Programming Assignment #1

Computer Integrated Surgery I
10/28/2021

Sean Darcy
sdarcy2@jhu.edu

Alexandra Szewc
aszewc1@jhu.edu

Description of formulation and algorithmic approach

We decided to develop our program in Python. Specifically, we chose to develop a python notebook (.ipynb) in Google Collab so that we could code together simultaneously and organize our code by task with instructions displayed.

We began by developing a Cartesian math package for 3D points, rotations, and frame transformations. We imported and developed proficiency with Numpy, a standard numerical library with support for large, multi-dimensional arrays and matrices, as well as a collection of functions that operate on those arrays. We implemented 3-D points as 1x3 Numpy arrays. We defined several distinct methods to generate rotation matrices based on varying input parameters, described in the program overview section that follows. We implemented frames as python classes with corresponding rotation matrices R and translations/offsets p. F is represented homogeneously as a 4x4 vector. Key linear algebra operations were also implemented, like skew(), norm(), etc. (described in detail in the following section).

Next, we implemented two methods for 3D point set to 3D point set rigid registration. The first is based on Arun's method which involves SVD on the matrix H generated from the point sets a and b. Unfortunately, Arun's method has 2 failing cases. The first- when det(R) = -1- we were able to handle based on Arun's paper. Because of the second failing case, when det(R) = -1 and none of the singular values of H are 0, we decided to implement a second method for 3D point set to 3D point set rigid registration method based on the unit quaternion.

We then implemented pivot calibration based on sphere fitting, as described in https://mphy0026.readthedocs.io/en/latest/calibration/pivot_calibration.html. We chose to implement pivot calibration based on sphere fitting as it made sense to estimate the possible pivoting poses of the tracked pointers as spheres centered at the pivot dimple.

Also, Ma et al. found that sphere fitting was a superior calibration method when data quality was good.

To perform steps 4, 5, and 6 in the assignment, we first had to develop several file input/output functions to read in the provided data, including calbody.txt, calreadings.txt, empivot.txt, optpivot.txt, and to output our results in output1.txt. To do so, we imported Pandas, which offers data structures and operations for manipulating numerical tables and time series in Python.

Next, we took a distortion calibration data set and computed the expected values C_exp for Ci using the give_C_exp() function. To do so, we first computed the transformation between optical tracker F_D and EM tracker coordinates, which involved computing a frame F_D such that $D_j = F_D * d_j$ using rigid registration. Next, we computed the transformation F_A between calibration object and optical tracker coordinates, which again involved computing a frame F_A such that $A_j = F_A * a_j$ using rigid registration. Finally, we were able to compute and output C_exp by $F_D^{-1} * F_A * cj$.

We were then able to read in the EM tracking data to perform pivot calibration for the EM probe. We used the first frame of the calibration data to define a local probe coordinate system as described in the assignment instructions. We then used our rigid registration method based on unit quaternions to determine Fg, the pose of the EM probe with respect to the EM tracker. We then were able to apply our pivot calibration method to determine the best estimate for ptip_g, the tip of the EM pointer with respect to its pose Fg.

We conducted a similar process to calibrate the optically tracked pointer, but first had to transform optical tracker beacon positions into EM coordinates by applying Fd^-. Next, we were able to again apply pivot calibration to solve for ptip_D, the tip of the optically tracked pointer with respect to its pose Fd.

<u>Program overview</u>

rotFromComponents(Rx, Ry, Rz)

This function generates a 3D rotation matrix from 3 input component 3D rotation matrices. This is derived from the fact that any rotation can be described by consecutive rotations about three primary axes x, y, and z.

