ROB 422 Final Project Report
Point Cloud Processing with Innovative Point Feature Histograms Method

Zehan Qin zehanqin@umich.edu **Shicheng Peng** shchpeng@umich.edu

1 Abstract

This paper presents the results of our UMich ROB 422 final project. The topic we chose is Point Cloud processing. Our target aims to implement the Point Cloud Feature Histogram method and come up with an innovative way to reduce computational time. By the end, we will illustrate that our method works on various testing data, including real-world point cloud examples.

2 Motivation and Introduction

The concept of point cloud registration plays a dominant role in modern engineering, robotics, and autonomous driving. It provides the capability of processing 3D data after the data is acquired by LiDAR, RGB, and image sensors. This allows machines and robots to have the ability of "sense" so that they understand the geographical location and shape of the surrounding environment. Therefore, the capability of filtering and processing massive 3D data within a short computational time becomes a key part.

Iterative Closest Point (ICP) is a well-known algorithm that iteratively computes the affine transformation based on the Euclidean distance of closest correspondences. It can achieve precise alignment of matching two clouds with a good initial placement. The downside of this is that it does not take into account of local geometry around the points and possible noise. Therefore, the matching process may fail if the point cloud is in different orientations or with a certain amount of noise. A good example of this is shown in Figure 1. The algorithm had difficulties in matching point clouds if rotation is present. In order to better match the target data, the method of computing point cloud features needs to be implemented.

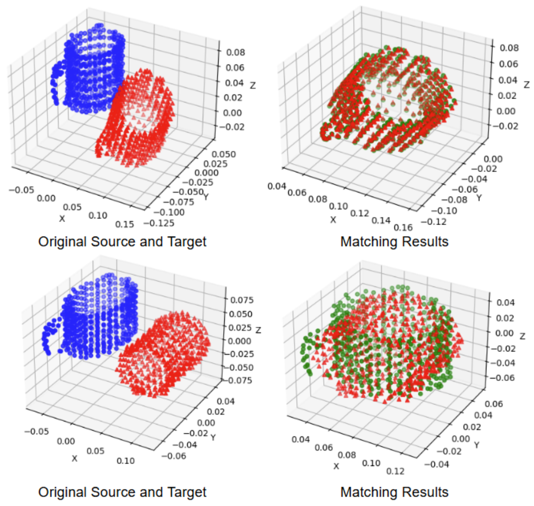


Figure 1: Visualization of failure matching due rotation of the target point cloud.

Point Feature Histograms (PFH) are widely used to facilitate reliable initial alignment in point cloud registration pipelines, thereby improving the convergence behavior of ICP. By encoding angular relationships between surface normals within a local neighborhood, PFH provides a rotation-invariant geometric descriptor that enables coarse but robust correspondences prior to geometric refinement [1]. However, PFH evaluates geometric relations for all point pairs within each neighborhood, resulting in a computational complexity of $O(nk^2)$, where n is the total number of points and k is the number of neighbors of a query point. This quadratic dependence on neighborhood size limits scalability on large or densely sampled point clouds.

One natural strategy to reduce this cost is to decrease the number of points on which PFH is computed. In practice, planar regions and densely sampled areas often contribute redundant information for initial alignment, whereas edges, corners, and locally sparse regions are more informative for matching. Therefore, downsizing and filtering to retain representative points can reduce the total PFH workload while preserving registration performance.

Fast Point Feature Histograms (FPFH) provide an alternative by reducing the per-point cost of descriptor construction. Instead of computing pairwise relations among all neighbors (as in PFH), FPFH first computes a simplified histogram (SPFH) using only relations between the query point and each of its k neighbors, yielding $O(k)$ cost per point. It then incorporates broader neighborhood context by aggregating the SPFHs of neighboring points via distance-weighted combinations, which adds another $O(k)$ term. Consequently, the overall complexity is reduced from $O(nk^2)$ to $O(nk)$ [2], while improving robustness to noise and non-uniform point density. Nevertheless, the distance-weighted aggregation introduces additional constant overhead, which may diminish FPFH's practical advantage under small neighborhood sizes or aggressive point filtering.

