EECS 498: Introduction to Algorithmic Robotics Final Project
Point Cloud Processing

Kristian Cho (krischo), Gillian Minnehan (gminn)

Introduction

Point set registration and alignment of point clouds is used across numerous applications including medical imaging, virtual and augmented reality, autonomous driving, and reconstruction of 3D objects from multiple laser scans. Finding alignments not only accurately but quickly is critical for the merging of multiple data sets into a globally consistent model and mapping new data to old data [1]. 3D data from LiDARs and RGB-D cameras often have various densities, noise levels, and initial poses, all of which present particular challenges with aligning 3D points with speed and accuracy. As we learned this term, a simplistic approach to point set registration is the Iterative Closest Point algorithm using Euclidean distance to find the point correspondences between a source and a target point cloud. In this report, we present an alternative approach to point set registration that is a significant improvement of this method by instead using pre-computed point feature histograms, which describe more about the curvature of the surface around the given point. As a result, our solution is pose invariant, robust to noise, and not sensitive to different sampling densities. First we will discuss our implementation and present our reasoning for our algorithmic choices. Then, we will present results of our implementation that demonstrate the efficacy of our solution and its superiority over more primitive point-registration methods.

Implementation

Iterative Closest Point

As per a previous homework assignment, we have already implemented the Iterative Closest Point (ICP) algorithm. Seen below is some pseudo code demonstrating how ICP works; note how there is no mention of a different process for computing initial alignment.

```
Algorithm 1 ICP
1:  procedure ICP($P, Q$)
2:      while not Done do
3:          $C = 0$
4:          for each point $p_i$ in $P$ do
5:              find the closest $q_i$
6:              $C = C \cup \{p_i, q_i\}$
7:          end for
8:          $R, t \Leftarrow \mathrm{GetTransform}(C_p, C_q)$
9:          if $\sum_{i=1}^{n} \|(\mathrm{R}C_{pi} + t) - C_{qi}\| < \epsilon$ then
10:             return $P$
11:         end if
12:         for each $p_i$ in $P$ do
13:             $p_i = Rp_i + t$
14:         end for
15:
```

Figure 1: ICP algorithm

In our implementation, we use ICP to find point-to-point correspondences between source and target point clouds. These correspondences are found using the minimal Euclidean distances between the points of the two point clouds. From here, we incrementally transform our source point cloud using our correspondences and Principal Component Analysis (PCA) until we achieve an error below that of a user-input threshold. This error metric is the summed squared difference between the points in both point clouds, as shown in line 9 of the pseudocode.

Point Feature Histogram (PFH)

ICP performs poorly because it uses Euclidean distance to computate correspondences. As a result, it is highly sensitive to noise and to the initial pose of the source cloud. Point Feature Histograms (PFH) were designed to address these weaknesses by representing each point with a signature that is representative of surface geometric properties between the point and its neighbors. Since the PFH method is only calculating correspondences, it can then be used to find an initial alignment that is further improved upon via ICP. A flowchart showing our PFH algorithm is shown below:

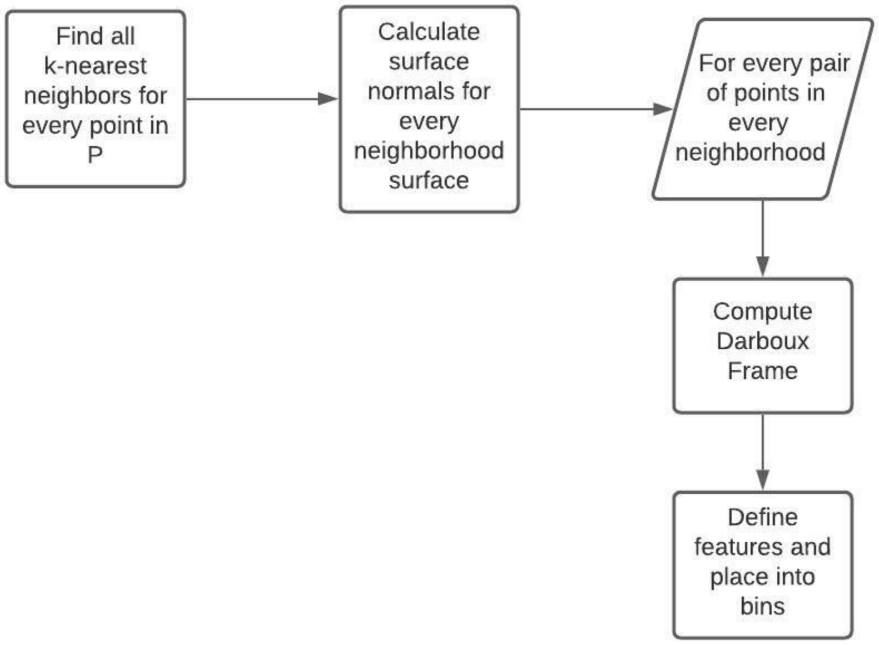

Find all k-nearest neighbors for every point in P
Calculate surface normals for every neighborhood surface
For every pair of points in every neighborhood
Compute Darboux Frame
Define features and place into bins