Rxyz(alpha, beta, gamma) = R(x, alpha)*R(y, beta)*R(z, gamma)

rotFromAngles(alpha, beta, gamma)

This function generates a 3D rotation matrix from 3 input rotation angles. 3 rotation matrices Rx, Ry, and Rz are generated from alpha, beta, and gamma. The method rotFromComponents(Rx, Ry, Rz) is then called to return the final rotation matrix R.

identity()

This function returns a 3x3 Numpy array representing the identity matrix:

```
[1, 0, 0]
[0, 1, 0]
[0, 0, 1]
```

norm(v)

This function returns the norm of the vector v, computed as v / sqrt(v*v). The Numpy functions np.dot()- which computes the dot product of two input parameters v and v- and np.sqrt()- which computes the square root of this result- are called.

skew(v)

This function generates a skew matrix from the input, which is a normalized (via call to norm() within the method) 1x3 Numpy array:

```
[0,    -Vz,  Vy]
[Vz,   0,    -Vx]  = skew(v)
[-Vy,  Vx,   0]]
```

smallAngleR(v, theta)

This function returns a small angle approximation for R ≅ [I + skew(norm(v)*theta)]

homogeneousVector(v, scale)

This function generates a 4-D homogeneous representation of 3-D vector v, with scaling factor scale.

scale * [Vx, Vy, Vz, 1]

class Frame

This class represents a frame, or pose, consisting of a Rotation matrix R and offset/translation vector p.

Methods:

getFrame(self)

Method defines frame with rotation R and translation as parameters and returns F

appFrame(self, v)

Method applies frame transformation F to input vector v, F*v = R*v + p.

appInvFrame(self, v)

Method applies frame transformation F^- to input vector v, F^- * v = R^- * v - R^- * p. R^- is computed via Numpy R.T, representing the transpose of R, R^T. Rotation matrices have the property that R^T = R^-.

Next, we developed 3D point set to 3D point set registration algorithms. One based on Arun's method, and another based on unit quaternions.

rigid_registration_Arun(a, b)

This function takes two point clouds, a and b, as parameters. First, the means of a and b are calculated using np.mean() and subtracted from each respective point in a and b, and stored in A and B. Next, the matrix H is computed via np.dot(A, B.T). The SVD of H is computed via np.linalg.svd(H), with output matrices U, S, V^T. Next, R is computed by np.dot(V^T.T, U.T) = np.dot(V, U.T). There are two potential failing cases. If det(R) = -1, then R = Vprime*U.T, where the third column of Vprime is the product of the third column of V multiplied by -1, and the 1st and second columns are equivalent. The second failing case is when none of the singular values of H are 0. We handled the first failing case as described, but for the second case chose to implement a different flavor of 3D point set to 3D point set registration. The function returns a frame corresponding to the transformation from a to b.

rigid_registration(a, b)

This function, based on the unit quaternion, takes two point clouds, a and b, as parameters. Similar to Arun's method, the means of a and b are first calculated using np.mean() and subtracted from each respective point in a and b, and stored in A and B. Next, the matrix H is computed via np.dot(A, B.T). Next, the matrix G is computed
G =
[trace(H), Δ^T]
[Δ, H + H^T - trace(H)I]

where Δ^T = [$H_{2,3}$ - $H_{3,2}$, $H_{3,1}$ - $H_{1,3}$, $H_{1,2}$ - $H_{2,1}$]

trace(H) was computed via np.trace(H). Note, I in this matrix was computed via np.eye(3) for numpy compatibility.

Next, the eigenvalue decomposition of G is computed via np.linalg.eig(G), with outputs l and Q corresponding to the eigenvalues and eigenvectors of G, respectively. The index qi of the largest eigenvalue of G in l is then computed via np.argmax(l), and then the largest eigenvector in Q is found via q = Q[:, qi]. This largest eigenvalue q is a unit quaternion corresponding to the rotation matrix. The rotation matrix can be computed via the components of q. The method returns a frame corresponding to the transformation from a to b.

pivotCalibration(frames)

Function used to perform sphere-fitting pivot calibration. Takes parameter frames, the N X 4 X 4 ndarray of frames used to perform the calibration. Function returns a list of the following:
ptr_o- 3-Dimensional pointer offset 1 X 3 ndarray
ptr_p- coordinates of the pivot point 1 X 3 ndarray
err- RMS error about centroid of pivot

read_calbody(name)

