Aligning Point Cloud Views with Point Feature Histograms

Po-Jen Chen, Yu-Ju Chiu
University of Michigan
cpojen, yujuchiu@umich.edu

***Abstract*— In this project, we implement point feature histograms(PFH) for aligning point cloud data views into a consistent global model. Our algorithm estimates robust 8D features describing the local geometry of each point. These features are employed in an initial alignment algorithm to approximate a rigid transformation for registering input datasets. The algorithm offers effective starting points for iterative registration algorithms like Vanilla ICP (Iterative Closest Point) by transforming datasets to its convergence basin, which is ICP-PFH. We also compare our approach with Vanilla ICP, highlighting improvements and differences, particularly in cases unaddressed by Vanilla ICP methods.**

I. MOTIVATION

Robotic sensors, particularly laser rangefinders, commonly generate 3D Point Cloud data, enabling tasks like self-localization, navigation mapping, and object identification. The Iterative Closest Point (ICP) algorithm plays a crucial role in estimating pose changes between consecutive data frames.

However, the ICP algorithm faces challenges in consistently converging to the correct solution. It relies on minimum Euclidean distance for point-to-point correspondences, overlooking the inter-connectedness among points. Using the mug in the point cloud assignment as an example, ICP encounters challenges in achieving accurate alignment when the initial guess is poor [Fig]. Specifically, accurate rotation estimation becomes difficult based on Euclidean distance measurements.

In this project, we address the challenge of formulating a discriminative feature descriptor, denoted as (α, ϕ, θ d), incorporating local geometry in a point's neighborhood and generating feature histograms correspondences. This approach facilitates the identification of the global minimum, enabling us to assess enhancements to ICP performance using the proposed feature descriptor. Anticipated applications of this work include advancements in Virtual/Augmented Reality, 3D Reconstruction, self-driving cars, and various mobile robotics applications.

II. DATASET

There are two data types for point clouds registration, one of which is Synthetic point clouds and the other is Aligning Point Cloud Views.

A. Synthetic point clouds

For such a data set, the features come from the same point cloud, so the surface features and inter-connectedness among points will be exactly the same, and it can be ensured that every point has a corresponding point. To put it simply, this point cloud structure is not a partial overlap, but a full overlap. Therefore, it can usually be solved with traditional ICP, but there will still be problems with pool initial states causing inability to match.

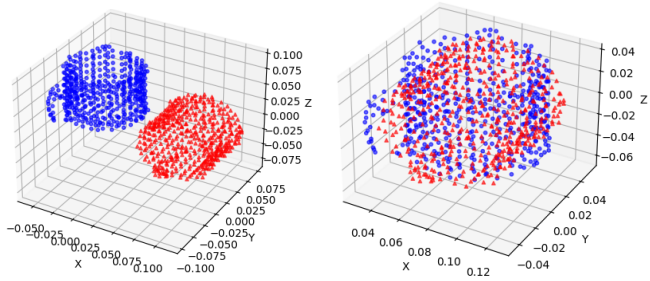


Fig. 1: Vanilla ICP achieve poor alignment

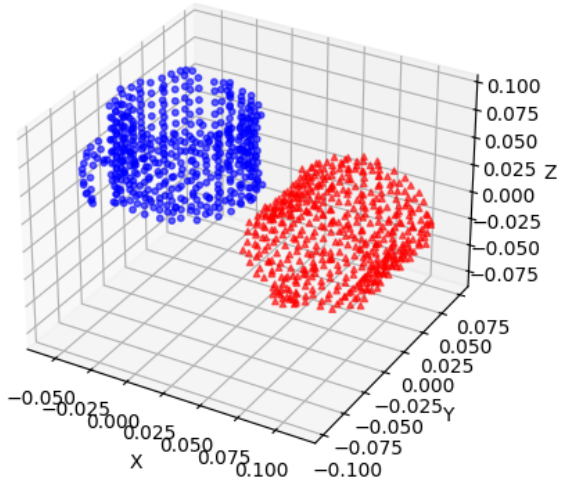


Fig. 2: logic

B. Aligning Point Cloud Views[3]

Another kind of data set only partially overlaps, not fully overlaps. For such data sets, traditional ICP usually cannot solve it, because the overlap part must be found first, and the inter-connectedness relationship must be found first, which is what this report requires. A place of challenge. Take the picture below as an example. The point cloud picture is divided into two parts, 1/3 of which is overlap.