Motivated by these observations, we incorporate the neighborhood-aware construction idea of FPFH into a modified PFH formulation that restricts computation to informative point pairs, and investigate whether PFH-based initial alignment can outperform FPFH under such conditions in both computational efficiency and registration accuracy.

we accomplished the following tasks:

1. Utilized voxel downsampling to reduce point cloud size and computational cost.
2. Implemented a density-based filter to remove outliers and preserve salient feature points for PFH/FPFH.
3. Implemented modified PFH, FPFH, and ICP for point cloud registration.
4. Compared computational cost between our density filter and Open3D Intrinsic Shape Signatures (ISS).
5. Evaluated matching algorithms' performance under different feature generation strategies, correspondence criteria, and histogram bin configurations.

In the end, by testing on various real-world point clouds, we found that utilizing voxel downsizing and our density filter with appropriate parameters can significantly reduce the computational time for PFH/FPFH while maintaining the registration success rate. Under these settings, although the improvement is modest, our modified PFH demonstrates consistently better performance than FPFH in both registration accuracy and computational efficiency.

3 Implementation

3.1 *Algorithm Flow Chart*

The general flow chart is shown in Figure 2. After point clouds are imported by our algorithm, they would first get downsized in to relatively small size, then the outliers and noises will be filtered out so that the main featured points will be kept. After feature extraction, PFH/FPFH is used to establish coarse correspondences between the source and target point clouds based on feature histogram similarity, enabling a reliable initial alignment. This initial alignment brings the point clouds into a common reference frame and provides a suitable starting point for refinement. Subsequently, ICP is applied to iteratively refine the alignment by repeatedly establishing correspondences based on Euclidean nearest neighbors and estimating the rigid transformation that minimizes the geometric alignment error. This iterative process continues until convergence or a maximum number of iterations is reached.

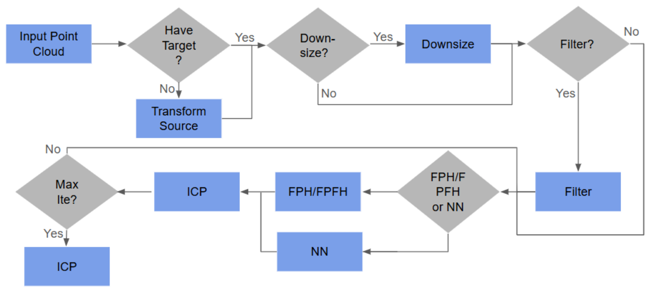


Figure 2: Algorithm Flow Chart.

3.2 *Downsizing*

In order to properly match huge dimension point clouds, the first problem our algorithm faced was downsizing. The issue of how to properly filter out non-critical/redundant points became a crucial point. The first attempt we had was to implement grid-based downsizing. The method discretizes the entire data space into grids and combines the data points that are in the same one. This achieved downsizing data points in a fast way without affecting the general shape of the point cloud.

$$\begin{aligned}
&P \in \mathbb{R}^3 \quad \text{where} \quad P = \{p_1,\ p_2,\ \dots,\ p_n\}\\
&G = \text{Grid Size}\\
&C_i = (C_{x,i},\ C_{y,i},\ C_{z,i}) = \left\lfloor \left(\frac{x}{G},\ \frac{y}{G},\ \frac{z}{G}\right) \right\rfloor\\
&P_{\text{Down}} \in \mathbb{R}^3 \quad \text{where} \quad P_{\text{Down}} = \{p_i \mid c_i\}
\end{aligned}$$

However, later we found out that as the grid size gets bigger, the distribution of the downsized data gets worse. This has a significant impact on later processes, especially for filtering the feature points.

With bad distribution, our density function could not precisely filter out noises. Therefore, the Open3D Voxel method was later utilized in our algorithm to better uniformly downsize the point clouds.