Function to read the data file for registration point locations.Takes parameter name corresponding to the name of the data set being read.Function return a list consisting of the following:
d_coords - coordinates of d N_D X 3 ndarray
a_coords - coordinates of a N_A X 3 ndarray
c_coords - coordinates of c N_C X 3 ndarray

read_calreadings(name)

Function to read the data file for calibration readings.Takes parameter name corresponding to the name of the data set being read. Function return a list consisting of the following:
D_coords - coordinates of D, F X N_D X 3 ndarray
A_coords - coordinates of A, F X N_A X 3 ndarray
C_coords - coordinates of C, F X N_C X 3 ndarray

read_empivot(name)

Function to read the data file for EM probe. Takes parameter name corresponding to the name of the data set being read. Function return a list consisting of the following:
G_coords - coordinates of G, F X N X 3 ndarray.

read_optpivot(name)

Function to read the data file for optical probe calibration. Takes parameter name corresponding to the name of the data set being read. Function return a list consisting of the following:
D_coords - coordinates of D, F X N X 3 ndarray
H_coords - coordinates of H, F X N X 3 ndarray

read_output(name)

Function to read the debug data set. Takes parameter name corresponding to the name of the data set being read. Function return a list consisting of the following:
em_pt - coordinates of the EM probe, 1 X 3 ndarray
opt_pt - coordinates of the optical probe, 1 X 3 ndarray
c_exp - coordinates of the expected values of C, F X N X 3 ndarray

write_output(name, em_pt, opt_pt, C_exp)

Function to write all output data to a text file. Takes parameters name, em_pt, opt_pt, and c_exp, corresponding to the name of the file, coordinates of the EM probe 1 X 3 ndarray, the coordinates of the optical probe (1 X 3 ndarray), and the coordinates of the expected values of C (F X N X 3 ndarray),respectively.

give_C_exp(name, save=False, m_pt=np.array([0,0,0]), opt_pt=np.array([0,0,0]))

Function used to compute the expected values of C. Parameters include name, save, em_pt, opt_pt, which correspond to the name of the data set being used, whether or not the output file is to be saved, coordinates of the EM probe, and coordinates of the optical probe, respectively. The function returns c_exp, the coordinates of the expected values of C.

em_pivot_calibration(name)

Function to apply EM tracking data to perform a pivot calibration for the EM tracking probe. Takes parameter name, the name of data set being used and returns em_pts - coordinates of EM probe wr/ EM tracker

op_pivot_calibration(name)

Function to apply optical tracking data to perform a pivot calibration for the EM tracking probe. Takes parameter name, the name of data set being used and returns op_pts - coordinates of optical probe wr/ optical tracker.

Validation approach

We implemented and conducted unit testing for each component of our program. We chose to organize our unit tests in code cells directly beneath those cells corresponding to the subject of each respective test for convenience, display, and efficiency. These unit tests include:

-Rotation test
-Transform test
-Transformation is 90 degree rot over x and +1 translation on z test
-Test module for 3D registration with two sets of points to check 3D-3D registration

-pivotCalibration

We ultimately tested our pivotCalibration method by plotting the error rate between generated calibration output and expected output for various pre-specified coordinate threshold values between 0 and 3.5 coordinate units, determining at which of these thresholds does the accuracy of our pivot calibration exceed 99.9%.

-test_Similarity(true, comp, error)