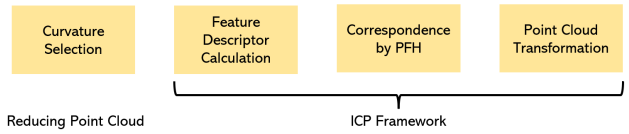


Fig. 4: The framework of ICP with Point Feature Histograms

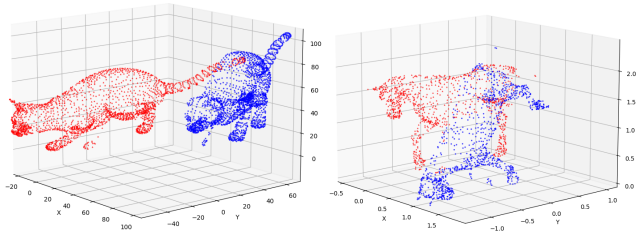


Fig. 3: Vanilla ICP achieve poor alignment

III. METHODOLOGY

Having implemented the Vanilla ICP, we encountered challenges in finding accurate matches. In this project, our objective is to enhance the correspondence search between two point clouds by implementing the Point Feature Histogram descriptor(ICP-PFH). Fig 4 illustrates the pipeline for our approach.

A. Vanilla Iterative Closest Point

Our initial approach is based on a vanilla variant of the ICP algorithm (Algorithm 1 5). This algorithm computes point-to-point correspondences utilizing minimum Euclidean distance as a similarity metric. The process involves incrementally computing transformations through Singular Value Decomposition (SVD), thereby shifting the source point cloud closer to the target point cloud. The termination criteria include the sum of squared errors between the transformed point cloud and the target point cloud falling below a threshold. Furthermore, we have introduced additional termination conditions to account for scenarios involving a lack of progress or reaching the maximum allowable iterations.

B. Reducing Computational Complexity: Curvature Selection

Calculating PFH for all points in the point cloud is inefficient and computationally expensive. Additionally, most points in the point cloud generate similar histograms, making them less expressive and not particularly useful for finding corresponding point pairs. Fortunately, we can address both issues by leveraging curvature.

Assuming that most points in the point cloud lie on a flat or smooth surface, resulting in small curvature, points with large curvature can be considered characteristic points of the point cloud. The curvature around a point can be calculated using the k-nearest neighbors of the point. The curvature equation is shown below.

For a given point x_i, we identify its k-nearest neighbors $x_{i1}, x_{i2}, ..., x_{ik}$, and put those points into a matrix X.

Algorithm 1 Vanilla Iterative Closest Point.

Inputs: P and Q, **Outputs**: R and t
while not Done **do**
  // Compute correspondences
  $C = \emptyset$
  for all $p_i \in P$ **do**
    find the closest $q_i \in Q$
    $C = C \cup \{p_i, q_i\}$
  end for
  // Compute Transforms
  $R, t \leftarrow$ **GetTransform**(C_{pq}, Cq)
  if $\sum_{i=1}^{n} \|RC_{pi} + t) - C_{qi}\|^2 < \epsilon$ **then**
    return R, t
  end if
  // Update all P
  for all $p_i \in P$ **do**
    $p_i = Rp_i + t$
  end for
end while

Fig. 5: The pseudo code of Vanilla Iterative Closest Point

We then perform Singular Value Decomposition (SVD) to find eigenvalues.

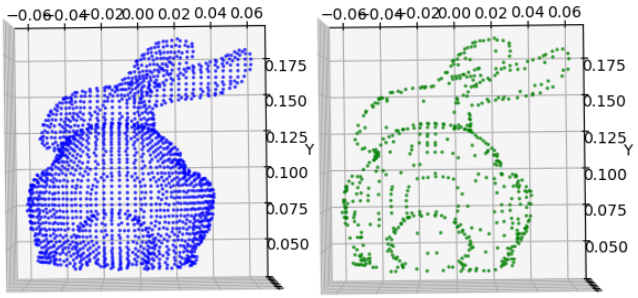
$$U, S, V^T = SVD(XX^T) \tag{1}$$

where $S = \lambda_0, \lambda_1, \lambda_2$ and $\lambda_0 < \lambda_1 < \lambda_2$

Curvature κ can be obtained from the eigenvalues.

$$\kappa = \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2} \tag{2}$$

Once we have κ for the entire point cloud, we can set a threshold κ_{thers} to identify characteristic points and compute PFH for those points.



(a) $\kappa_{thers} = 0$ (Original) (b) $\kappa_{thres} = 0.58$

Fig. 6: Reducing bunny point cloud example

C. Point Feature Histogram

The vanilla ICP algorithm's limitation lies in its point-to-point matching, neglecting the local geometry. The Point Feature Histogram (PFH) from [1] improves this by introducing a pose-invariant feature descriptor, capturing generalized geometric properties of a point's neighborhood. To enhance matching, the vanilla ICP method is adapted to use histogram signature distances as the similarity metric.