3.3 *Filtering*

After downsizing the point cloud, our intention was to further filter out noise so that it could release the burden of the matching process. The Open3D ISS function is aimed at detecting key points of a 3D shape object [3] based on the Eigenvalue Decomposition (EVD) of the scatter data. Similar to the Principal Component Analysis (PCA), the data with high eigenvalues are kept since they hold the most information. However, when we utilized ISS, we found those parameters for ISS are hard to tune, especially for a high-dimensional point cloud. Due to it requires to tune the threshold of salient radius and eigenvalues without knowing additional information, sometimes, the number of feature points and matching success rate become extremely low without proper parameters.

Therefore, later, we came up with our density filter function. It measures the local sparseness of each point by measuring the mean distance ($\underline{d_i}$) between the query point and k-nearest neighbors (NN). Once all mean distances are gathered, the global mean distance ($\mu_{\underline{d}}$) and global standard deviation ($\sigma_{\underline{d}}$) can also be calculated. After that, the distance threshold ($D_{\max}$) can be set as:

$$D_{\max} = \mu_{\underline{d}} + \gamma \cdot \sigma_{\underline{d}}$$

γ is the standard deviation ratio. With a higher ratio, it means that more than γ times of the standard deviation will be considered as noise, which makes the filter less aggressive. Moreover, in this filter, the number of nearest neighbors also plays an important role as a parameter. If NN is too small, it can lead to filtering out sharp corners. On the other hand, if NN is too big, it may miss filtering outliers.

3.4 *Point Features Histogram (PFH) and Fast Point Features Histogram (FPFH)*

Unlike PFH, which explicitly evaluates geometric relationships for all unordered point pairs within a local neighborhood, FPFH restricts feature computation to pairs formed by the query point and each of its neighbors [2]. This star-shaped interaction pattern avoids the quadratic cost associated with neighbor-neighbor pair evaluations while still capturing local geometric context through subsequent feature aggregation. Inspired by this design, we adopt the same query-neighbor pairing strategy in our modified PFH implementation, limiting feature computation to informative point pairs while preserving the core histogram-based representation of local geometry.

3.4.1 *Features Generation*

Let $p_i = (x_i, y_i, z_i) \in \mathbb{R}^3$ denote a query point with an associated surface normal $\mathbf{n}_i$. For each point p_i, PFH considers a local neighborhood P_k^β consisting of all points p_j such that $\|p_j - p_i\| \le r_\beta$, where r_β is a predefined neighborhood radius and k is the number of neighbors within this radius. If surface normals are not available, they are estimated using Principal Component Analysis (PCA) over a smaller neighborhood P_k^α with radius $r_\alpha < r_\beta$, and consistently oriented using viewpoint information.

For each unordered pair of distinct points (p_{j_1}, p_{j_2}) in the neighborhood P_k^β, the source point p_s is selected as the point whose surface normal forms a smaller angle with the vector connecting the two points. Specifically, given two points p_{j_1} and p_{j_2} with normals $\mathbf{n}_{j_1}$ and $\mathbf{n}_{j_2}$, p_{j_1} is chosen as the source point if

$$\arccos\left(\frac{\mathbf{n}_{j_1} \cdot (p_{j_2} - p_{j_1})}{\|p_{j_2} - p_{j_1}\|}\right) \le \arccos\left(\frac{\mathbf{n}_{j_2} \cdot (p_{j_1} - p_{j_2})}{\|p_{j_1} - p_{j_2}\|}\right),$$

otherwise p_{j_2} is selected as the source point.

A Darboux coordinate frame is then constructed at the source point:

$$\mathbf{u} = \mathbf{n}_s, \quad \mathbf{v} = \frac{(p_t - p_s) \times \mathbf{u}}{\|p_t - p_s\|}, \quad \mathbf{w} = \mathbf{u} \times \mathbf{v}.$$

Using this local frame, three geometric features are computed for each point pair:

$$\begin{aligned} f_0 &= \langle \mathbf{v}, \mathbf{n}_t \rangle, \\ f_1 &= \frac{\langle \mathbf{u}, p_t - p_s \rangle}{f_1}, \\ f_2 &= \operatorname{atan2}(\langle \mathbf{w}, \mathbf{n}_t \rangle, \langle \mathbf{u}, \mathbf{n}_t \rangle). \end{aligned}$$

Features $(f_0, f_1, f_2)_{j_1,j_2}$ of all point pairs (p_{j_1}, p_{j_2}) in the local neighborhood of p_i generate a set of feature values for each feature dimension. Collectively, these values form an empirical distribution that characterizes the local geometric relationships around the query point p_i. Each feature dimension is treated independently, and the corresponding empirical distribution is discretized into histogram bins to approximate the underlying probability distribution of that geometric feature within the neighborhood.

