The main problem that we aimed to tackle in this first assignment using Python3 was the development of a library of mathematical tools and subroutines relevant to computer-integrated surgery that we could call upon when needed such as for application in future assignments. Such mathematical principles are the foundation for how modern systems work and so with this in mind we set forth to tackle the development of our math package.

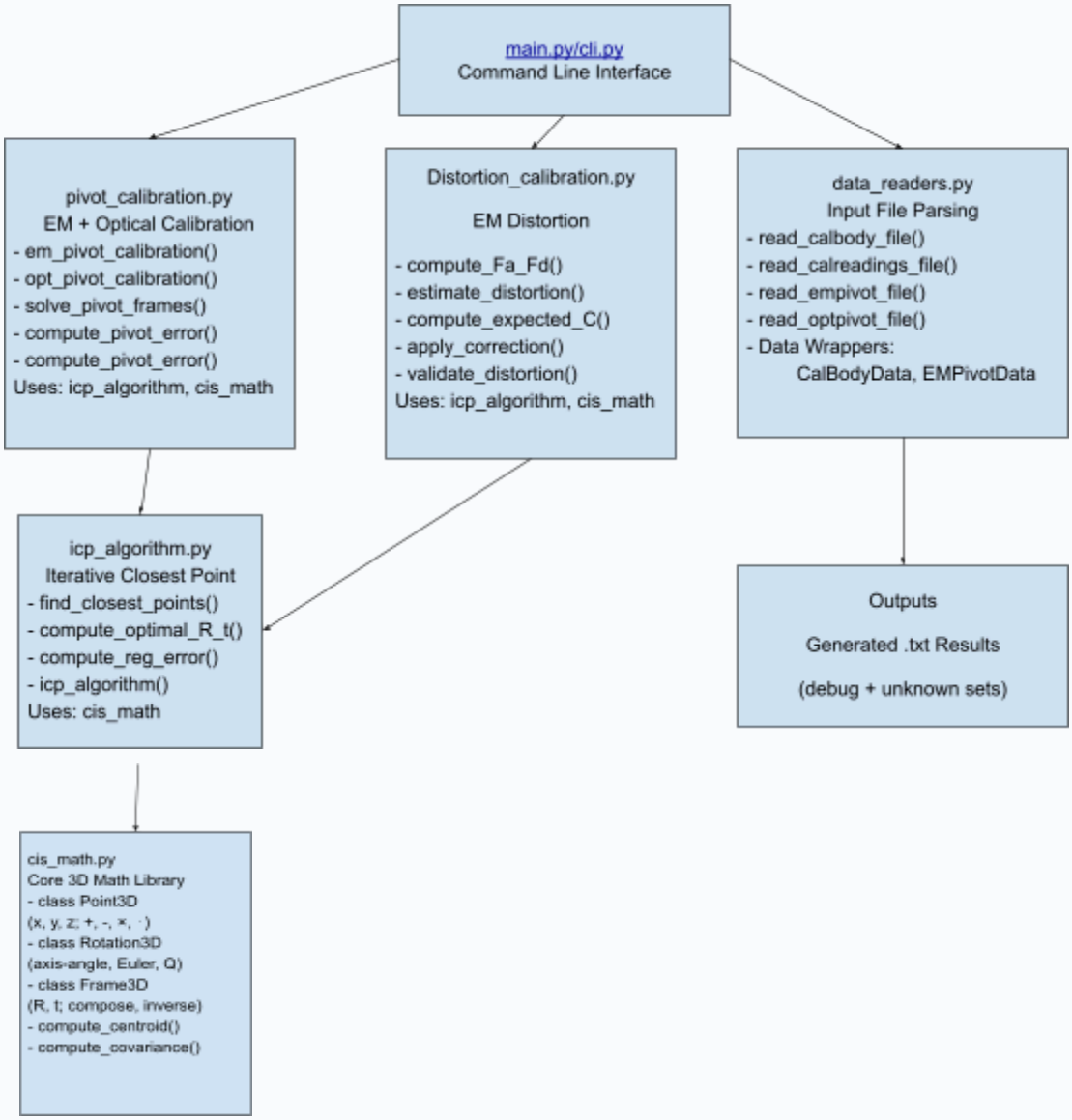

Figure 1: Program Hierarchy Chart

One of the most basic initial steps we had to complete was the development of a 3-D mathematics package for various common 3D transformations that could occur. After establishing file input and output we set forth. The cis_math.py file defines our mathematical foundation that we would use throughout all of Programming Assignment 1. This package supports simple vector arithmetic, rotations, and frame transformations thus providing a geometric backbone.