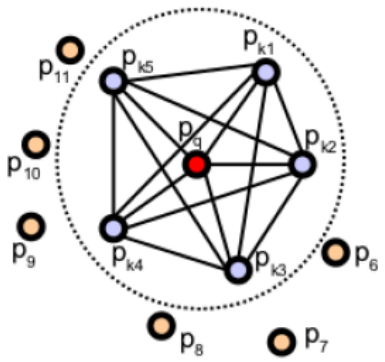


Fig. 7: PFH neighborhood visualization[paper], where p_q has k neighbors p_k within radius r [2]

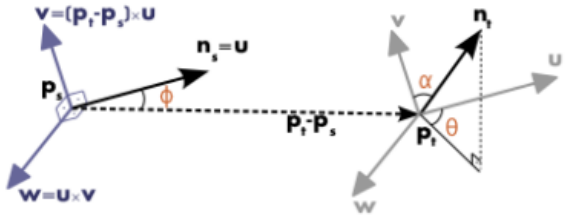


Fig. 8: The computed Darboux frame (vectors u, v and w) placed at the source point where α, ϕ, θ represent f_1, f_3, and f_4.

For PFH formulation, each 3D point requires an estimated surface normal, determined by finding k nearest neighbors, constructing a 3x3 covariance matrix, and selecting the right singular vector. To maintain consistency, all surface normals are oriented towards the viewpoint Fig 7.

Considering a point set P with the query point pi and its k neighbors pj within radius r, features are defined for each unique pair of points, connecting them with lines[Fig]. Source and target points are selected based on the smaller angle between the associated normal and the connecting line.

$$\begin{cases} p_s = p_i, p_t = p_j, & \text{if } n_i \cdot (p_j - p_i) \leq n_j \cdot (p_i - p_j) \\ p_s = p_j, p_t = p_i, & \text{otherwise} \end{cases} \tag{3}$$

and then define the Darboux frame [Fig] with the origin in the source point as:

$$u = n_s, v = (p_t - p_s) \times u, w = u \times v \tag{4}$$

The features $f = (f_1, f_2, f_3, f_4)$ are binned into a div bin histogram, where div is the number of subdivisions for each feature's value range. Fig.8 illustrates these features. In our implementation, we exclude f_2 from binning the histogram as its informativeness diminishes with increasing distance from the sensor. Therefore, our histogram comprises div^3 bins.

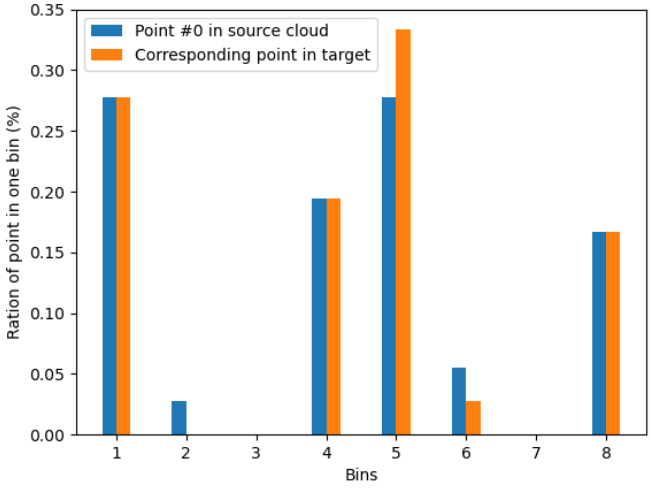


Fig. 9: Point feature histogram for the cat point cloud data

$$\begin{aligned} f_1 &= v \cdot nt \\ f_2 &= \|pt - ps\| \\ f_3 &= \frac{u \cdot (pt - ps)}{f_2} \\ f_4 &= \text{atan}\,(w \cdot nt, u \cdot nt) \end{aligned} \tag{5}$$

An additional aspect of the histogram binning involves creating a 1-dimensional histogram from a multidimensional feature vector f. The histogram consists of div^3 bins, and Rusu offers an equation in equ. 5 to determine the index of the histogram's bins to increment based on f's divisions. The equation for div = 2 is as follows:

$$\text{step}(s, f) = \begin{cases} 0 & \text{if } f < s \\ 1 & \text{otherwise} \end{cases} \tag{6}$$

$$\text{idx} = \sum_{i=1}^{i \leq 3} \text{step}(s_i, f_i) \cdot 2^{(i-1)} \tag{7}$$

Here is a histogram using the cat point cloud data as an example. The fig 10 displays randomly paired points. To calculate the histogram difference, we directly compute the norm value for each bin.