3.4.2 *Bins Discretizations and Histogram Generation*

Each feature is discretized into a fixed number of bins, and the resulting three-dimensional feature tuple is accumulated into a histogram that represents the distribution of geometric relationships within the neighborhood. In our implementation, we support both uniform binning across all feature dimensions and heterogeneous binning schemes in which different feature dimensions are assigned different numbers of bins. This flexibility allows us to further investigate which geometric features are more informative under different point cloud characteristics and noise conditions.

Moreover, we consider two different discretization strategies when assigning feature values to histogram bins. The first strategy uses uniform binning based on equal-width partitioning of the feature range. For a feature value f with theoretical range $[f_{\min}, f_{\max}]$ and d bins, the bin index is computed as

$$\mathrm{idx}_{\mathrm{uniform}} = \left\lfloor \frac{f - f_{\min}}{f_{\max} - f_{\min}} \cdot d \right\rfloor.$$

The second strategy employs percentile-based binning, where bin boundaries are determined by empirical quantiles of the feature distribution. Let $\{q_0, q_1, \ldots, q_d\}$ denote the quantiles corresponding to probabilities $\{0, \frac{1}{d}, \ldots, 1\}$. The bin index is then assigned according to

$$\mathrm{idx}_{\mathrm{percentile}} = \arg\max_i \{ i \mid q_i \le f < q_{i+1} \}.$$

By comparing these binning strategies and bin configurations, we analyze how discretization choices influence feature discriminability and matching performance across different point cloud datasets.

In each bin, we count the number of feature values from the precomputed feature set that fall within the corresponding bin range. To avoid numerical instability in subsequent correspondence measurements, a small constant $\epsilon = 10^{-8}$ is added to bins with zero count. Finally, the raw bin counts are normalized by the total number of feature values in the feature set (including the added ϵ) to convert the histogram into a percentage-based representation.

3.4.3 *FPFH Histogram Generation*

Fast Point Feature Histograms (FPFH) are constructed as an efficient approximation of PFH by reducing the number of point pair interactions within a local neighborhood [2]. For each query point p_i, FPFH computation consists of two stages: the computation of a Simplified Point Feature Histogram (SPFH), followed by a weighted aggregation of neighboring SPFHs.

Given a query point p_i with surface normal $\mathbf{n}_i$, we first identify its local neighborhood $P_k = \{p_j \mid \|p_j - p_i\| \le r\}$. The SPFH of p_i is the histogram computed in sections 3.4.1 and 3.4.2.

The final FPFH descriptor incorporates broader neighborhood information by aggregating the SPFHs of neighboring points:

$$\mathrm{FPFH}(p_i) = \mathrm{SPFH}(p_i) + \sum_{p_j \in P_k} \frac{1}{\omega_{ij}} \mathrm{SPFH}(p_j),$$

where $\omega_{ij} = \|p_j - p_i\|$ is a distance-based weighting factor that emphasizes closer neighbors.

3.4.4 *Correspondence Measurement*

Correspondence measurement quantifies the similarity between feature histograms and is used to establish correspondences between points from different point clouds. Given two normalized histograms

$\mathbf{P} = (p_1, \dots, p_B)$ and $\mathbf{Q} = (q_1, \dots, q_B)$, where B denotes the number of bins and each bin represents a percentage, we evaluate multiple distance and similarity measures.

