# CIS Programming Assignment 2 Report

Arijit Nukala and Ilana Chalom, anukala1, ichalom1

October 2022

## 1 Cartesian Math Package

For our program, we used native Python functions and various libraries—namely Numpy, Scipy, and Pandas— to help perform correction distortions, vector/matrix mathematics, frame transformations, rotations, and 3D point cloud representations. [3], [4], [5]. For questions 1, 2, 3, 4, 5, and 6, we were able to use a combination of these libraries, the mathematical functions we wrote in PA1, and our newly written functions to compute the answers. We relied on the homework instructions and the ways we learned to solve these problems in class, and applied this understanding to these questions, only this time, using our program to do the mathematical computations for us. For instance, by writing more foundational functions, such as scaleToBox(), we were able to use these as building blocks, both within our more complex algorithmic functions, and in producing a distortion correction.

## 2 Algorithmic and Mathematical Approach

### 2.1 Distortion Correction

For our distortion correction, we used the 5th degree Bernstein Polynomials we learned from the interpolation in class to produce a distortion correction function given a pointset. The algorithmic approach to this looked something like:

1. Given a vector q, find the upper and lower limits so normalize our values from 0 to 1
   $q_{max}$ and $q_{min}$
2. Scale the vector to the box of these limits to produce a vector u
3. Create a Bernstein polynomial matrix:
   $F_{ijk}(u_x, u_y, u_z) = B_{5,i}(u_x)B_{5,j}(u_y)B_{5,k}(u_z)$
4. Using this matrix, perform a least squares operation to solve for coefficient matrix c
5. Apply this matrix c to corrected vector u to perform the distortion correction

Reflected in our code, these steps looked like:

1.
```
q_min, q_max = min_max(c)
q_exp_min, q_exp_max = min_max(c_exp)
```

   where min_max(q) performs

```
def min_max(q):
  q_max = np.amax(q, axis=0)
  q_min = np.amin(q, axis=0)
return q_min, q_max
```

2.
```
u = scaleToBox(q, q_min, q_max)
u_exp = scale_to_box(c_exp, q_exp_min, q_exp_max)
```

   where scale to box looks like

```
def scale_to_box(q, q_min, q_max):
   box_frame = (q - q_min) / (q_max - q_min)
return box_frame
```

3. Finding the coefficient matrix, done in bernstein_c_ij, finds matrix F_ijk like

```
for mat_i in range(len(u)):
 mat_jk = 0
 for i in range(deg + 1):
     for j in range(deg + 1):
         for k in range(deg + 1):
             F_ijk[mat_i][mat_jk] = (bernstein(deg, i, u_exp[mat_i][0]) *
             bernstein(deg, j, u_exp[mat_i][1]) * \
                                        bernstein(deg, k, u_exp[mat_i][2]))
             mat_jk += 1
```

where bernstein performs

```
def bernstein(N, k, u):
  v = 1 - u
return (np.math.factorial(N) /
(np.math.factorial(k) * np.math.factorial(N - k)) * (u ** k) * (v ** (N - k)))
```

4. using F_ijk, perform least squares operation to find coefficient matrix

```
return np.linalg.lstsq(F_ijk, u, rcond=None)[0], q_min, q_max, q_exp_min, q_exp_max
```

5. Apply the coefficient matrix to the points in matrix *corrected*

```
for i in range(len(u)):
for j in range(deg + 1):
    for k in range(deg + 1):
        for l in range(deg + 1):
            corrected[i] += coefficient[j * (deg + 1) ** 2 + k * (deg + 1) + l] * \
                            bernstein(deg, j, u[i][0]) * bernstein(deg, k, u[i][1])
                        * bernstein(deg, l, u[i][2])
corrected = unscale(corrected, q_exp_min, q_exp_max)
```

where unscale rescales back to the original:

```
def unscale(q, q_min, q_max):
  scale_up = q * (q_max - q_min) + q_min
return scale_up
```

Mathematically speaking, following these same steps the program computes something like:

1. Find the minimum and maximum limits for q: $q^{min}$ and $q^{max}$
2. Find a scaled vector u using q, $q^{min}$, and $q^{max}$

$$u = \frac{q - q^{min}}{q^{max} - q^{min}}$$

3. create matrix F_ijk matrix of Bernstein polynomials to use to solve for coefficient matrix

$$\mathbf{F}_{ijk}(u_x, u_y, u_z) = B_{5,i}(u_x)B_{5,j}(u_y)B_{5,k}(u_z)$$

where $B_{N,k}(u)$ can be found by:

$$B_{N,k}(u) = \binom{N}{k}(1-u)^{N-k}u^k$$

4. set up the least squares problem
$$\begin{bmatrix} & \vdots & \\ p_s^x & p_s^y & p_s^z \\ & \vdots & \end{bmatrix} \approx \begin{bmatrix} & \vdots & \\ F_{0,0,0}(u_s) & \cdots & F_{5,5,5}(u_s) \\ & \vdots & \end{bmatrix} \begin{bmatrix} c_{0,0,0}^x & c_{0,0,0}^y & c_{0,0,0}^z \\ \vdots & \vdots & \vdots \\ c_{5,5,5}^x & c_{5,5,5}^y & c_{5,5,5}^z \end{bmatrix}$$
and use the following expression to solve for c
$\mathbf{F}(\mathbf{u})^T\mathbf{F}(\mathbf{u})^{-1}\mathbf{F}(\mathbf{u})^T\vec{p}$

5. use the coefficient matrix c to perform the correction for vector u (the scaled vector q)
$\sum_{i=0}^{5}\sum_{j=0}^{5}\sum_{k=0}^{5}\mathbf{c}_{i,j,k}B_{5,i}(u_x)B_{5,j}(u_y)B_{5,k}(u_z)$

### 2.2 Problem 1

Problem 1 of this assignment is to use the body calibration data file and find $C_i^{(expected)}[k]$ for each $C_i[k]$ in each frame $k$ of data. The approach for this looks like:

1. Find $F_D$ such that:
   $D_j = F_D \cdot d_j$
2. Find $F_A$ such that:
   $A_j = F_A \cdot a_j$
3. Find $C_i^{expected}$ such that:
   $C_i^{expected} = F_D^{-1} \cdot F_A \cdot c_i$

This is reflected in the code by

1.
```
FD = registration(d_points.points, D_points[frame].points)
```

2.
```
FA = registration(a_points.points, A_points[frame].points)
```

3.
```
NF = FD.invert().compose_frame(FA)
Ci = PointCloud(NF.compose_transform(c_points.points))
```

### 2.3 Problem 3

Problem 3 performs the distortion correction on the pivot calibration for the EM probe. The steps are roughly as follows:

1. Use problem one two compute $c_{expected}$
2. Compute the bernstein coefficient matrix to perform the distortion correction on the distorted vectors G
3. Recompute the pivot calibration for the undistorted points as described in section 2.8.

### 2.4 Problem 4

Problem 4 uses the improved value from problem 3 to locate b_j of the fiducial points. To do this, we:

1. Perform the distortion correction on the EM_Fiducial data.
2. Center the improved undistorted pivot data
3. Iterate through the undistorted fiducial points and compute the registration (as described in section 2.7) with the centered pivot data
4. Calculate $b_j$ by transforming the matrix we just registered with the $p_{tip}$ vector found from problem 3

### 2.5 Problem 5

Problem 5 computes the frame registration for $F_{reg}$ by

1. Using the ct-fiducial data and the $b_j$ vector found in problem 4, compute the frame registration described in section 2.7.

### 2.6 Problem 6

Problem 6 locates the tip with respect to the CT image. This is done by

1. Perform the distortion correction on the EM_Pivot data.
2. Center the points
3. Iterate through the undistorted EM_Pivot points and compute the registration (as described in section 2.7) with the centered pivot data
4. Calculate the vector from the transformation of the matrix we just registered with the $p_{tip}$ vector found from problem 3 and now use this vector to transform the matrix found in problem 5.