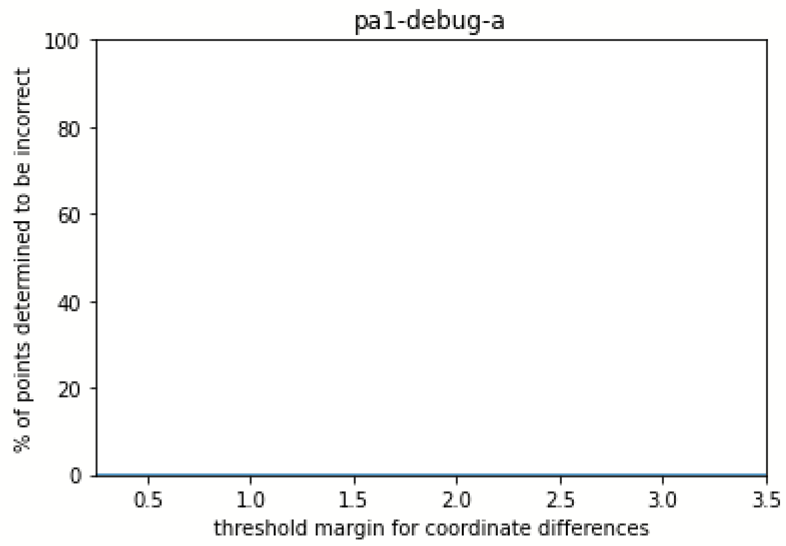
Function to test similarity between given output file and computed output values. Takes parameters true, comp, and err. Parameter true is a list of EM and optical coordinates and respective C_exp values for all frames and markers from given output. Parameter comp is a list of EM and optical coordinates as well as C_exp values computed for all frames and markers and outputted by our program. Parameter error is the threshold representing the maximum difference between any coordinate true value and our computed value for computed output to be considered correct. The function returns the % accuracy of correct.

-resultsTables(debug)

Function generates content described below and takes debug data as input.

Discussion of Results

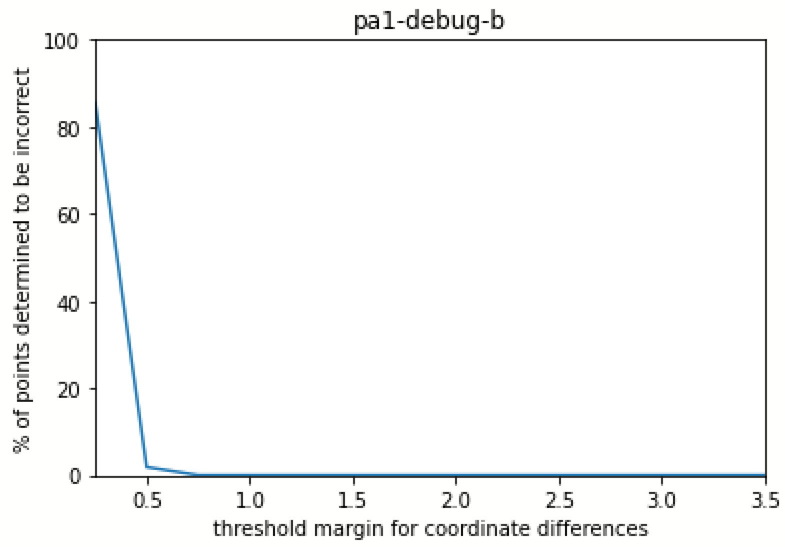
We were able to verify that individual components of our program work as expected via unit testing, as discussed previously. To evaluate the overall performance of our program, we ran a suite of test_Similarity() tests for every debugging data set provided for programming assignment 1, for multiple values of the error parameter, which represents some threshold value for computed coordinates to be considered accurate. This test_Similarity suite can be found in Main Module, which essentially executes EM and Optical probe pivot calibration on the data, computes C_expected from the calibrated data via give_C_exp(), generates an output file, reads the output file, and runs test_Similarity() between each debugging output set and the computed output file. In addition, Main Module also generates output for the unknown data sets, but this output has no reference to test similarity. Below are the results for each debug set for varying values of error threshold. Each debug set has varying levels of EM noise, EM distortion, and OT ‘jiggle’.