L1 Distance. The L1 distance moderates the influence of significant large distance in some dimensions and it measures the sum of absolute differences between corresponding bins:

$$d_{\ell_1}(\mathbf{P}, \mathbf{Q}) = \sum_{i=1}^{B} |p_i - q_i|.$$

L2 Distance. The L2 distance is the most common distance measurement and it computes the Euclidean norm of bin-wise differences:

$$d_{\ell_2}(\mathbf{P}, \mathbf{Q}) = \sqrt{\sum_{i=1}^{B} (p_i - q_i)^2}.$$

Chi-Square Distance. The chi-square distance is a robust method and includes idea of relative comparison [4], ensuring the distance is in the same level to be compared, and is defined as:

$$d_{\chi^2}(\mathbf{P}, \mathbf{Q}) = \frac{1}{2} \sum_{i=1}^{B} \frac{(p_i - q_i)^2}{p_i + q_i},$$

Jensen–Shannon Divergence (JSD). Inspired by distribution-based comparison, we also measure histogram similarity using Jensen–Shannon Divergence [5].

$$D_{\mathrm{JS}}(\mathbf{P}, \mathbf{Q}) = \frac{1}{2} D_{\mathrm{KL}}(\mathbf{P} \parallel \mathbf{Q}) + \frac{1}{2} D_{\mathrm{KL}}(\mathbf{Q} \parallel \mathbf{P}),$$

where the Kullback–Leibler divergence is given by

$$D_{\mathrm{KL}}(\mathbf{P} \parallel \mathbf{Q}) = \sum_{i=1}^{B} p_i \log \frac{p_i}{q_i}.$$

Cosine Similarity. Cosine similarity evaluates the angular similarity between two histograms, and it is popular in Machine Learning literature [6]:

$$s_{\cos}(\mathbf{P}, \mathbf{Q}) = \frac{\sum_{i=1}^{B} p_i q_i}{\sqrt{\sum_{i=1}^{B} p_i^2} \sqrt{\sum_{i=1}^{B} q_i^2}}.$$

A larger cosine similarity indicates higher correspondence likelihood.

The resulting PFH descriptor captures the local surface geometry around each point and can be used to establish coarse correspondences between point clouds, making it particularly suitable for guiding initial alignment prior to iterative geometric refinement.

3.4.5 *Initial Alignment*

PFH/FPFH descriptors are used to establish a coarse initial alignment between the source and target point clouds prior to ICP refinement. We repeatedly sample a small set of corresponding point pairs, compute the histogram correspondence, estimate a candidate rigid transformation, and evaluate its alignment error. The transformation with the minimum error is selected as the initial alignment (See Algorithm 1).

Algorithm 1: PFH/FPFH-Based Initial Alignment

```
Input: Source point cloud $S$, target point cloud $T$;
Feature histograms $\mathcal{F}_S, \mathcal{F}_T$;
Number of iterations $N$, sample size $m$
Output: Initial rigid transformation $\mathbf{T}_0$ and error $e_{\min}$
Initialize minimum error $e_{\min} \leftarrow \infty$;
Compute feature correspondences between $S$ and $T$ using histogram distance;
for $i = 1$ to $N$ do
    Randomly sample $m$ correspondence pairs $\{(p_s^{(k)}, p_t^{(k)})\}_{k=1}^{m}$;
    Estimate rigid transformation $\mathbf{T}_i$ from sampled pairs;
    Apply $\mathbf{T}_i$ to transform source points $S$;
    Compute alignment error $e_i$ using nearest-neighbor distances;
    if $e_i < e_{\min}$ then
        $e_{\min} \leftarrow e_i$;
        $\mathbf{T}_0 \leftarrow \mathbf{T}_i$;
return $\mathbf{T}_0, e_{\min}$;
```

4 Results

Our Experience was performed on a Dell XPS 15 laptop, running Ubuntu 24.04 in Oracle Virtual

Machine. The software environment was Python, numpy, Open3d, scipy, and matplotlib. The cup example was provided during the previous homework.

The following testing examples in voxel testing and filter testing are real-world data (ism_train_michael.pcd and room_scan2.pcd) from PointCloudLibrary on Github. The ism_train_michael.pcd has 3400 data points and room_scan2 has 112624 data points.