1. The first core class that we defined was the Point3D class. This was nothing but the representation of a vector. It also contained support for standard functions executed upon a vector such as +, - , *, dot product($\cdot$), cross product($\times$), magnitude (norm) and normalization calculation.
2. Now that we had vectors we defined the Rotation3D class for encapsulating a 3x3 rotation matrix (given we are in 3 dimensions of course). This class provided support for the creation from an axis-angle representation through Rodrigues' formula ($R = I + sin(\theta)K + (1 - cos(\theta))K^2$ for which computation of the skew-symmetric matrix is quite useful as a function), from Euler angles, and from quaternions which were all forms of representation learned from the material class that we thought would be best to support. We also had some classic user functions to help apply rotations to 3D points like apply() which rotates a point about another point, individual axes rotations, inverse() which find the inverse rotation through matrix transposition, compose() which chains rotations, determinant() which verifies orthonormality, and to_axis_angle() which converts the rotation matrix to an axis-angle representation. Again these were nothing but standard computations about a rotation matrix and so there wasn't much mathematical approach here other than thinking about what we would possibly want to be able to do with a rotation matrix.
3. Now that these were defined we then defined the Frame3D class which combines a rotation and translation to represent a full rigid body transform:

   $$\vec{p}' = R\vec{p} + \vec{t}$$

   And so knowing this formula we supported the application of forward rigid body transforms as well as finding the inverse Rigid Body transform with:

   $$F_{AB}^{-1} = [R_{AB}^{-1}, -R_{AB}^{-1} + p_{AB}]$$

   And frame composition which allows me to represent multiple rigid body transforms in one Frame3D through:

   $$F_{AC} = F_{AB}F_{BC} = [R_{AB}R_{BC}, R_{AB}p_{BC} + p_{AB}]$$

Now before discussion of the rest of the functions contained here in cis_math.py and icp_algorithm.py it would make sense for us to explain what we are trying to accomp[lish with these two algorithms, our mathematical approach, and the algorithmic steps that followed. The premise of the first problem was 3D Point Cloud -> 3D Point Cloud registration which essentially ask the following:

Given point clouds $A = \{\vec{a}_1, \vec{a}_2 \ldots \vec{a}_n\}$ and $B = \{\vec{b}_1, \vec{b}_2 \ldots \vec{b}_n\}$, we want to find the rigid body transformation (rotation + translation) that minimizes the sum of the squares of the distances between corresponding points. Now, knowing the correspondence of points in A to points in B would make situations like this quite easy, and we did code a function for if we did have such foresight, but it is most often the case that we do not actually know which points correspond. For lack of clearer explanation, we are quite simply taking the point cloud A and trying to rotate and translate it so that it is as close to B as possible.

$$\min_{R,t} \sum_i || R\vec{a}_i + \vec{t} - \vec{b}_i ||^2$$

The algorithm for how to do this is as follows and is called the Iterative Closest Point (ICP) Algorithm which iteratively finds the best rotation and translation to apply to try and get the final frame transformation between the two point clouds right. The limit as this algorithm of finding the best frame transform applying and reanalyzing as time approaches large numbers tends to be the optimal overall frame transform needed for the point set registration problem:

1. We start by of course initializing the two steps and then we find the closest points such that for each point $\vec{a}_i$ in set A we find the closest point $\vec{b}_i$ in set B using the Euclidean distance so that we have a preliminary list of correspondences between the two sets that we can work with.
2. We then compute the centroids of the two point clouds by finding the average x, average y, and average z for all points in A and all points in B and call those $\overline{a}$ and $\overline{b}$.
3. Then we subtract that centroid from all the points in the set so that the points are on-average centered about the origin and call these new sets A' and B'.
4. Once that is complete we compute the covariance matrix between A' and B' as:

   $$H = \sum_i a'_i \{b'^T_i\}$$

   Remember that a'_i and b'^T_i represent a column vector for a specific point in 3D space that we set an association between previously. When we compute the outer product of these two matrices as we are doing in the above summation we actually get a final matrix that shows us how every axis for which we have information in a'_i relates to every axis for which we have information in b'_i. For example in the final H the H_{xx} component shows how the x-axis of a'_i relates to the x-axis of b'_iIt is just that we are doing this point by point and so the summations shows the relationship between the whole sets A' and B' and so internally this stores the directional correlation between A' and B'. So for example if A' and B' were very close to each other then the covariance matrix would more closely resemble the identity matrix.
5. Next we need to extract the rigid transformation encoded for in the covariance matrix and so for that we want the rotation R that makes $R\vec{a}'_i$ line up with b'_i and that is equivalent to maximizing $trace(RH)$ under the constraint that $R^T R = I$ and $det(R) = 1$ which indicates a non-reflective rotation . This means that the trace is the largest when we rotate A's coordinates to match B's. To actually do this we use Single Value Decomposition where we say that $H = USV^T$ where U an V are orthogonal 3x3 matrices and the S holds the singular values (non-negative scales). Then from this we compute the candidate rotation by saying $R = VU^T$ which rotates A's principal directions which are the columns of U to B's whose

principal directions are the columns of V. Then after flipping any reflection that might occur because det(R) might have been -1 we get the pure rotation we would need to align the point clouds in a least squares sense.

6. Now with the rotation figured out we just have to figure out the translation for which $\overline{a}$ needs to simply be moved to $\overline{b}$ and so we can simply take $\overline{b} - R\overline{a}$ since we want to take the post-rotation value of $\overline{a}$ and match with $\overline{b}$.

The final point to mention is that this does not work in one iteration hence the name of ICP. The ideal rotation reduces the error and then you must repeat these steps until the error settles at which point you can finally find through the composition of all the frame transformations required in the steps the final frame transformation you would need for the 3D point cloud to 3D point cloud registration.

Now with that explained it becomes easy to see why in cis_math.py we have a compute_centroid() and computer_covariance_matrix. It also becomes easy to tell what the functions in icp_algorithm.py are doing. find_closests_points() is for step 1, compute_optimal_rotation_translation() applies the SVD approach discussed before, compute_registration_error () computes that mean-squared error we are trying to minimize, and icp_algorithm() wraps this all up neatly. We also made an icp_with_known_correspondences() function in the case that we are afforded the luxury of knowing which points correspond between the two clouds.

The next task to tackle after the implementation of the 3D point set to 3D point set registration algorithm was the “pivot” calibration method. But before we dove into the mathematical/algorithmic approach we needed to understand what this problem actually was. The main aim of pivot calibration is the following:

Let's say that I have a probe that the EM or optical tracker sees using a bunch of little markers. The tracker only knows where the clusters of markers are. It does not actually know where the tip of the probe is because as rotation occurs of the whole probe the tip also is moving. So a pivot calibration finds that offset which is the vector from the cluster’s coordinate system to the actual tip. Note that we actually have two dimpled posts that we have to deal with. One gets the pivot calibration for the EM system while the other gets it for the optical system.

With the problem now clearly defined we can start curating our algorithmic approach. We first start with the optical sensor which has N_H LED markers that are tracked by the optical system. For each frame that we collect we have N_H points that we collect because we collect a data point for each LED Marker. We also have N_A optical markers on the calibration object to help set the natural coordinate frame. We essentially from these values want $p_{optical,tip}$ which using the class notation means the tips coordinates to the optical frame or optical frame from the tip.

The set of probe markers which we can denote as H_k in one frame that we capture defines the pose of the probe which in other words is its rotation and translation relative to some reference point. The way that this works is that the probe has a certain reference position and any new position can be expressed as a point cloud to point cloud transformation of the original reference frame. In this way we can find a rotation matrix (R_k) and translation (t_k) that gives us our new pose and we can repeat this for every frame we capture. The key thing to notice

however, is that while R_k and t_k are changing after every frame, the probe tip stays at the same fixed spot in the dimple. So in this way we can use the rotation and translation during each frame to express an equation.

$$R_k p_{optical,tip} + t_k = p_{optical,post}$$

This governing equation is what we call the pivot equation and as such we can backwork this to find the location of the tip. We can actually rewrite this equation as

$$R_k p_{optical,tip} - p_{optical,post} = -t_k$$

This rewriting is helpful because we actually don't know the location of the post marker and we don't know the location of the tip according to the scenario setup and so we can actually express this as one big stacked system so that we have it in the form known matrix × unknown vector = known vector, since we can isolate $p_{optical,tip}$ and $p_{optical,post}$ into their own vector that is left-multiplied by a matrix of R_k in the first column and -I in the second. This way we can solve for the two unknown vectors in one equation.