D. L2 norm for Correspondence Search

We utilized the L2 norm between histogram signatures as a straightforward similarity metric for the correspondence search step of ICP. This approach yielded accurate results on our test data. While alternative distance metrics exist, we did not explore them in this study.

IV. RESULTS

A. Synthetic point clouds

TABLE I: Average computation time comparison for mug

Computation Time (s)	$\kappa_{thers} = 0$	$\kappa_{thers} = 0.58$
Per ICP iteration	0.4727	0.1734
computing PFHs	0.2142	0.07423

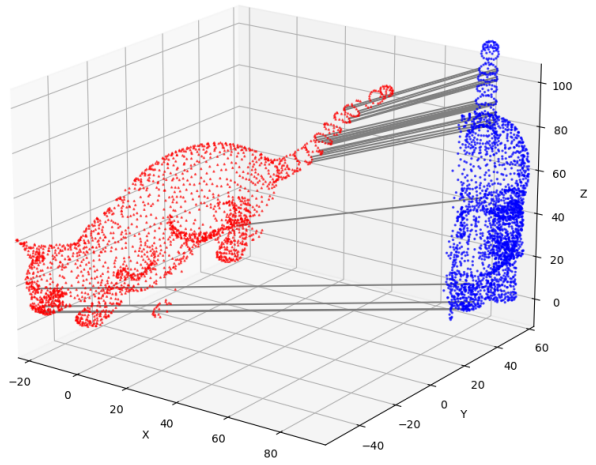


Fig. 10: correspondence pair by L2 norm between histogram signatures

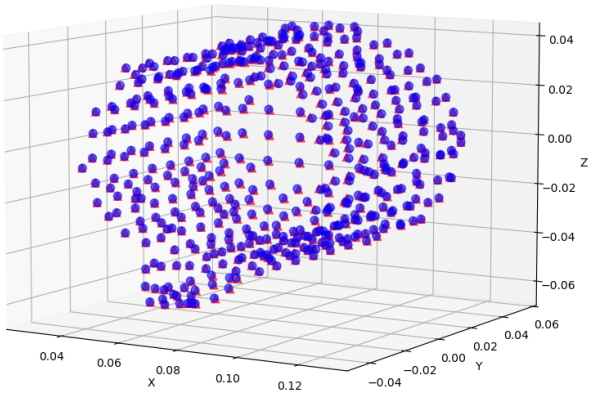


Fig. 11: ICP-PFH result for mug

B. Aligning Point Cloud Views

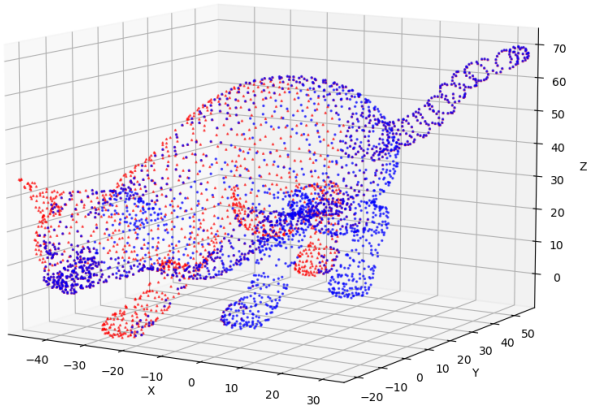


Fig. 12: ICP-PFH result for cat

TABLE II: Average computation time comparison for cat

Computation Time (s)	$\kappa_{thers} = 0$	$\kappa_{thers} = 0.58$
Per ICP iteration	3.1227	1.5287
computing PFHs	1.4624	0.6995

V. Discussion

When we compare the alignment results of ICP-PFH and Vanilla ICP in the images (Fig. 1b, Fig. 13, Fig. 11, and Fig. 12), a noticeable trend emerges: ICP utilizing PFH for finding correspondences demonstrates a higher level of robustness compared to Vanilla ICP using Euclidean distance.