4.1 *Preprocessing Method*

During the testing of the preprocessing method, FPFH is utilized for all test cases.

4.1.1 *Cup Example*

For the cup example, we decided not to use any preprocessing on both source and target point clouds since the amount of point data is small enough. The result is shown in Figure 3, it only takes 1.5s to finish matching. Compared to previous homework, which only utilized ICP, now, these two point clouds can quickly and precisely match together. The error reached the lower threshold in just one iteration. This clearly shows that FPFH is a better method when the initial transformation contains a high rotation angle.

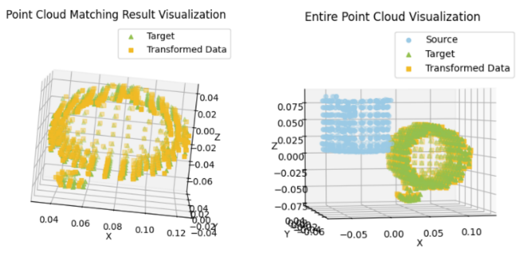


Figure 3: Cup Matching with FPFH.

4.1.2 *Voxel Downsizing*

During the implementation of the downsizing method, we compared the results of using grid-based downsizing and Open3D Voxel downsizing. The result is shown in Figure 4. As the grid/voxel size gets bigger, the number of points after downsizing decreases. With a larger grid size, Open3D's distribution of downsized models slightly outperformed grid-based downsizing. This has a huge impact on the density filter process.

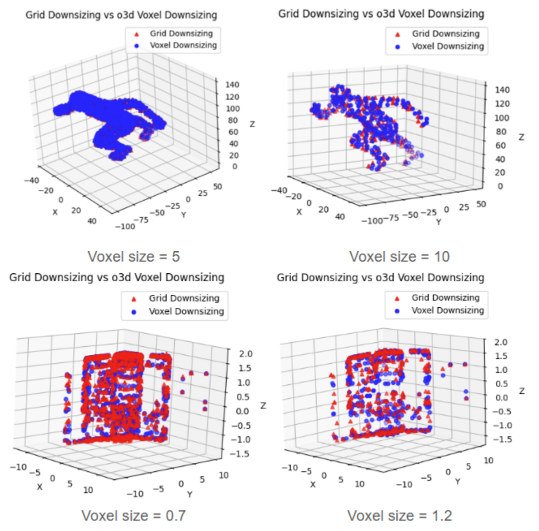


Figure 4: Comparison of grid-based and voxel downsizing.

By doing those tests, we also found that, for Michael's example, with the voxel size changed from 5 to 10, the data points reduced from 916 to 242. The total computational time also reduced from 6.9306s to 1.0922s. Moreover, for the Room's example, with voxel size changed from 0.7 to 1.2, the data points reduced from 989 to 381. The total computational time also reduced from 18s to 3s.

4.1.3 *Feature Filtering*

For understanding the performance of our density filter, we compared the outcome with utilizing Open3D's ISS function, the result is shown in Table 1. As the number of downsized points decreases, the ISS function becomes less stable; meanwhile, our density filter can maintain a high success rate without changing the parameters. Moreover, with the visualization of selected feature points shown in Figure 5, the density filter shows better results.

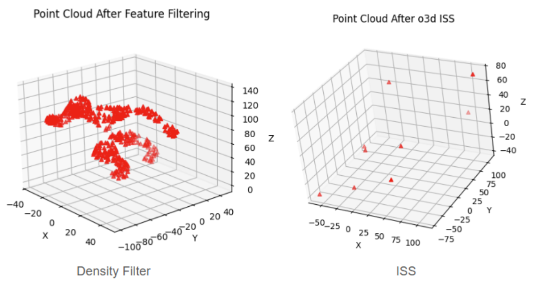


Figure 5: Density Filter (Voxel = 5, NN = 20, Ratio = 0.001) vs ISS for Michael.