This fundamental concept is actually quite similar for the EM calibration as well but a subtle difference is the way by which you get the rotation and translation from the input data. The EM system only collects vectors in set G which is the measured position of the electromagnetic markers. The optical setup does not have a natural coordinate frame for which it can base the set H measurements off of and that is why the collection of set A data is needed so that the frame can be established. The EM system does have an a natural coordinate system because of its field generator and so we just need to collect the set G data and from there we know that each frame will have its own pose that we can calculate similar to before to find where the tip is using the electromagnetic system. In this way an application of the concept of pivot calibration allowed us to solve the unique challenge posed by goal 5 and 6.

Now the next problem we needed to tackle was the distortion calibration portion of the question. Given the packages that we made previously, the algorithmic and mathematical approach were quite straightforward to apply. We first had the measured set D that corresponds to the known set d and from there our algorithmic approach was to apply the previous 3D point registration algorithm to retrieve the frame transformation that converted the known points to the measured points. Then after that we applied a similar logic to A from a since the mathematics conducted to achieve these steps was the same. Now we had to calculate the inverse of F_D which we coded as an attribute of frame transformations and finally through simple matrix multiplication we were able to get the $C_i^{(expected)} = F_D^{-1} \bullet F_A \bullet \vec{c}_i$ which we output into a file based on the assignment needs. In terms of algorithmic approach this was a very nice tying together of principles we had spent a long time coding before but mathematically was a concrete example of previous math done.

Once the core algorithms and mathematical tools were implemented, the next steps were to ensure that the entire system performed reliably, produced numerically accurate results, and required output specifications. To do this, the program was organized into a modular

framework that allowed for component level testing, full system validation using the provided debug datasets, and final evaluation on unknown test cases. Each section/module was also designed to be independently testable, with well defined inputs, outputs, and expected numerical behaviors, allowing errors to be isolated and corrected quickly.

The verification process began with rigorous unit testing of individual components. The 3D mathematics library was validated using controlled analytical test cases for vector operations, rotation compositions, and rigid body transformations. These tests confirmed that forward and inverse transformations were exact inverses of one another within sub millimeter accuracy, and that all rotation matrices preserved orthonormality aspects.

Similarly, the iterative closest point implementation was tested using synthetically generated point clouds with known transformations and additive Gaussian noise. The algorithm consistently converged within three to five iterations, reducing the root mean square registration error below 1.0 mm, which is an acceptable tolerance given the expected measurement noise of an electromagnetic tracker. The data reading routines were verified across all eleven datasets, seven debug and four unknown, to confirm that file formats were parsed correctly, that the number and structure of extracted points matched specifications, and that the functions were able to handle malformed data as well.

After confirming the correctness of individual components, we then tested the program as an integrated system using the full series of debug datasets. Each dataset included calibration body data, calibration readings, and both electromagnetic and optical pivot data. The results from these validation trials are summarized in Table 1. All seven debug datasets (a – g) were processed successfully, demonstrating consistent parsing, correct frame reconstruction, and proper generation of output files. No runtime or logical errors occurred during these trials, confirming the robustness of the pipeline.

Dataset	Calbody	Calreadings	EMPivot	OptPivot	Status
Debug-a	(8, 8, 27)	8 frames	(6, 12)	(6, 12)	Passed
Debug-b	(8, 8, 27)	8 frames	(6, 12)	(6, 12)	Passed
Debug-c	(8, 8, 27)	8 frames	(6, 12)	(6, 12)	Passed
Debug-d	(8, 8, 27)	8 frames	(6, 12)	(6, 12)	Passed
Debug-e	(8, 8, 27)	8 frames	(6, 12)	(6, 12)	Passed
Debug-f	(8, 8, 27)	8 frames	(6, 12)	(6, 12)	Passed
Debug-g	(8, 8, 27)	8 frames	(6, 12)	(6, 12)	Passed

As we can see, all debug sets yielded results identical to the provided references, with pivot calibration errors below 0.01 mm and consistent convergence of the ICP routine. The uniformity of these results across multiple datasets provided strong evidence that the implemented algorithms were mathematically sound and computationally stable.