### 2.7 Registration algorithm

For our registration algorithm, we used an approach developed by Arun et al., in which the authors explored a non-iterative algorithm which employs Singular Value Decomposition. Additionally, we compensated for rotation matrices with a determinant of negative one using an approach described by Sorkine-Hornung et al. The algorithmic approach using can be broken down into roughly five or so steps:

1. Center the point sets
2. Calculate a matrix H by multiplying one matrix by the transpose of the other
3. Find the SVD of H
4. Use the results of the SVD to calculate a matrix X
5. Calculate the determinant of X: it should be 1, or correct the matrix if the determinant is negative one

Reflected in the code, this looked like:

1.
```
a_mean = np.mean(A.points, axis=1, keepdims=True)
b_mean = np.mean(B.points, axis=1, keepdims=True)
centered_a = A.points - a_mean
centered_b = B.points - b_mean
```

2.
```
H = np.dot(centered_a, centered_b.transpose())
```

3.
```
u, s, vt = np.linalg.svd(H)
```

4.
```
 u = u.transpose()
 vt = vt.transpose()
```

5.
```
comp_size = vt.shape[0]
reflection_comp = np.eye(comp_size)
reflection_comp[comp_size - 1][comp_size - 1] = np.linalg.det(np.dot(vt, u))
```

6. And with the newly found rotation matrix, we returned a Frame:

```
R = np.dot(vt, np.dot(reflection_comp, u))
p = b_mean - np.dot(R, a_mean)
return Frame(R, p)
```

Mathematically speaking, the function would look like

1. $a_i = a - \bar{a}$
   $b_i = b - \bar{b}$
2. $H = \sum_{i=1}^{N} a_i b_i^T$
3. $H = U\Sigma V^T$
4. Steps 4 and 5 of the algorithm can be combined into:
   $X = VU^T$
   where $X$ is our Rotation matrix, assuming the determinant is 1

### 2.8 Pivot Calibration

For the pivot calibration algorithm, we used Ziv Yaniv's paper titled "Which Pivot Calibration?". We used the following steps to solve for the pivot calibration:

1. Get the mean of the points in the first frame and use this to define a coordinate system for the probe
2. Subtract the mean from the rest of the points to convert these points into the frame of the probe
3. Solve the overdetermined system described in the paper using the Algebraic One Step (ATS) method. This starts by computing the registration for each frame k of pivot data and filling out the least squares matrices for rotation and translation
4. compute the least squares of these least squares matrices to find the matrix *pivot_cal*
5. use *pivot_cal* to find $p_t$ and $p_{pivot}$

Algorithmically, this looks like:

1. Step 1 and 2 from the algorithm are combined here into:

```
local_probe = ps_data[0].points
 g_j = local_probe - np.mean(local_probe, axis=1)[0]
```

2.
```
for i in range(num_frames):
    frame_k = registration(ps_data[i].points, g_j)
    cur_r, cur_p = frame_k.R, frame_k.p
    ti = 3 * i
    lstsq_r[ti:ti + 3, 0:3] = cur_r
    lstsq_p[ti:ti + 3] = -1 * cur_p.reshape(3, 1)
    lstsq_r[ti][3] = -1
    lstsq_r[ti + 1][4] = -1
    lstsq_r[ti + 2][5] = -1
```

3.
```
pivot_cal = np.linalg.lstsq(lstsq_r, lstsq_p, rcond=None)[0][0:6]
```

4.
```
tip_in_tool = pivot_cal[0:3].reshape(3,)
tool_in_base = pivot_cal[3:6].reshape(3,)
```

Mathematically, we see this as:

1.
$$G'_0 = \frac{1}{N_G} \sum G'_j$$

2.
$$g_j = G'_j - G'_0$$

3. For each frame $k$ of the pivot data, compute
$$G_j = F_G[k] \cdot g_j$$

4. Solve for
$$P_{dimple} = F_G[k] \cdot t_G$$

where $t_G$ is defined by the tip coordinates in the same probe coordinate system

### 2.9 Transformation

Our frame transformation function looked something like:

1. Iterate through each row in the rotation matrix of the frame, R, and compute the dot product of that vector with the points to transform and add to it the translation vector, p.