Figure 2: PFH Algorithm

The first step of this process involves finding the k-nearest neighbors for each point in our source point cloud P. For this, we used a k-d tree with Euclidean distance. With these neighborhoods now collected, we then estimated the surface normals for each of these neighborhoods using the PCA algorithm by building covariance matrices. Given that we do not know what direction these normals point, we re-orient them using the current viewpoint.

$$\text{if } \frac{\langle v - p_i,\ n_{p_i} \rangle}{\|v - p_i\|} < 0, \text{ then } n_{p_i} = -n_{p_i}$$

Figure 3: Re-orienting surface normals [2]

Then, for every unique pair in each neighborhood, we compute the Darboux frame according to the figure below:

$$u = n_s,\ v = (p_t - p_s) \times u,\ w = u \times v.$$

Figure 4: Darboux Frame calculation [2]

Finally, we compute the features for this unique pair and use their values to calculate its bin index. Note that feature 2 is a distance between the two points, and as we will see later, its usage is not necessary.

$$\left.\begin{aligned} f_1 &= \langle v,\ n_t\rangle \\ f_2 &= ||p_t - p_s|| \\ f_3 &= \langle u,\ p_t - p_s\rangle / f_2 \\ f_4 &= atan(\langle w,\ n_t\rangle, \langle u,\ n_t\rangle) \end{aligned}\right\} idx = \sum_{i=1}^{i\leq 4} step(s_i, f_i) \cdot 2^{i-1} \qquad (3)$$

Figure 5: Feature and bin calculation [2]

These bin indices correspond to where in the histogram a given pair is located such that we add the number of pairs for each index to a 16D vector allocated for each point in P. For example, if we got 5 pairs in bin 1 and 2 pairs in bin 3, our vector would be [0 5 0 3 0 0 0 0 0 0 0 0 0 0 0 0]. We used 16 bins as our initial limit as calculated by q^d where q is subdivision intervals in a feature's value range and d is the number of features selected (for our implementation, we used 2 subdivision intervals and 4 features, totalling 16 bins). Finally, we normalize each 16D vector to calculate our signature, which is the percentage of pairs in each bin. Our signatures contain this 16D vector for every point in P to create a k x 16 matrix.

We call our PFH function on both the P and the target point cloud Q. We run ICP such that the first pass uses the signatures of P and Q to calculate their correspondences. This in turn gives us our initial alignment, where we use ICP with the euclidean distances between P and Q to finalize our solution.

Fast Point Feature Histogram (FPFH)

While PFH generates a significantly more accurate solution over exclusively using ICP, its runtime proves to be an issue as it runs in $O(nk^2)$ time. This occurs because we must generate every unique pair of points through every neighborhood. To improve computation time, we implemented the Fast Point Feature Histogram (FPFH). This algorithm uses a separate algorithm known as Simplified Point Feature Histogram (SPFH) to bring down the runtime to $O(nk)$. SPFH is identical to PFH in regards to calculating bin indices except for one important caveat: instead of looking for every unique pair of points in each neighborhood, SPFH exclusively uses the pairs where a given point is the source point. Because we calculate signature vectors for each point, our given point when using SPFH is always the point whose neighborhood we are using. After calculating our SPFH signatures, we use these values to calculate our final signatures:

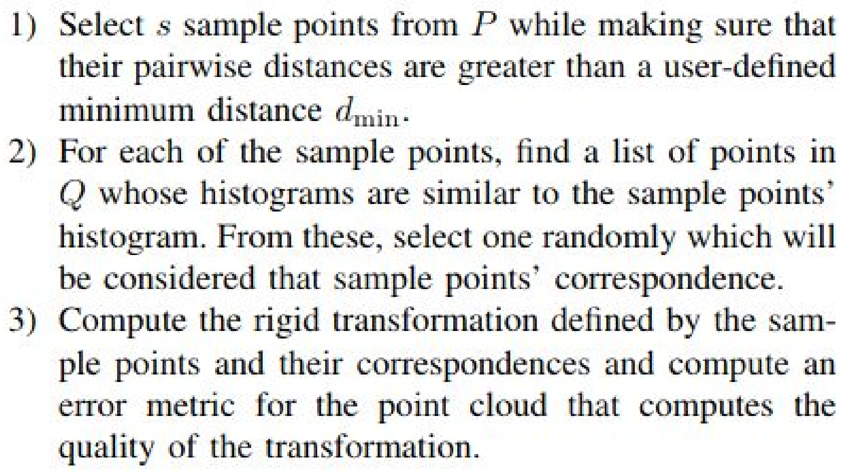
$$FPFH(p) = SPF(p) + \frac{1}{k}\sum_{i=1}^{k} \frac{1}{\omega_k} \cdot SPF(p_k)$$