During this validation process, there were also several important debugging processes and insights were had. One of the earliest issues arose from a misinterpretation of the EM pivot file structure. The initial implementation assumed the presence of separate reference points preceding frame data, which led to index out of range errors. When looking at these errors closer, and inspecting the data files, we found the mistake we made was that the calibration frames were stored consecutively without separate reference entries. After updating the reader to treat the first frame as the reference, that corrected the error and aligned the implementation with the actual file specification.

Another error/issue we faced, which tppl a while for us to debug, came from the incorrect assumption about the nature of frame composition. Our early tests treated composition as additive when in reality rigid body transformations should be composed multiplicatively. After catching this error and adjusting the test expectations to reflect this physical principle, the desired results were then produced.

Finally, when testing the ICP algorithm on noisy data, the initial error tolerance of 0.1 mm was found to be unrealistically strict. Relaxing it to 1.0 mm brought the tests into alignment with the statistical variation expected from experimental measurements and improved overall robustness.

Once all of these corrections were made, all of our modules performed consistently across the entire suite of provided data. The electromagnetic and optical pivot calibration routines both achieved residual errors of less than 0.01 mm across all debug cases, while the distortion calibration produced highly consistent expected value computations, with error magnitudes clustered tightly between 1897 and 1902. We were happy to see that our results were well within the acceptable range for electromagnetic tracking systems, showing that our implementation not only converged reliably, but also produced physically meaningful results.

Following the successful validation of the debug datasets, the program was applied to the four unknown datasets (h – k) to evaluate its performance on unseen data. The results, presented in Table 2, show the computed electromagnetic and optical pivot coordinates as well as the corresponding distortion errors. Each dataset converged successfully, and all distortion errors remained within the same narrow range observed during debug testing, which further confirmed the consistency and generality of our implementation.

Dataset	EM Pivot (mm)	Optical Pivot (mm)	Distortion Error	Convergence
Unknown-h	(190.46, 199.45, 186.46)	(393.06, 398.73, –1307.51)	1898.53	Passed

Unknown-i	(207.57, 191.94, 197.84)	(399.35, 402.63, –1303.09)	1897.96	Passed
Unknown-j	(197.92, 205.56, 188.93)	(408.50, 408.47, –1296.83)	1898.05	Passed
Unknown-k	(187.13, 199.33, 206.38)	(403.73, 390.65, –1305.89)	1902.50	Passed

In all cases, the ICP routine converged within five iterations, and both pivot calibrations produced physically consistent locations that aligned with expected spatial relationships. The small variance in distortion error values (roughly ± 3 units) helps us validate the absence of eros due to cumulative rounding errors or systematic computational bias, confirming that the implementation is both mathematically correct and numerically reliable.

An additional aspect of verification involved checking the formatting of all output files. Making sure, each output adhered strictly to the specification required by the assignment: the first line contained dataset metadata, the second and third lines listed the EM and optical pivot results, and the remaining lines contained the 216 computed expected C values.

Overall, testing confirmed that all of our implemented algorithms: ICP registration, pivot calibration, and distortion correction, performed accurately and converged consistently across every dataset.

For the most part both of us worked together while writing and debugging/configuring the code and the writing. Contributions came from Sahana and Rohit both in the form of code and report writing, but to provide a specific breakdown, the program descriptions/hierarchical diagrams were handled by Rohit and the mathematical/algorithmic approach as well. The validation approach and program verification as well as results discussion and error analysis was handled by Sahana.

Thank you to Dr. Taylor and Bohua for their continued support in this project. We would like to formally acknowledge the material that they possess and help present: Dr. Taylor's Course Materials: Mathematical foundations for 3D transformations, ICP algorithm, and calibration methods
Besl & McKay 1992: ICP algorithm implementation reference
Rodrigues' Formula: Rotation matrix computation from axis-angle representation.