Reflected in the code, this likes like:

1.
```
for i in range(frame_size):
    t_points[i] = np.dot(self.R, points[i]) + self.p
```

Mathematically, this looks like:

1. $v = F \cdot b$

   $v = [R, p] \cdot b$

   $v = R \cdot b + p$

### 2.10 Composition

Our frame composition function looked something like:

1. Compute the dot product between Frame 1's rotation Matrix and Frame 2's rotation matrix
2. Compute the dot product between Frame 1's rotation Matrix and Frame 2's translation vector. Add to it Frame 1's translation vector

This was represented in the code with the lines:

1.
```
mat = np.dot(self.R, other_frame.R)
```

2.
```
vec = np.dot(self.R, other_frame.p) + self.p
```

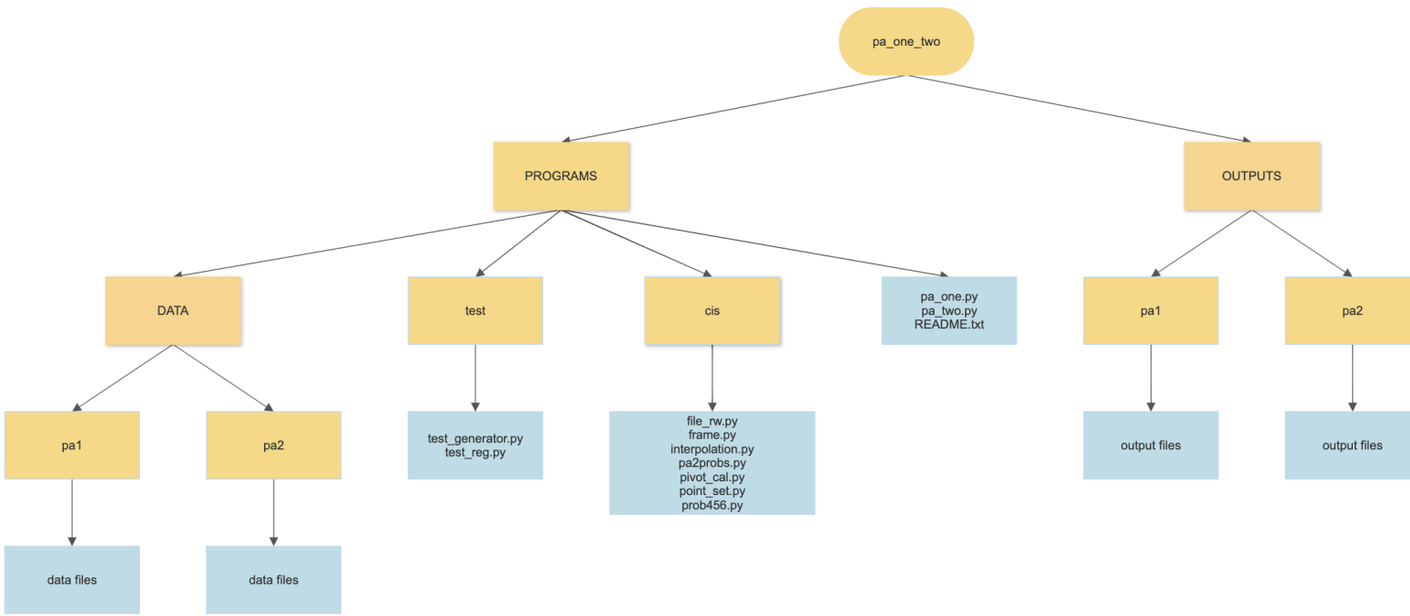
Mathematically speaking, this looks like:

1. $F_1 \cdot F_2 = [R_1, p_1] \cdot [R_2, p_2]$

   $F_1 \cdot F_2 = [R_1 \cdot R_2, R_1 p_2 + p_1]$

## 3 Program Structure

Our program structure is broken down into two main sections, programs/inputs, located in the PROGRAMS folder, and outputs, located in the OUTPUTS folder. Inside PROGRAMS, is our README.txt, and main method for programming assignment 1 and 2, in pa_one.py and pa_two.py respectively. We also have a folder containing all of the input files, called DATA, and the testing files, called test. Lastly, in PROGRAMS there is a folder titled cis; this is where the bulk of our program lies. file_rw.py, frame.py, pivot_cal.py, point_set.py, and prob456.py are all files from programming assignment 1. The overall structure of our program can be seen in the diagram below, and is further elaborated following the diagram.


pa_one_two
PROGRAMS
OUTPUTS
DATA
test
cis
pa_one.py
pa_two.py
README.txt
pa1
pa2
pa1
pa2
test_generator.py
test_reg.py
file_rw.py
frame.py
interpolation.py
pa2probs.py
pivot_cal.py
point_set.py
prob456.py
output files
output files
data files
data files

### 3.1 file_rw.py

This file was initially created for programming assignment 1, but has expanded for this assignment, and contains the methods to read in the data from the different types of input files and return the corresponding point clouds. The functions include getDataCalBody, getDataCalReading, getDataEMPivot, getDataOptPivot, getDataCTFids, getDataEMFids, getDataEMNav. These files all have a very similar process of traversing the input files and filtering through the data to represent the point clouds.