Figure 6: FPFH calculation [3]

Here, w_k corresponds to the distance between the source point and its neighbor. The signatures calculated by FPFH are used just as the signatures of PFH were used; FPFH is called on both P and Q to be used in our ICP implementation to find our initial alignment. To further (minimally) decrease runtime, we also eliminated the use of our 2nd feature, distance, which brought our number of bins down to 8. Recent studies as well as our results found that the elimination of the distance feature causes no significant change in robustness.

Sample Correspondence Initial Alignment (SAC-IA)

With our FPFH implemented, we decided to also try an additional algorithm known as Sample Correspondence Initial Alignment (SAC-IA) to improve issues in which our initial alignment became trapped in local minima. This algorithm is used after our signatures are calculated, taking place during our first pass through ICP to substitute our correspondence calculations. This process is shown below.



1) Select s sample points from P while making sure that their pairwise distances are greater than a user-defined minimum distance $d_{\min}$.
2) For each of the sample points, find a list of points in Q whose histograms are similar to the sample points' histogram. From these, select one randomly which will be considered that sample points' correspondence.
3) Compute the rigid transformation defined by the sample points and their correspondences and compute an error metric for the point cloud that computes the quality of the transformation.

Figure 7: SAC-IA steps [3]

For the third step, we use a different error metric known as the Huber penalty measure:

$$L_h(e_i) = \begin{cases} \frac{1}{2}e_i^2 & ||e_i|| \leq t_e \\ \frac{1}{2}t_e(2||e_i|| - t_e) & ||e_i|| > t_e \end{cases}$$

Figure 8: Huber penalty measure [3]

We run SAC-IA for a specific number of iterations, keeping track of our best computed transformation according to our error metric. Finally, we set the initial alignment with this transform and run ICP as normal. In practicality, however, we found that the time increase associated with this solution far outweighed the minimal benefit to alignment. Therefore, we chose not to use it in our final implementation.

Results

To test our final implementation, we used real point cloud data from the Point Cloud Library Dataset [4] with varying densities, poses of the target cloud in relation to the source cloud, and noise levels. These experiments involved loading the source and synthesized target point cloud, passing them both into our algorithm, then printing relevant metrics to measure the desirability of the solution. For each experiment, we first looked specifically at how our implementation performed in regards to overall speed, number of iterations, and accuracy for each source and target point cloud pair compared to the original ICP algorithm and the original PFH algorithm. To reach our final implementation, we also made modifications to the number of bins and the number of neighbors to optimize our algorithm for speed and accuracy, which we present data on last.

Well-Formed Lioness Data

The results of two separate experiments using a well-formed, real point cloud of a lioness figure with a total of 3400 points are displayed in **Figures 9** and **10** below. Each experiment involved testing the original ICP algorithm, ICP with PFH, and ICP with FPFH using a different source pose that was synthetically generated from the lioness data. This original figure acted as our target point cloud.