Filter	NN	Ratio	Feature Points	Success	Downsize+Filter Time (s)	Matching Time (s)	Total Time (s)
Density (Voxel=5, NN=20, Ratio=0.001)	20	0.001	452	High	0.034	6.8966	6.9306
ISS	#	#	9	Median	0.017	0.2665	0.2835
Density (Voxel=10, NN=20, Ratio=0.001)	20	0.001	173	High	0.01	1.0822	1.0922
ISS	#	#	2	Low	0.008	0.0715	0.0795

Table 1: Density Filter vs ISS for Michael.

Cloud	NN	Ratio	After Filtered Remaining Points	Downsize+Filter Time (s)	Matching Time (s)	Total Time (s)
Michael	20	0.001	173	0.01	1.0822	1.0922
Michael	50	0.001	160	0.012	0.9236	0.9356
Michael	20	1.5	225	0.007	1.6799	1.6869
Michael	50	2	230	0.0153	1.8221	1.8374
Room	20	0.001	323	0.02	3.3821	3.4021
Room	50	0.001	305	0.02	3.0467	3.0667
Room	20	1.5	362	0.0206	4.126	4.1466
Room	50	2	363	0.0234	4.272	4.2954

Table 2: Density Filter with the same voxel grid in each experiment.

Moreover, we tested the impact of changing our density filter parameters on total computational time. During the filter testing, we keep the voxel size the same for each of the samples. By tuning NN and the standard distribution ratio, we wanted to observe how total time was changing while maintaining the same success rate. The result is shown in Table 2. This shows that by fine-tuning the parameters, the total computational time can be reduced even further.

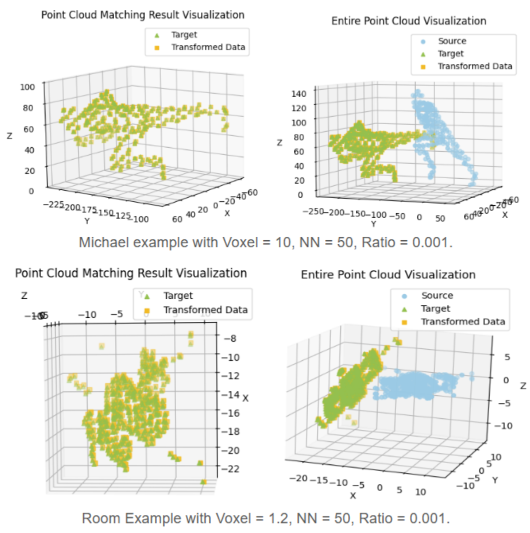


Figure 6: Michael and Room Example Matching Results.

In addition, we wanted to find out whether our method of preprocessing data (voxel + density filter) was actually helping reduce computational time for FPFH. We compared the difference between tests with only FPFH and the ones with both preprocessing methods and FPFH. The matching results are shown in Figure 6. The acquired total computational time and success rate information are obtained in Table 3. The computational time reduced from 37s to 1s for Michael's example, which is nearly a 98% reduction. In the Room's example, the time reduced from 45s to 3s, which is nearly an 93% reduction. The difference in computational time clearly shows that the preprocessing method significantly saves the cost for FPFH.

4.2 *PFH/FPFH Parameter Testing*

All results are obtained from six real-world point cloud datasets and are reported as averages across datasets. The baseline configuration uses **FPFH with six bins per feature, percentile-based binning, and the l_2-norm for correspondence measurement**. All tests are based on matching a source to its target with **partially overlap**.

