Point Set Registration - Final Report

Achyuth Kashyap
University of Michigan
akashyap@umich.edu

Madhavan Iyengar
University of Michigan
miyen@umich.edu

1 Introduction

1.1 Motivation

Point set registration is a solution to many problems both in and out of robotics. The key goal is to align a model of a 3D object or scene to a real-world point set.

In robotics, point set registration is often used to perform Simultaneous Localization and Mapping (SLAM). When a robot needs to map its environment, it can use sensors such as LiDAR to construct a 3D representation of the environment based on its current location. When the robot moves to another location, it can construct a new 3D representation of the same scene from a different view. Point set registration can be used to reconcile the differences and align the two scenes.

Outside of robotics, point set registration is often used in the medical world. When medical scans (CT, MRI, etc.) are taken, point set registration can be used to align corresponding scans. This can help medical professionals make better diagnoses.

1.2 Iterative Closest Point

Iterative Closest Point (ICP) is a popular algorithm used for point set registration. At a high level, the algorithm computes correspondences between points in two different point clouds (commonly referred to as the **source** and **target**) using a predetermined similarity metric. The most basic of these is the Euclidean distance metric. ICP then computes a transformation (rotation and translation) to map one set of points onto the other. This process is done iteratively until a stopping criteria is met.

There are many problems that occur with vanilla ICP:

1. ICP is sensitive to sensor noise
2. ICP does not account for connectivity of points in a point set
3. ICP is sensitive to the initial orientations of the point sets

In our project, we explore solutions to these problems by improving on the distance metric used to compute correspondences in ICP.

1.3 Point Feature Histogram

Point Feature Histograms (PFH) is an improvement upon traditional ICP. In specific, PFH captures the geometric properties of a neighborhood of points, accounting for how properties vary in different directions. At the core, each point receives a **signature**, which is a vector constructed based on the change in surface normals around that point. These signatures are used as the metric to determine the similarity of points while running ICP instead of the traditional Euclidean distance metric described previously.

The results of this improvement include:

1. ICP w/ PFH can estimate correspondences based on surface patch similarity (initialization matters less)
2. ICP w/ PFH is invariant to the pose of the underlying surface
3. ICP w/ PFH is not very sensitive to noise

Throughout the course of this report, we will showcase our findings about the improvements of PFH compared to vanilla ICP.

2 Implementation

2.1 Original ICP Algorithm

Below is a high-level pseudocode for the original ICP Algorithm. Our report focuses on improving the distance function DIST() used in line 7. The most basic version of the ICP algorithm uses the Euclidean distance:

$$\mathrm{DIST}(p, q) = \sqrt{\|p - q\|_2}$$

Algorithm 1 Iterative Closest Point (ICP) Algorithm

```
1:  Input Point clouds S, T
2:  while not done do
3:      Compute Correspondences:
4:      $C \leftarrow \emptyset$
5:      for each point $s_i \in S$ do
6:          Find the closest point $t_i \in T$ using DIST()
7:          $C \leftarrow C \cup \{s_i, t_i\}$
8:      end for
9:      Compute Transform:
10:     $(R, t) \leftarrow \mathrm{GetTransform}(C)$
11:     Update $S$:
12:     $S \leftarrow RS + t$
13: end while
14: return $P$
```

Algorithm 1: Pseudocode for the Iterative Closest Point (ICP) Algorithm [1].

2.2 Point Feature Histogram

We will use PFH to create a more robust distance metric to use in line 6 of Algorithm 1. At a high level, this algorithm computes the distance between points s_i and t_i by leveraging the neighborhood of points around each point of interest. It uses the change in surface normals around the points to determine distances, assigning a lower distance for points that have similar neighborhoods.

2.3 PFH Derivation

For every point in the point clouds, we would like to compute a **signature** that is more expressive than just the xyz coordinate of that point. In other words, for p_i, we want to compute $s(p_i)$. Figure 1 shows the pipeline to compute $s(p_i)$.