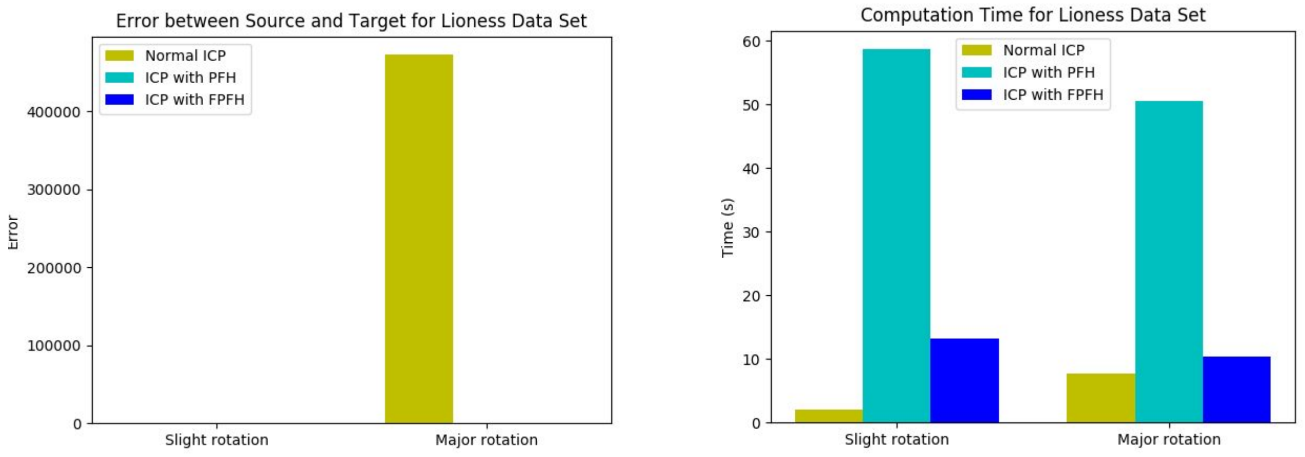


Figure 9: Comparison of accuracy and speed of each algorithm on each test case in the lioness data set

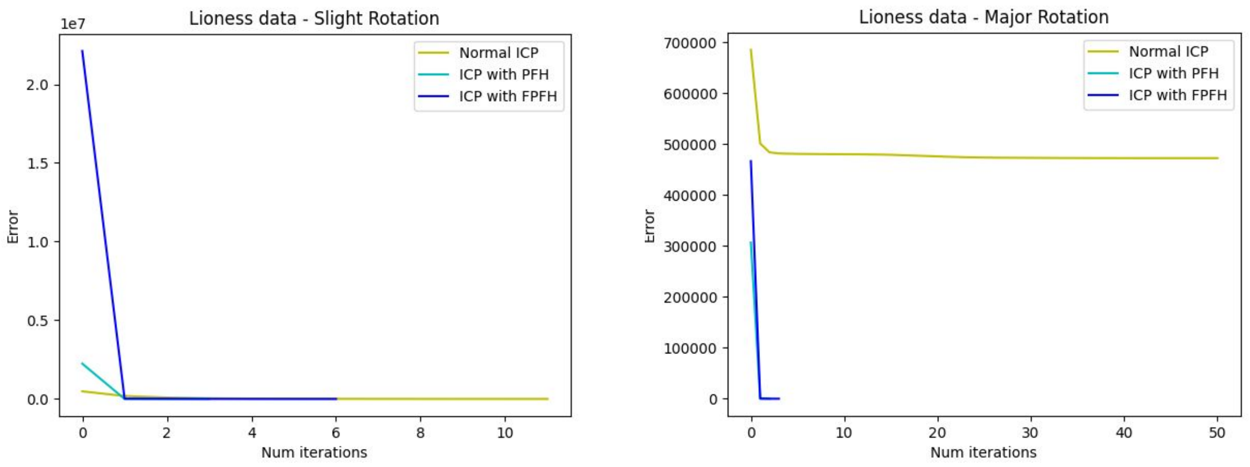


Figure 10: Comparison of crror between source cloud and target cloud per iteration of the ICP algorithm

To start, we slightly rotated this point cloud about the y-axis to obtain the source and target point clouds, which can be seen in the far left graph in **Figure 11**. All three algorithms reached a negligible error, as seen in the graph on the left in **Figure 9**. However, they reached these thresholds at varying speeds, which is displayed in the graph on the right in **Figure 9**. As expected, we see a significant difference in the computation time of PFH and FPFH, clearly indicating that FPFH reduces time without sacrificing accuracy. However, normal ICP performed about 10 times faster than FPFH. This result indicates that for a source with only a slight rotation, normal ICP is preferred since it takes less time to find an initial alignment. As shown in the left graph of **Figure 10**, FPFH requires less iterations to reach the error threshold, which indicates the superiority of the initial alignment using point feature histograms over that of the original ICP algorithm using Euclidean distance.