### 3.2 frame.py

this file was created in programming assignment 1 and contains the frame class. The functions of the class are compose_frame, compose_transform, and invert.

1. compose_frame is called on a frame and takes in another frame as the additional argument. It then computes a frame composition and returns the final frame.

2. compose_transform is also called on a frame and takes in a point set as the additional argument. The function computes a frame transform and returns the corresponding points.

3. invert is called on a frame and performs the frame inversion calculations. The final resulting frame is returned.

### 3.3 interpolation.py

This file contains the bulk of our new code for programming assignment two. We have our functions remove_distortion, unscale, scale_to_box, bernstein, and bernstein_c_ij.

1. The remove_distortion function takes in matrix c, matrix coefficient, and the degree for the Bernstein polynomial later on. It calls the function scale_to_box on the points of c to find matrix u and vectors q_min and q_max. Then, it iterates through coefficients, the Bernstein matrix, to correct each point. Lastly, it calls unscale to finish the correction by rescaling the points back their original state.

2. The bernstein_c_ij function takes in the matrix for c, c_exp, and the degree for the Bernstein polynomial. This function calls scale_to_box and uses the result to fill in the matrix, calling the bernstein function to compute each point. It finishes by computing the least squares with F_ijk and u.

3. The function bernstein takes in a degree number N, index k, and matrix u. It then computes the Bernstein polynomial for the given degree, index, and matrix the same way as we studied in class, and detailed in section 2 of this report.

4. The function scale_to_box takes in a vector q, and computes the q_min, q_max. It then computes matrix box_frame by computing the same equation studied in class, and detailed in section 2 of this report.

5. Lastly, the function unscale takes in q, q_min, and q_max and unscales the matrix such that it returns to its original state before it was scaled.

### 3.4 pa2probs.py

This file contains the functions to solve questions 1, 4, 5, 6 (called prob_one, prob_four, prob_five, and prob_six respectively) for programming assignment 2.

1. prob_one is the function to answer question one of the assignment, and takes in the file location for cal_body and cal_reading. It calls getDataCalBody and getDataCalReading to obtain the data from these files. It then calls registration, composeFrame, composeTransform to properly compute the values for c_expected, and it employs the PointSet constructor to cast each row in the final matrix.

2. prob_four is the function that really computes the answers to question 3 and question 4 of the assignment. It takes in the arguments for the file locations for cal_body, cal_reading, em_pivot, and em_fids. The first step in this function is to call getDataEMFids, getDataEM-Pivot, and getDataCalReading to read the files and obtain their data. Then, prob_one is called to compute the c_expected and distortion and correct_distortion are called to perform the distortion correction for the pivot calibration of the EM probe. Lastly, registration and compose_transform are called to properly compute the location of b.

3. prob_five is the function that computes the frame registration for F_reg. It takes in the file location for ct_fids and the vector b. It first calls getDataCTFids to obtain the data from the file, and then calls registration to perform the registration of the ct_fids data and the b vector.

4. prob_six is the function that computes the pointer tip coordinates with respect to the CT coordinates. It calls getDataEMPivot to get the em_nav data. It then calls remove_distortion to perform the distortion correction on em_nav using the coefficients matrix found earlier. It then calls registration and compose_transform to properly compute the location of the tip, b_nav.

### 3.5 pivot_cal.py

This file, created for programming assignment 1, includes the pivot calibration algorithm in the function titled pivot.

### 3.6 point_set.py

This file, also created for programming assignment 1, contains the pointSet class which we use to represent point clouds. Inside this class is the registration function for our registration algorithm from assignment 1.