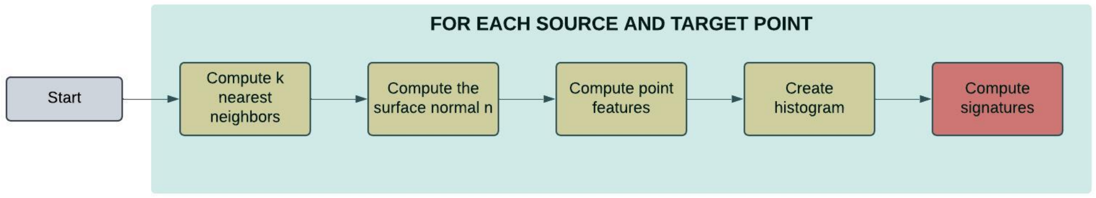

FOR EACH SOURCE AND TARGET POINT
Start
Compute k nearest neighbors
Compute the surface normal n
Compute point features
Create histogram
Compute signatures

Figure 1: PFH signature computation process.

2.3.1 Compute Normal Vectors

First, we need to compute a **normal vector** n_i for p_i. To do this, we select the k closest points by Euclidean distance. We do this to form a surface patch around p_i that has a normal vector. Let the set of all k points form matrix X. The surface normal for p_i is the direction of minimum variance for the set X. This is equivalent to the eigenvector corresponding to the smallest eigenvalue of X. We can compute this normal vector n by taking the singular value decomposition of $X^T X$.

2.3.2 Compute Point Features

Once we have n_i for each point in the point clouds, we can then proceed to compute **point features** for each p_i that highlights how the surface normal changes in its neighborhood. The goal is that we can use these features to better determine similarities between points. To do this, we start by computing a frame (reference the picture below) for each pair of points defined by the following equations.

1. Calculate the distance between points. $d = ||p_t - p_s||$
2. Define the surface normal. $u = n_s$
3. Calculate supporting axis one. $v = u \times \frac{p_t - p_s}{d}$
4. Calculate supporting axis two. $w = u \times v$

From this computed frame, we can calculate **angular features** that capture the change in surface normal between the points. Given these angular features, we can describe the variations in the surface around point p_i, which will hopefully provide unique and useful signatures for each point, allowing for a good distance metric. Refer to Figure 2 for the exact definitions of the angles.

1. $\alpha = v \cdot n_t$
2. $\phi = u \cdot \frac{p_t - p_s}{d}$
3. $\theta = arctan(w \cdot n_t, u \cdot n_t)$

We can now form a vector representation for each pair of points using the computed angular features.

$$< \alpha, \phi, \theta, d >$$

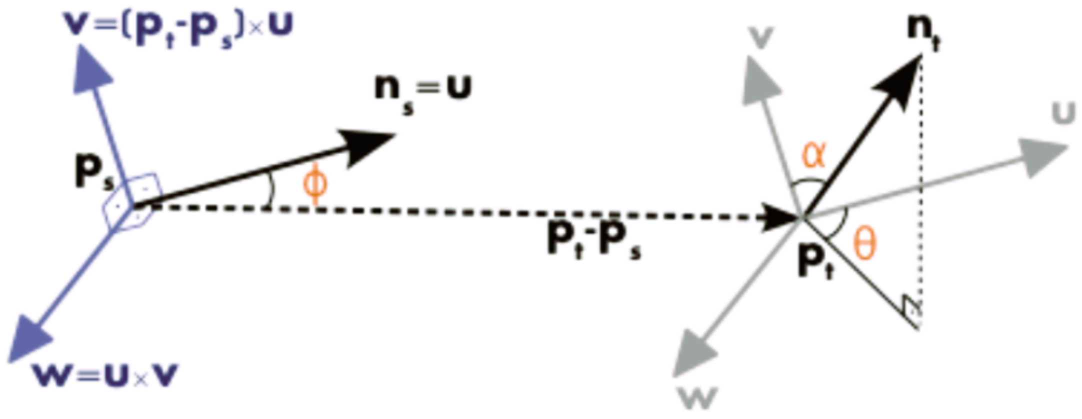


Figure 2: Diagram depicting change in surface normal between two points [1].