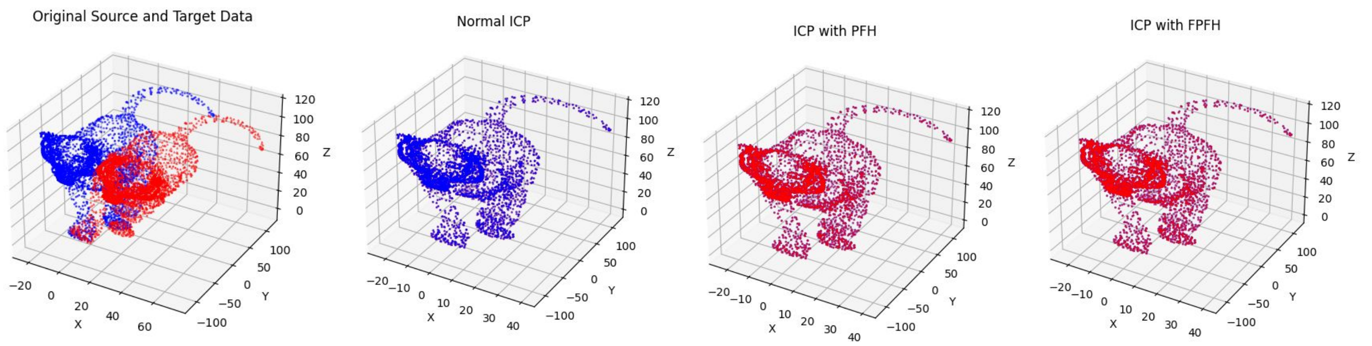


Figure 11: Visualization of original point cloud data (far left) and resulting alignments for each algorithm

To test the performance of both algorithms when the source is in an extremely different pose than the target, we performed a drastic transformation of the original point cloud, which can be seen in the far left graph of **Figure 12.** Since the point feature histogram method is pose invariant, unlike regular ICP, we expect FPFH and PFH to perform well and normal ICP to fail. After running this test, this is exactly what occurs. The error for normal ICP converges at a very high value, and these results are shown in **Figures 9** and **10.** Its final alignment along with PFH and FPFH can be seen below in **Figure 12.**

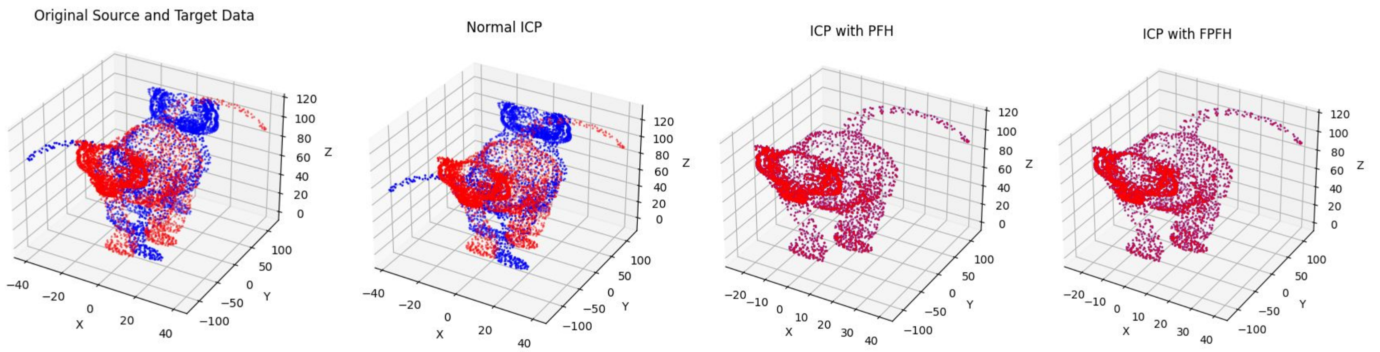
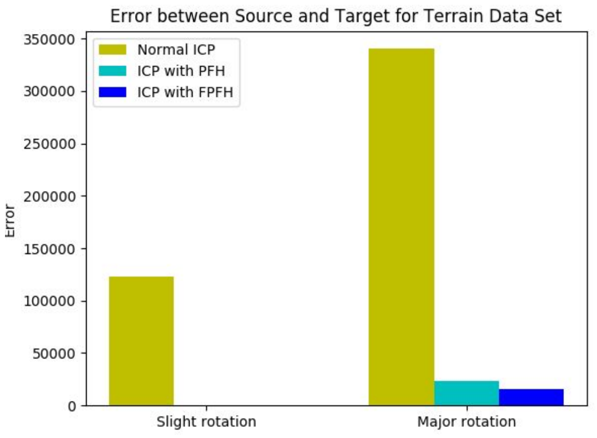


Figure 12: Visualization of original point cloud data for the majorly rotated data (far left) and resulting alignments from each algorithm

Our solution dropped steeply down to the error threshold while regular ICP gradually curved down to its final error rate (**Figure 10**). Interestingly, our solution is faster than the previous experiment, which demonstrates that FPFH and PFH are invariant to pose both with respect to computation time and accuracy.

Terrain Data

For our next experiment, we chose a more dense and noisy point cloud of a terrain with 15,645 points in order to test the performance of the two algorithms when dealing with a more complex surface with more natural noise.