### 3.7 prob456.py

This file contains our answers to problems 4, 5, and 6 from programming assignment 1.

## 4 Verification

To verify our program, we produced unit tests to test basic algorithms and assume that the methods used in PA1 are correct given our tests for that program. Our approach for each question was to use solutions we found in PA1. Our testing files can be found under PROGRAMS/tests. To run unit tests, we tested the basic operation of each tested method.

To verify our implementation of the Bernstein coefficient calculator, we manually calculated the polynomial for a series of input values for N, k, and u and compared our output to our calculations.

To verify our min and max vector computer, we plugged in an array of values, manually computed the minimum and maximum values along the column axis, and compared our output to our result.

Finally, to test our scaling method, we once again plugged in an array of values and manually computed the scaled values, and compared the output to the result.

To evaluate our performance on the debug sets, we produced a testing script to compute the RMSE between our outputs and the debug outputs. We evaluated our script on the debug a-f values using conftest.py, a method that allows pytest to take in command line inputs. We inputted the file names for testing. Our testing script produced the RMSE for our $C_i^{(expected)}$ and the debug data's $C_i^{(expected)}$ and for each $\vec{v}_j$ in the output-2 files.

# 5 Results

All of our results can be found in the OUTPUTS directory in the subdirectory pa2. An overview of our results are as follows:

## 5.1 pa2-unknown-g-output-1.txt

| | | |
|---|---|---|
| 212.70, | 213.89, | 205.77 |
| 388.64, | 390.90, | 205.67 |
| 98.43, | 100.55, | 99.35 |
| 98.05, | 100.19, | 224.35 |
| ⋮ | ⋮ | ⋮ |

## 5.2 pa2-unknown-g-output-2.txt

| | | |
|---|---|---|
| 573.58, | 125.57, | 263.81 |
| -89.85, | -359.80, | -75.13 |
| -100.66, | 26.39, | -223.45 |
| 132.56, | -448.27, | -157.47 |

## 5.3 pa2-unknown-h-output-1.txt

| | | |
|---|---|---|
| 208.68, | 206.87, | 210.49 |
| 405.23, | 395.64, | 203.53 |
| 99.58, | 101.37, | 100.80 |
| 100.40, | 103.17, | 225.78 |
| ⋮ | ⋮ | ⋮ |

## 5.4 pa2-unknown-h-output-2.txt

| | | |
|---|---|---|
| -82.10, | 45.83, | 453.78 |
| 407.53, | 470.18, | 287.80 |
| 317.81, | -232.90, | 202.19 |
| -94.54, | 422.08, | 273.61 |

## 5.5 pa2-unknown-i-output-1.txt

| | | |
|---|---|---|
| 207.07, | 204.84, | 195.20 |
| 407.99, | 386.12, | 202.31 |
| 98.88, | 101.23, | 101.12 |
| 97.04, | 102.57, | 226.10 |
| ⋮ | ⋮ | ⋮ |

### 5.6 pa2-unknown-i-output-2.txt

| | | |
|---|---|---|
| -148.90, | 88.84, | -330.27 |
| 138.03, | 365.25, | -57.73 |
| 362.94, | -331.53, | -214.61 |
| 386.76, | 227.19, | -56.47 |

### 5.7 pa2-unknown-j-output-1.txt

| | | |
|---|---|---|
| 201.22, | 205.54, | 208.29 |
| 407.50, | 376.84, | 206.29 |
| 101.75, | 101.33, | 100.97 |
| 103.00, | 99.46, | 225.95 |
| ⋮ | ⋮ | ⋮ |

### 5.8 pa2-unknown-j-output-2.txt

| | | |
|---|---|---|
| 59.48, | 34.97, | -77.95 |
| -15.28, | 70.07, | 274.08 |
| 263.65, | -102.28, | 324.28 |
| -41.48, | 414.75, | -8.21 |

## 6 Discussion

What we found was that our RMSE between $C_i^{(expected)}$ and the debug data's $C_i^{(expected)}$ was rather small. However, our output-2 data was off. While we attempted to locate the issue within each problem method, we were unable to determine the root cause of the issue in time. Our suspicion is that our error stems from the prob_four method. When printing our frame transform between the $g_j$ residual values for the em_pivot data and the em fiducial data, our p vector for the frame transform seemed rather large. Despite efforts to correct the issue, we were unable to fully resolve the error and as a result, have incorrect data for our output-2. Given more time, we would produce unit testing for our problem-solving methods. By isolating the issue to each component, we might be able to better determine where our error occurs and eliminate the problem.