2.3.3 Create Histogram of Point Features

Now that we have the point features for each point in the source and target point cloud, we can compute a **histogram** of the point features for each point. The histogram will provide a better metric to compare two points as it groups the k nearest neighbors for the points by their feature vectors. By doing this, we are considering the features of the neighborhood of each point when performing a comparison rather than just the point itself, which results in a more thorough comparison metric. We choose the k closest neighbors in terms of the Euclidean distance because those will allow for the best estimate of the surface patch around p_i. Each bin in the histogram is a range of values for the point feature vector $< \alpha, \phi, \theta, d >$. The number of bins is a hyperparameter that the user can experiment with to yield the best results.

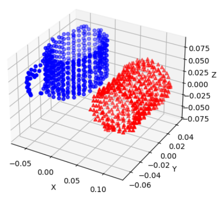
2.3.4 Compute Signatures

The final step is to consolidate all of the work into a single comparison metric known as the **signature**. The signature for each point $s(p_i)$ is a vector of the percentage of pairs that fall into each bin of the respective histogram. In other words, it is a probability distribution representing the likelihood of the angular features of the k nearest neighbors of each point. The signature is the concise metric that we computed to replace the previous distance metric in line 6 of Algorithm 1. Now, we are able to modify the ICP algorithm to use the more robust signature distance metric to achieve the point set registration task.

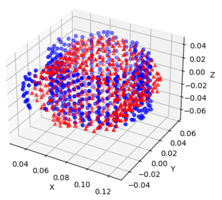
3 Results

3.1 Improvements from our method

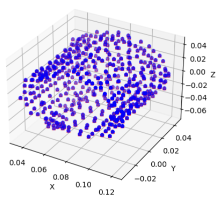
We notice significant performance improvement in point set registration with ICP when replacing the Euclidean distance metric with the PFH. Figure 3 visualizes the final aligned point clouds when running ICP with the two metrics. We notice that the alignment between point clouds is significantly better when using PFH. We quantitatively measure this in Figure 4. We notice that ICP with the Euclidean distance metric takes many more iterations to converge, only reaching a minimum error of **0.042**. Using PFH, we only require 2 iterations to converge, reaching a much lower error, below **1e-30**. However, we notice that the overall convergence times are very similar, with ICP w/ Euclidean taking **22.79** seconds for 28 iterations and ICP w/ PFH taking **23.14** seconds for 3 iterations. It's clear that leveraging PFH causes each iteration to take much longer, which makes sense due to the increased computation that is required for generating point signatures. From these direct comparisons, we have evidence to suggest that using PFH instead of Euclidean distance improves the quality of point set registration with ICP.



(a) Initial configuration

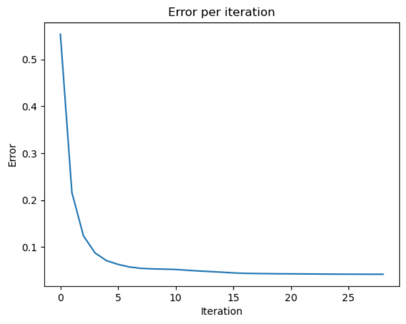


(b) ICP w/ Euclidean metric

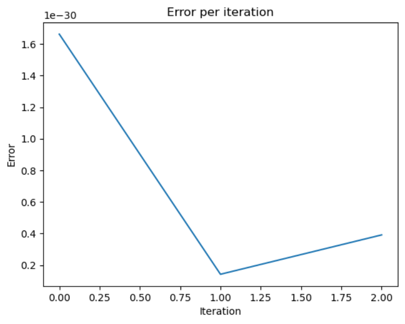


(c) ICP w/ PFH

Figure 3: Comparison of ICP results using different distance metrics (source pcd in blue, target pcd in red). **(a)** pre-aligned pointclouds before running ICP. **(b)** Final alignment using ICP with Euclidean distance metric, showing a slight misalignment. **(c)** ICP using a PFH distance metric, demonstrating significantly improved alignment.