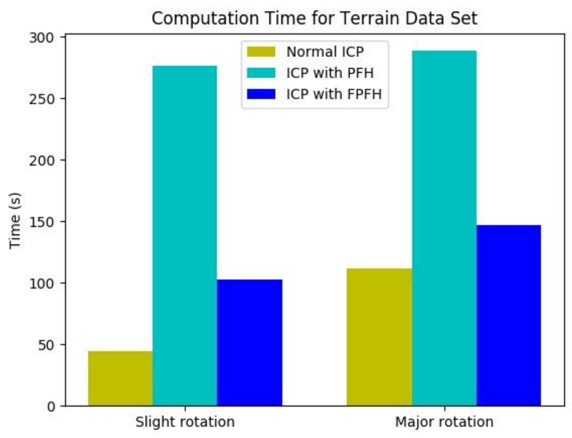
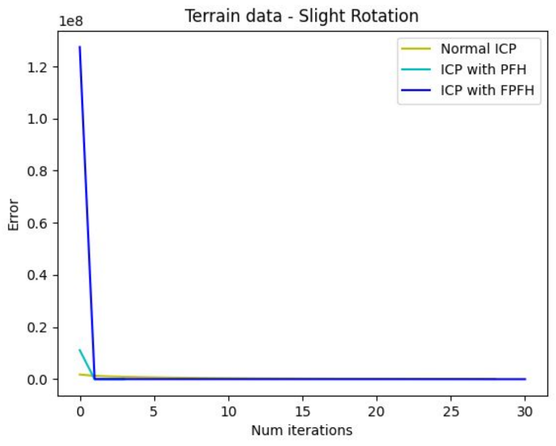


Figure 13: Comparison of accuracy and speed of each algorithm on each test case in the terrain data set



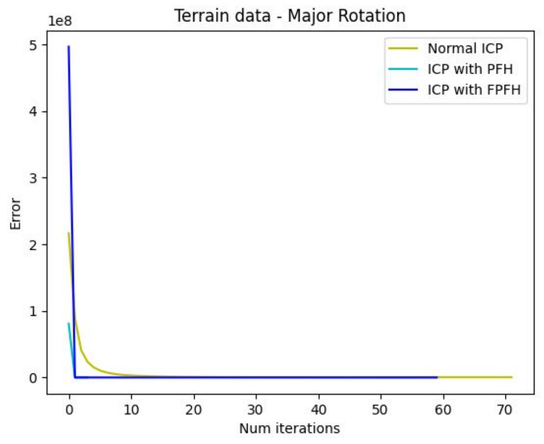


Figure 14: Error between source cloud and target cloud per iteration of the ICP algorithm for slightly rotated and majorly rotated source clouds

The terrain and transformed target can be seen in **Figure 15.** As plotted in the left graph of **Figure 13**, normal ICP had a significant amount of error while PFH or FPFH had a negligible error, demonstrating that the histogram method performs better than normal ICP when there is more noise in the data set.

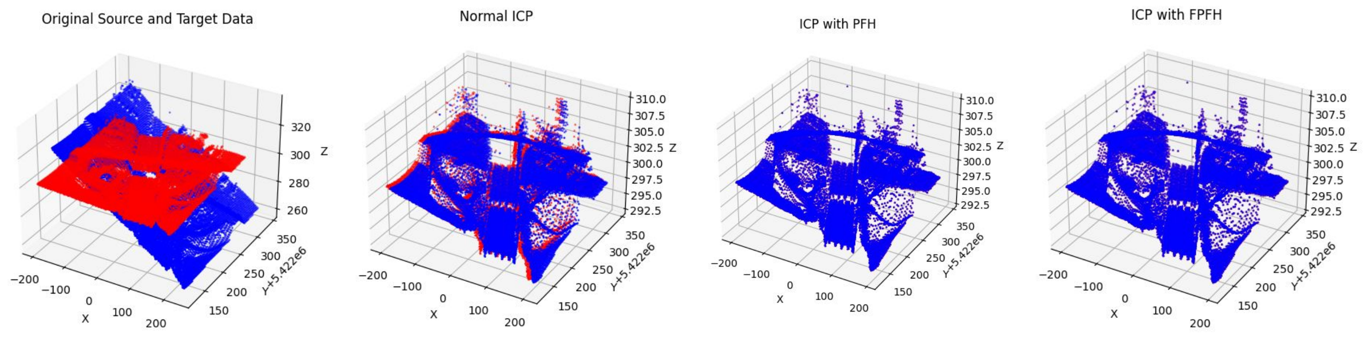


Figure 15: Visualization of original point cloud data for the slightly rotated data (far left) and resulting alignments from each algorithm

In regards to speed, FPFH was about twice a slow as normal ICP while PFH performed significantly worse than both. These results are graphed in the right image in **Figure 13**.