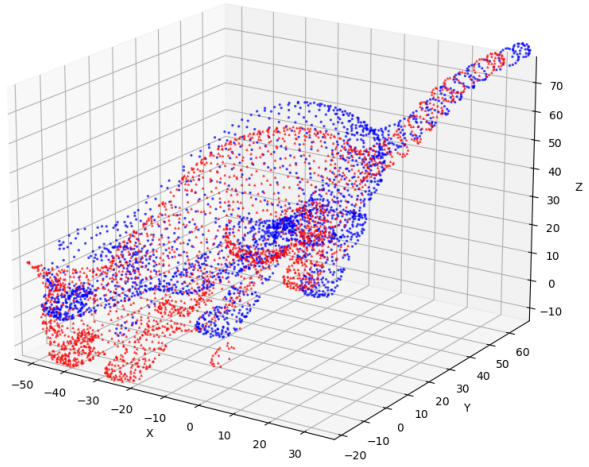


Fig. 13: Vanilla ICP result of cat using Euclidean distance

The key to this enhanced robustness lies in the inherent properties of PFH features, which exhibit pose invariance. In simpler terms, regardless of the initial position or rotation, the ICP algorithm with PFH consistently delivers reliable and accurate alignment results. This means that even when starting from different positions or orientations, the PFH-based ICP method is adept at finding the correct correspondences between points, leading to more stable and accurate alignments.

While ICP-PFH offers increased robustness, its main drawback lies in the computation time. Vanilla ICP requires only 0.001 to 0.004 seconds to complete one iteration (with the number of points in the point cloud ranging from 500 to 3000). In contrast, ICP-PFH takes more than 0.5 seconds for each iteration. Therefore, it is crucial to reduce computational complexity for more real-time and online applications. By setting a threshold to curvature decrease the number of points (as shown in TABLE III, where the number of points is reduced to 10% to 20% by setting $\kappa_{thers} = 0.58$) for computing PFH, the ICP time per iteration drastically decreases (see TABLE I and TABLE II). Additionally, we conducted a computation time comparison between Vanilla ICP, ICP-PFH, ICP-PFH with Curvature Selection.

TABLE III: Comparison between Vanilla ICP, ICP-PFH without Curvature Selection (CS) $\kappa_{thres} = 0$ and ICP-PFH $\kappa_{thres} = 0.58$

(a) Mug

Methods	point num	time per it	success
Vanilla ICP	495	0.002041	No
ICP-PFH w/o CS	495	0.4727	Yes
ICP-PFH w/ CS	48	0.1734	Yes

(b) Cat

Methods	point num	time per it	success
Vanilla ICP	2300	0.004095	No
ICP-PFH w/o CS	2300	3.1227	No*
ICP-PFH w/ CS	441	1.5287	Yes

*with slight offset (Fig. 14)

Moreover, selecting points based on curvature offers additional benefits, such as identifying 'key' points in the point cloud that better represent features compared to other points. Without curvature selection, the results may be unsatisfactory, as the algorithm may attempt to align points with similar PFH across the entire point cloud (see Fig. 14).

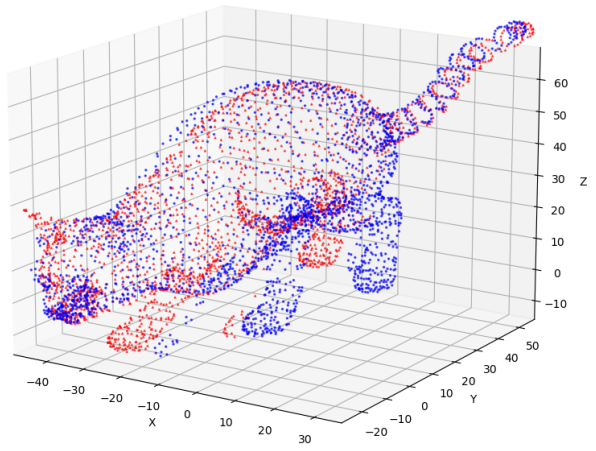


Fig. 14: ICP-PFH result of cat without curvature selection

Note that we utilize distances between PFH to establish correspondences between the source point cloud and target point cloud. However, it's crucial to be careful, as using distance may not always result in accurate correspondences, especially when the two point clouds overlap only slightly. In such scenarios, we need to explore alternative methods to find correspondences, such as matching PFH using KL divergence, to achieve more precise and reliable results. Our future work aims to find a more robust algorithm for finding correspondences between the source point cloud and the target point cloud.

VI. Conclusion

ICP-PFH can provide a satisfactory alignment result even with a poor initial guess, thanks to its pose-invariant property.

Reducing the number of points for improved computational performance and more accurate correspondence finding is crucial, and setting a threshold for curvature is a good starting point.

Exploring alternative methods to find correspondences, such as KL divergence, could potentially enhance the robustness of our algorithm.

References

[1] Rusu, Radu Bogdan, et al. "Aligning point cloud views using persistent feature histograms." 2008 IEEE/RSJ international conference on intelligent robots and systems. IEEE, 2008.

[2] Sphinx, Point Feature Histograms (PFH) descriptors in Point Cloud Library

[3] aichim, holzers, jspricke, pointclouds, sgedikli, Point Cloud Library Files: Standalone, large scale, open project for 3D point cloud processing