(a) ICP w/ Euclidean metric



(b) ICP w/ PFH

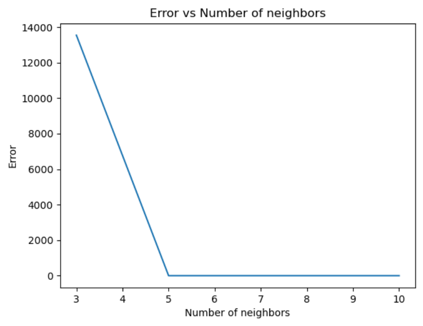
Figure 4: Comparison of alignment errors per ICP iteration using different distance metrics. **(a)** errors for ICP with Euclidean metric, which take over 25 iterations to converge. **(b)** errors for ICP with PFH metric, which are much lower than the errors in (a).

3.2 Hyperparameter tuning

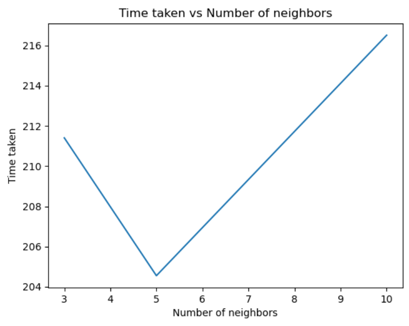
The two most important hyperparameters when computing PFH are the number of neighbors used to compute variation in normals and the number of bins used for computing histograms. We run an extensive hyperparameter search to determine the best combination of these parameters. To make the problem more difficult, we use a pointcloud of a cat from [2], which is more dense than our mug example. We only run ICP for one iteration because we empirically found that increasing the iterations past that does not help with convergence. The source pointcloud is generated by duplicating the target and applying a rotation + translation.

3.2.1 Searching over NUM_NEIGHBORS

We search over NUM_NEIGHBORS $\in \{3, 5, 10\}$. The smallest value we can do is 3 because we require 3 neighbors to compute a normal vector. Our results are shown in figure 5. We see that the error decreases up to a certain point after which it remains the same. However, the time seems to generally increase as we increase the number of neighbors.



(a) Error plot

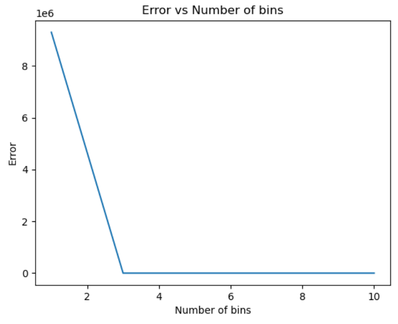


(b) Time plot

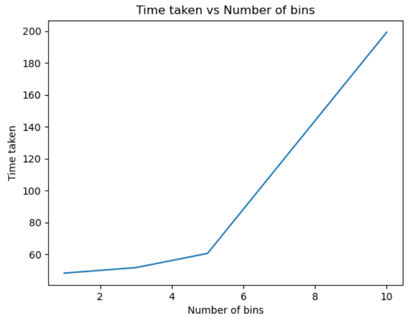
Figure 5: Error vs number of neighbors for ICP with PFH. We test NUM_NEIGHBORS $\in \{3, 5, 10\}$ with a constant NUM_BINS $= 10$. We find that we reach the minimum error by NUM_NEIGHBORS $= 5$.