To test the performance of both algorithms when the source is in an extremely different and non-overlapping pose than the target, we performed a drastic transformation of the original point cloud and ran each solution with the data, results of which can be seen in **Figure 16.** We found that computation time and minimum error reached were significantly more optimal with our solution than that of the normal ICP algorithm. Regular ICP converged at an error of over 300,300 in about 100 seconds while our solution reached a negligible error after about 150 seconds. Again, these results are summarized in the right graph in **Figure 13**. Similar to the previous experiment, FPFH was slower than normal ICP, but by only about 20%. As expected PFH performed significantly worse than both. Additionally, **Figure 14** shows that FPFH converges much faster than normal ICP, indicating again that the initial alignment for FPFH was more accurate. Note that the scale for the right graph of **Figure 14** is very large, which makes it appear as though the final errors of all algorithms are similar when in fact normal ICP's error is much greater. The final errors reached are more clearly displayed in the left graph of **Figure 13**.

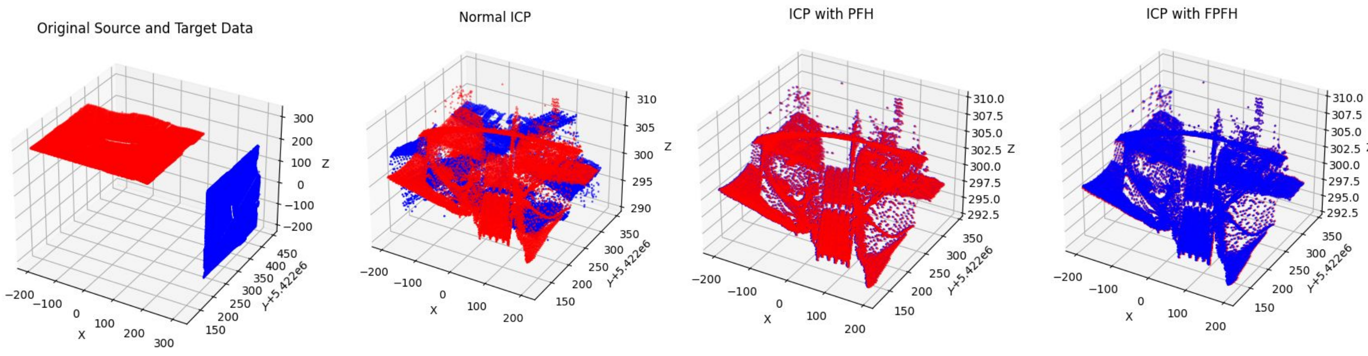


Figure 16: Visualization of original point cloud data for the majorly rotated data (far left) and resulting alignments from each algorithm

As discussed in our implementation section, we made several adjustments to our solution including choosing the fast point histogram method, changing the number of bins, and changing the number of neighbors. Since it has a better initial alignment, PFH took less iterations of ICP and therefore less time to align than FPFH, as seen in **Figures 10 and 14**. Therefore, as expected, these results indicate that the largest time improvement when using FPFH occurs in computing of the signatures. Secondly, we investigated the effect of changing the number of bins, and results of a simple test for the slightly rotated lioness data set are summarized in **Figure 17.** By eliminating the distance parameter from our features vector we reduced our number of bins from sixteen to eight. By removing an additional parameter, alpha, we reduced our number of bins to four. As seen **Figure 17**, reducing the number of bins resulted in a reduction in the computation time, and a reduction in error. Additionally, the similarity in the error between eight and sixteen bins indicates that the distance metric is largely not useful for registration since distance between points changes with distance from the laser scanner. Therefore, we chose 4 bins for our final implementation.

