second line is the optical pivot calibration result, i.e. Question 6 and the remaining is the C expected results, i.e. Question 4.

Dataset	Provided Results	Our Results	Residual

a	202.89, 190.20, 201.55	202.89, 190.2 , 201.55	0.0, 0.0 , 0.0
	403.49, 390.87, 199.80	403.49 , 390.87 , 199.8	0.0 , 0.0 , 0.0
	210.20, 208.33, 211.74	210.2 , 208.34 , 211.74	0.0 , -0.01 , 0.0
	209.61, 207.96, 336.73	209.61 , 207.96 , 336.73	0.0 , 0.0 , 0.0
	209.03, 207.59, 461.73	209.03 , 207.59 , 461.73	0.0 , 0.0 , 0.0
	208.08, 333.32, 212.10	208.08 , 333.32 , 212.1	0.0 , 0.0 , 0.0
	207.50, 332.94, 337.10	207.5 , 332.94 , 337.1	0.0 , 0.0 , 0.0
	206.92, 332.57, 462.10	206.92 , 332.57 , 462.1	0.0 , 0.0 , 0.0
	205.97, 458.30, 212.47	205.97 , 458.3 , 212.47	0.0 , 0.0 , 0.0
b	194.49, 200.42, 199.63	194.49 , 200.42 , 199.63	0.0 , 0.0 , 0.0
	402.36, 393.91, 197.35	402.36 , 393.9 , 197.35	0.0 , 0.01 , 0.0
	209.51, 208.45, 211.69	209.27 , 208.54 , 211.56	0.24 , -0.09 , 0.13
	206.95, 212.05, 336.65	206.88 , 212.13 , 336.49	0.07 , -0.08 , 0.16
	204.31, 215.71, 461.44	204.49 , 215.72 , 461.42	-0.18 , -0.01 , 0.02
	206.58, 333.45, 207.85	206.69 , 333.46 , 207.93	-0.11 , -0.01 , -0.08
	204.06, 336.91, 332.79	204.3 , 337.05 , 332.85	-0.24 , -0.14 , -0.06
	201.84, 340.98, 457.52	201.91 , 340.64 , 457.78	-0.07 , 0.34 , -0.26
	203.88, 458.87, 203.88	204.11 , 458.38 , 204.29	-0.23 , 0.49 , -0.41
c	211.72, 191.78, 200.50	211.72 , 191.79 , 200.51	0.0 , -0.01 , -0.01
	395.09, 399.24, 209.38	395.09 , 399.24 , 209.38	0.0 , 0.0 , 0.0
	210.49, 207.77, 209.36	210.27 , 208.26 , 209.47	0.22 , -0.49 , -0.11
	211.82, 211.87, 334.12	211.89 , 211.91 , 334.4	-0.07 , -0.04 , -0.28
	213.22, 215.93, 459.03	213.5 , 215.55 , 459.34	-0.28 , 0.38 , -0.31
	211.11, 332.92, 205.93	210.92 , 333.21 , 205.81	0.19 , -0.29 , 0.12
	212.70, 336.74, 330.72	212.53 , 336.85 , 330.75	0.17 , -0.11 , -0.03
	214.22, 340.54, 455.66	214.14 , 340.5 , 455.69	0.08 , 0.04 , -0.03
	211.72, 458.14, 202.48	211.57 , 458.15 , 202.16	0.15 , -0.01 , 0.32

d	204.38, 205.39, 191.75	204.38 , 205.39 , 191.75	0.0 , 0.0 , 0.0
	408.15, 399.37, 191.80	408.15 , 399.37 , 191.81	0.0 , 0.0 , -0.01
	210.87, 209.79, 209.85	210.87 , 209.79 , 209.84	0.0 , 0.0 , 0.01
	209.08, 207.77, 334.82	209.07 , 207.77 , 334.82	0.01 , 0.0 , 0.0
	207.28, 205.76, 459.79	207.27 , 205.76 , 459.79	0.01 , 0.0 , 0.0
	207.26, 334.72, 211.81	207.25 , 334.72 , 211.81	0.01 , 0.0 , 0.0
	205.46, 332.71, 336.78	205.46 , 332.71 , 336.78	0.0 , 0.0 , 0.0
	203.67, 330.69, 461.75	203.66 , 330.69 , 461.75	0.01 , 0.0 , 0.0
	203.64, 459.65, 213.77	203.64 , 459.65 , 213.77	0.0 , 0.0 , 0.0
e	197.12, 206.27, 196.52	197.11 , 206.27 , 196.52	0.01 , 0.0 , 0.0
	403.09, 390.63, 191.35	403.09 , 390.63 , 191.35	0.0 , 0.0 , 0.0
	209.56, 212.07, 211.79	208.86 , 210.97 , 211.44	0.7 , 1.1 , 0.35
	212.32, 214.27, 336.90	211.59 , 214.42 , 336.36	0.73 , -0.15 , 0.54
	215.32, 216.81, 462.35	214.32 , 217.87 , 461.28	1.0 , -1.06 , 1.07
	209.95, 335.87, 207.29	209.28 , 335.92 , 207.98	0.67 , -0.05 , -0.69
	212.54, 339.79, 332.38	212.02 , 339.37 , 332.9	0.52 , 0.42 , -0.52
	215.27, 343.85, 457.85	214.75 , 342.82 , 457.82	0.52 , 1.03 , 0.03
	210.71, 459.26, 203.34	209.71 , 460.87 , 204.52	1.0 , -1.61 , -1.18
f	192.35, 219.12, 193.70	192.35 , 219.11 , 193.69	0.0 , 0.01 , 0.01
	402.15, 405.80, 201.26	402.15 , 405.8 , 201.25	0.0 , 0.0 , 0.01
	209.58, 208.68, 212.80	209.63 , 208.7 , 211.85	-0.05 , -0.02 , 0.95
	211.37, 210.54, 336.55	209.83 , 211.05 , 336.83	1.54 , -0.51 , -0.28
	213.06, 212.95, 460.52	210.03 , 213.39 , 461.8	3.03 , -0.44 , -1.28
	208.28, 333.90, 209.80	208.26 , 333.67 , 209.5	0.02 , 0.23 , 0.3
	209.66, 336.33, 334.43	208.46 , 336.02 , 334.48	1.2 , 0.31 , -0.05
	210.53, 339.34, 459.37	208.66 , 338.36 , 459.46	1.87 , 0.98 , -0.09
	207.68, 459.20, 206.52	206.89 , 458.64 , 207.16	0.79 , 0.56 , -0.64