3.2.2 Searching over NUM_BINS

We search over NUM_BINS $\in \{1, 3, 5, 10\}$. Our results are shown in figure 6. We see a similar pattern as in Figure 5, with error decreasing up to a certain point before tapering off and time increasing directly with number of bins.



(a) Error plot



(b) Time plot

Figure 6: Error vs number of neighbors for ICP with PFH. We test NUM_BINS $\in \{3, 5, 10\}$ with a constant NUM_NEIGHBORS $= 5$. We find that we reach the minimum error by NUM_BINS $= 5$.

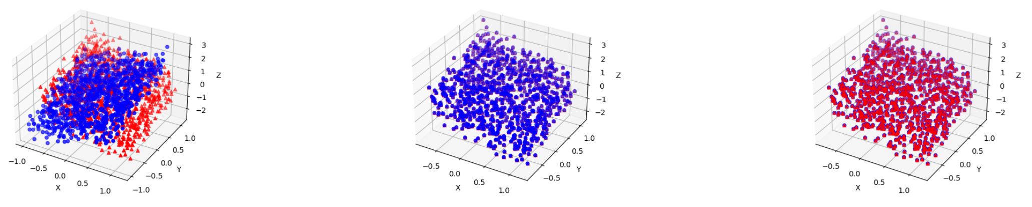
Therefore, from our hyperparameter search, we find an optimal combination of NUM_NEIGHBORS $=$ NUM_BINS $= 5$. Using higher values than these leads to the same performance with longer computation time. Therefore we use these parameters for the experiments in Section 3.3.

3.3 More results with PFH

To further verify our performance improvements, we performed more comparisons between our method and ICP with Euclidean distance, detailed in this section.

3.3.1 Synthetic Pointcloud

We generate a synthetic source pointcloud by constructing a plane in 3D with noise to imitate noise from sensor readings and constructing a target pointcloud by transforming the source. We demonstrate relatively similar performance in Figure 7 with both distance metrics, with PFH (final error 1.73e-28) still outperforming Euclidean distance (final error 1.19e-26). This tells us that that even though the PFH distance metric does better, the amount by which it does better is highly dependent on initialization.

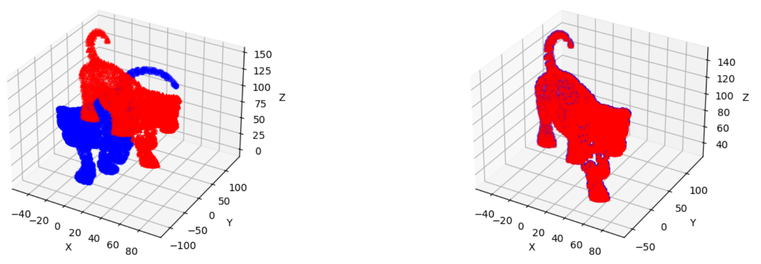


(a) Initial configuration (b) ICP w/ Euclidean metric (c) ICP w/ PFH

Figure 7: Comparison of ICP results on synthetic example using different distance metrics (source pcd in blue, target pcd in red).

3.3.2 Pointcloud from internet

We also test our results on another pointcloud from the internet. Figure 8 depicts a lioness pointcloud obtained from [2]. We show that the



(a) Initial configuration (b) ICP w/ PFH

Figure 8: Visualization of ICP results with PFH on the lioness example (source in blue, target in red).

4 Conclusion

In this report we demonstrate tangible improvements onto ICP by utilizing the PFH signature as a distance metric for determining correspondences. We perform both quantitative and qualitative comparisons between both methods as well as perform a hyperparameter search to determine the optimal parameters for PFH. Future work might involve including more heuristics to improve the quality of correspondence matching from PFH.

References

[1] Berenson, D. (2024). *Introduction to Algorithmic Robotics Lecture Slides*, University of Michigan.

[2] Point Cloud Library Data Repository. (2014). Available at `https://github.com/PointCloudLibrary/data`.