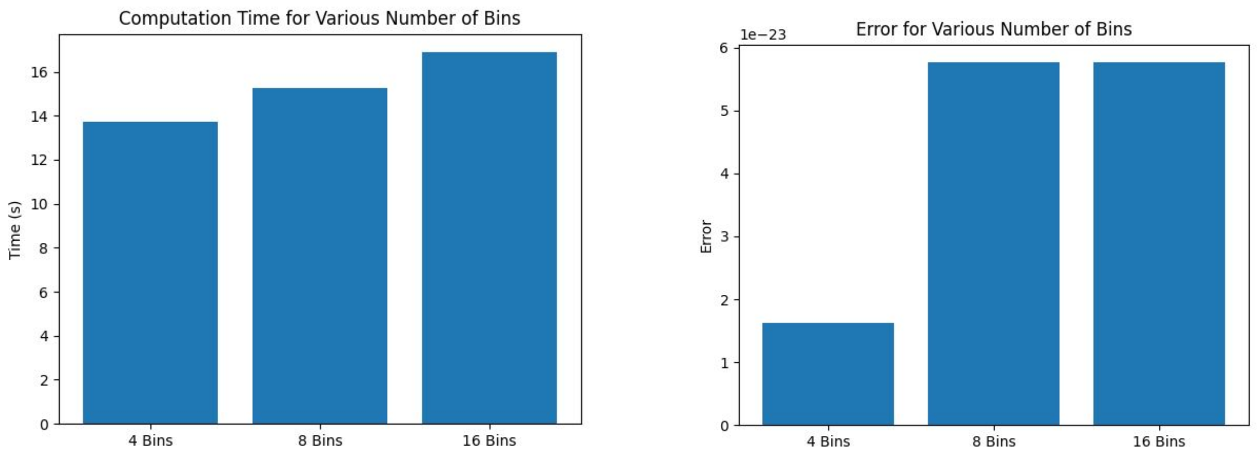


Figure 17: Comparison of the results of FPFH on the slightly rotated lioness data set with varying number of bins

Finally, we adjusted the number of neighbors to consider for the fast point feature histogram method and recorded the computation time and error for 5, 10, and 25 neighbors again for the slightly rotated lioness data set, which is displayed in **Figure 18.** The results indicate that a goldilocks number of neighbors exists; too many neighbors takes too much time and not enough neighbors tends to have a slightly higher error. Therefore, we chose 10 neighbors for our final implementation.

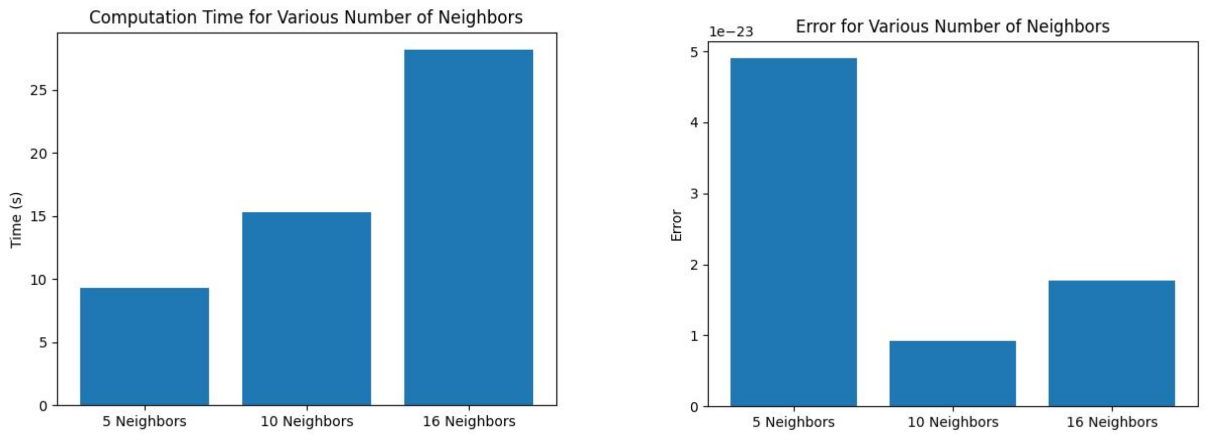


Figure 18: Comparison of the results of FPFH on the slightly rotated lioness data set with varying number of neighbors

As mentioned previously, we also implemented the SAC-IA method to improve the initial alignment for FPFH so the ICP algorithm converged faster. However, the computation time was unacceptably larger than that of FPFH without SAC-IA and the accuracy was about the same. Therefore, we chose not to use SAC-IA in our final algorithm.

5. References

[1] Point set registration. (2020, December 10). Retrieved December 16, 2020, from https://en.wikipedia.org/wiki/Point_set_registration

[2] R. B. Rusu, N. Blodow, Z. C. Marton, and M. Beetz. Aligning point cloud views using persistent feature histograms. In 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 3384–3391. IEEE, 2008.

[3] R. B. Rusu, N. Blodow, and M. Beetz. Fast point feature histograms (fpfh) for 3d registration. In 2009 IEEE International Conference on Robotics and Automation, pages 3212– 3217, 2009.

[4] Point Cloud Library. (2020, December 10). Retrieved December 16, 2020, from https://github.com/PointCloudLibrary/data