EM noise = False

EM distortion = False

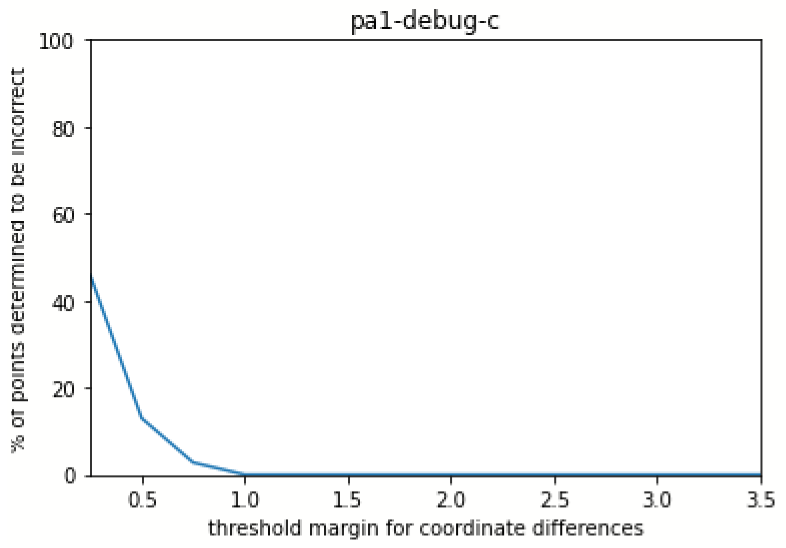
OT ‘jiggle’ = False



EM noise = True

EM distortion = False

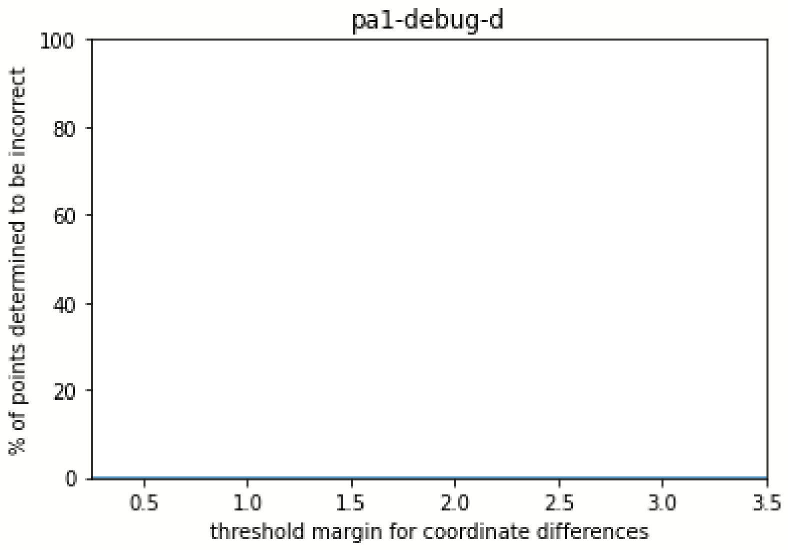
OT ‘jiggle’ = False



EM noise = False

EM distortion = True

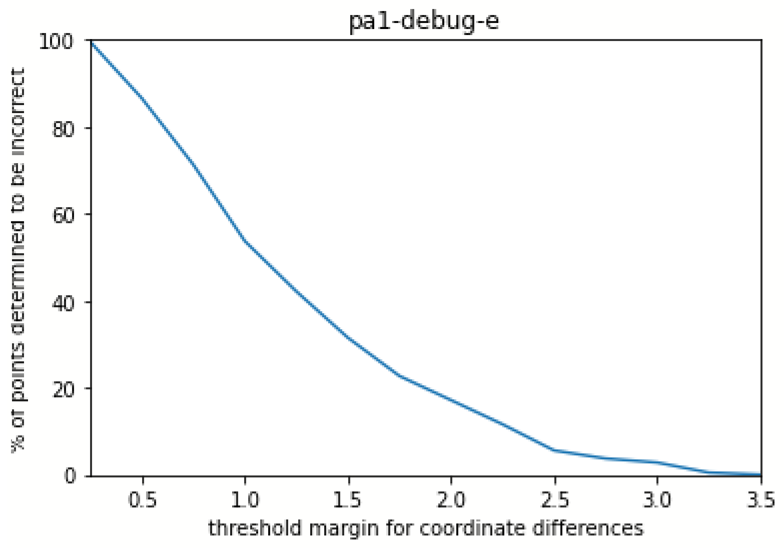
OT ‘jiggle’ = False



EM noise = False

EM distortion = False

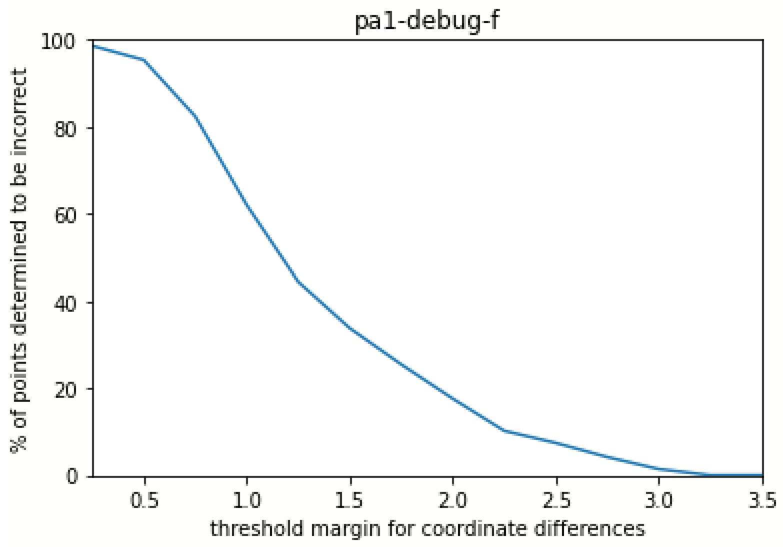
OT ‘jiggle’ =True



EM noise = False

EM distortion = True

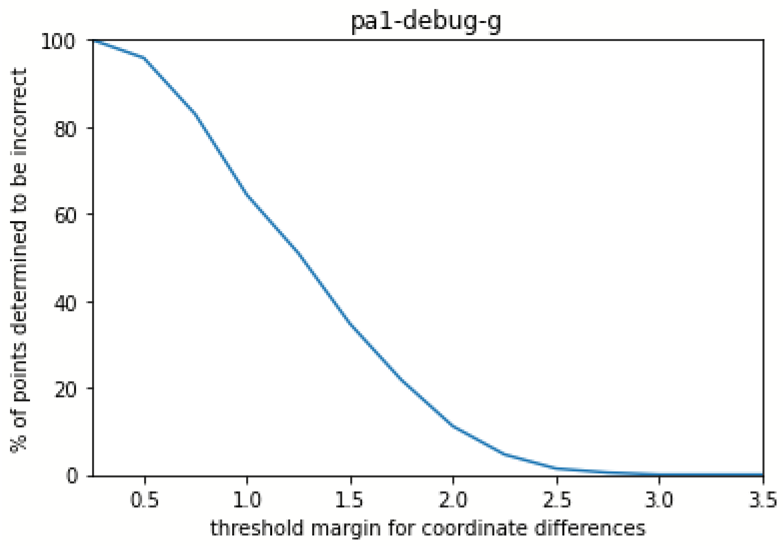
OT ‘jiggle’ = True



EM noise = True

EM distortion = True

OT ‘jiggle’ =True



EM noise = True
EM distortion = True
OT ‘jiggle’ = True

We wanted to find a threshold margin such that each set of debugging data had an accuracy rate greater than 99.9%, or an error rate less than 0.1% to determine the optimal error margin of error for our data.

The optimal values for each debug set are below:

pa1-debug-a	< 0.25
pa1-debug-b	0.75
pa1-debug-c	1
pa1-debug-d	< 0.25
pa1-debug-e	3.25
pa1-debug-f	3.25
pa1-debug-g	2.75

Thus a margin of 3.25 coordinate units is the accepted error in our calibration that returns an accuracy greater than 99.9% for all combinations of EM noise, EM distortion, and OT “jiggle”.

References

Ma, B., Banihaveb, N., Choi, J., Chen, E. C., & Simpson, A. L. (2017, March). Is pose-based pivot calibration superior to sphere fitting?. In *Medical Imaging 2017: Image-Guided Procedures, Robotic Interventions, and Modeling* (Vol. 10135, p. 101351U). International Society for Optics and Photonics.

K. Arun, et. al., IEEE PAMI, Vol 9, no 5, pp 698-700, Sept 1987