g	206.05, 204.26, 192.64 396.41, 401.91, 203.86 210.65, 209.84, 209.02 210.28, 206.38, 333.04 209.80, 202.42, 457.84 212.44, 333.02, 212.56 212.81, 330.78, 337.75 212.88, 327.38, 462.86 215.09, 456.83, 215.86	206.05 , 204.24 , 192.64 396.41 , 401.91 , 203.86 209.79 , 208.38 , 208.68 209.15 , 204.98 , 333.63 208.51 , 201.58 , 458.59 212.82 , 333.3 , 212.1 212.18 , 329.9 , 337.05 211.54 , 326.5 , 462.0 215.86 , 458.22 , 215.51	0.0 , 0.02 , 0.0 0.0 , 0.0 , 0.0 0.86 , 1.46 , 0.34 1.13 , 1.4 , -0.59 1.29 , 0.84 , -0.75 -0.38 , -0.28 , 0.46 0.63 , 0.88 , 0.7 1.34 , 0.88 , 0.86 -0.77 , -1.39 , 0.35
h	/	205.64 , 236.1 , 216.23 392.34 , 395.75 , 203.76 208.52 , 209.48 , 209.92 208.11 , 209.18 , 334.92 207.69 , 208.88 , 459.91 205.64 , 334.45 , 210.21 205.22 , 334.15 , 335.21 204.81 , 333.85 , 460.21 202.75 , 459.41 , 210.5	/
i	/	201.42 , 192.71 , 203.32 409.36 , 397.62 , 202.95 208.52 , 209.55 , 209.31 210.51 , 209.17 , 334.29 212.5 , 208.79 , 459.28 206.44 , 334.53 , 209.72 208.43 , 334.16 , 334.7 210.42 , 333.78 , 459.69 204.36 , 459.52 , 210.13	/

j	/	204.35 , 203.37 , 198.7 406.79 , 397.43 , 200.2 209.48 , 211.41 , 208.91 210.84 , 209.12 , 333.89 212.2 , 206.84 , 458.86 208.4 , 336.38 , 211.21 209.76 , 334.1 , 336.18 211.12 , 331.81 , 461.15 207.32 , 461.35 , 213.5	/
k	/	198.68 , 193.63 , 202.39 400.47 , 398.48 , 197.75 211.07 , 210.44 , 211.14 210.48 , 211.52 , 336.13 209.89 , 212.6 , 461.13 207.46 , 335.38 , 210.04 206.87 , 336.46 , 335.03 206.28 , 337.55 , 460.03 203.85 , 460.32 , 208.94	/

From the table, we find that the residual errors during the calculation are small, and some of them are almost zero. The biggest residual error is 3.03, which is 1.4%. Since we already did automatic check our programming, the calculation should be very close to ground truth. Therefore, we suppose this error comes from the data collected. But after all, our error rate is less than 1% for most of the data. Based on the residual shown and our previous program test, we believe our program can provide a correct calculation regarding the pivot calibration and point cloud to point cloud registration.

e) Work Distribution

Name	Work
Tianyu Song	• Math Packages • Point cloud to point cloud registration program • Question 4 • Question 6 (collaborated with Huixiang)
Huixiang Li	• Pivot calibration program • Question 5 • Question 6 (collaborated with Tianyu Song) • Program testing

References

[1] K.S. Arun, T.S. Huang, and S.D. Blostein. Least-Squares Fitting of Two 3-D Point Sets. IEEE PAMI. Vol. 9. No. 5. Sept. 1987: 698-700.

[2] E. H. Moore. On the reciprocal of the general algebraic matrix. Bulletin of the American Mathematical Society, 26(9):394–395, 1920. doi:10.1090/.