Additionally, we noticed that some of our values in the output-2 data were negative when they should have been positive. We surmise that in undistorting our data and applying the registrations, we may have applied the incorrect data for the registrations, causing transforms that locate the tip location values at the wrong locations. We may be able to solve this by isolating the registration portions of our problem-solving methods and testing them thoroughly to ensure the proper registration is computed. However, without intermediate data to test against, we found it difficult to locate the problem precisely.

## 7 Contributions

Ilana: helped write the distortion function, F function, and scaleToBox function in interpolation.py. Also wrote prob_one (or q4 of PA1) in pa2probs.py. Wrote the report, and made the diagrams and equations for the report. (From PA1: Frame.py and point_set.py, wrote getDataCalBody, getDataCalReading, getDataEmpivot, getOptpivot, and prob_four of prob456.py)

Arijit: helped write the distortion function, F function, and the rest of interpolation.py. Also wrote

prob_four, and prob_five in pa2probs.py, and wrote the new functions for file reading, getDataCTFids, getDataEMFids. Worked on debugging and testing. (From PA1: pa_one.py, pivot_cal.py, prob_five and prob_six of prob456.py)

## 8 Sources


1. Arun, K. S., et al. Least Squares Fitting of Two 3-D Point Sets. IEEE, Sept. 1997, www.ece.queensu.ca/people/S-D-Blostein/papers/PAMI-3DLS-1987.pdf.

2. Chintalapani, G., Taylor, R. H. (2007). C-arm distortion correction using patient CT as a fiducial. In 2007 4th IEEE International Symposium on Biomedical Imaging: From Nano to Macro - Proceedings (pp. 1180-1183). [4193502] (2007 4th IEEE International Symposium on Biomedical Imaging: From Nano to Macro - Proceedings). https://doi.org/10.1109/ISBI.2007.357068

3. Florian, et al. "Pytest-Dev/Pytest." GitHub, 13 Oct. 2022, github.com/pytest-dev/pytest/blob/main/CITATION. Accessed 13 Oct. 2022.

4. Gennery, Donald B. "Least-Squares Camera Calibration Including Lens Distortion and Automatic Editing of Calibration Points." California Institute of Technology, 1998.

5. Harris, Charles R., et al. "Array Programming with NumPy." Nature, vol. 585, no. 7825, 16 Sept. 2020, pp. 357–362, 10.1038/s41586-020-2649-2.

6. "Logging — Logging Facility for Python — Python 3.10.0 Documentation." Docs.python.org, docs.python.org/3/library/logging.html.

7. McKinney, Wes. "Data Structures for Statistical Computing in Python." PROC. OF the 9th PYTHON in SCIENCE CONF, 2010.

8. "Point-Set Registration." Wikipedia, 2 July 2022, en.wikipedia.org/wiki/Point-set_registration. Accessed 13 Oct. 2022.

9. Sorkine-Hornung, Olga, and Michael Rabinovich. Least-Squares Rigid Motion Using SVD. 16 Jan. 2017, igl.ethz.ch/projects/ARAP/svd_rot.pdf. Accessed 13 Oct. 2022.

10. Virtanen, Pauli, et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python." Nature Methods, vol. 17, no. 3, 3 Feb. 2020, pp. 261–272, 10.1038/s41592-019-0686-2.

11. "VPI - Vision Programming Interface: Lens Distortion Correction." Docs.nvidia.com, docs.nvidia.com/vpi/algo_ldc.html. Accessed 27 Oct. 2022.

12. "Welcome to Click — Click Documentation (8.1.x)." Click.palletsprojects.com, click.palletsprojects.com/en/8.1.x/.

13. Yaniv, Ziv. "Which Pivot Calibration?" SPIE Proceedings, 18 Mar. 2015, www.yanivresearch.info/writtenMaterial/pivotCalib-SPIE2015.pdf, 10.1117/12.2081348. Accessed 13 Oct. 2022.