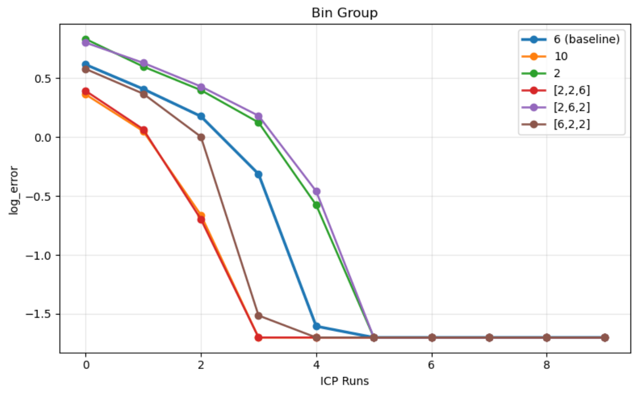
Cloud	Voxel	Downsize	Filter	NN	Ratio	Feature Points	Success	Downsize+Filter Time	Matching Time	Total Time
Michael	5	916	Null	#	#	#	High	0	37.5629	37.5629
Michael	5	916	Den	20	0.001	452	High	0.034	6.8966	6.9306
Michael	10	242	Den	50	0.001	160	High	0.012	0.9236	0.9356
Room	0.7	989	Null	#	#	#	Median	0	45.5492	45.5492
Room	0.7	989	Den	20	0.001	736	High	0.044	18.1581	18.2021
Room	1.2	381	Den	50	0.001	306	High	0.02	3.0467	3.0667

Table 3: Computational Difference for two examples.

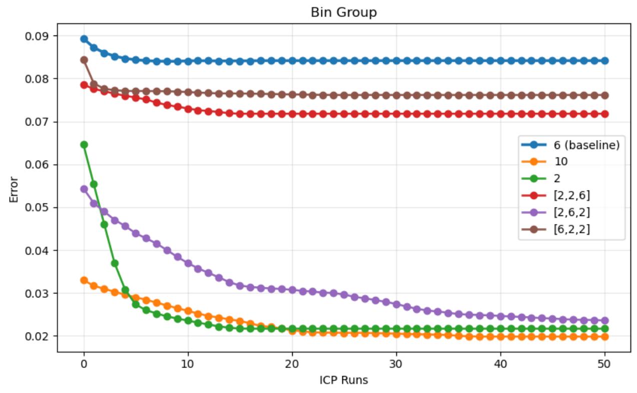
Table 4: Effect of histogram bin configurations on matching performance (relative to baseline).

Bin Configuration	2	6 (Baseline)	10	[6,2,2]	[2,6,2]	[2,2,6]
Accuracy Change (%)	-1897.42	0.00	15.28	1.90	-1924.42	-1911.38
Matching Time Change (%)	-89.66	0.00	373.27	-83.00	-82.61	-83.16

4.2.1 *Numbers of bins*



(a) Log error convergence on general point clouds



(b) Error convergence on planar-dominant point clouds

Figure 7: ICP error convergence under different bin configurations.

We evaluate the impact of histogram bin granularity on matching performance by varying the number of bins used for feature discretization. Specifically, we test uniform bin configurations with 2, 6 (baseline), and 10 bins for all feature dimensions. To further isolate the contribution of individual features, we conduct a controlled analysis in which one feature dimension uses 6 bins while the remaining two use 2 bins, resulting in configurations $[6, 2, 2]$, $[2, 6, 2]$, and $[2, 2, 6]$.

Table 4 summarizes the relative performance changes with respect to the baseline setting (FPFH with 6 bins per feature). All values are reported as percentage changes.

Overall, increasing the number of histogram bins improves matching accuracy by providing a finer discretization of local geometric features. However, this benefit comes at the cost of increased computational overhead, as reflected by the substantial rise in matching time when using 10 bins.

From Figure 7a, we observe that under general, geometrically rich point clouds, finer discretization of the angular features f_0 and f_2 yields the most significant performance gains. These features encode relative normal orientations, which are more informative in highly non-planar and irregular surfaces.

In contrast, Figure 7b shows that for planar or structurally repetitive point clouds, discretization of the distance-related feature f_1 plays a more critical role. In such scenarios, geometric variation is limited, and angular features become less discriminative. Interestingly, configurations with either very coarse or very fine binning on f_1 outperform moderate discretization. This behavior suggests that intermediate bin granularity may amplify noise or ambiguity in distance distributions, whereas extreme binning either smooths out noise or captures subtle but consistent geometric cues.

These observations highlight that the optimal bin

Table 5: Effect of correspondence measurement on matching performance (relative to baseline).

Metric	ℓ_1	ℓ_2 (Baseline)	Chi-square	JSD	Cosine
Accuracy Change (%)	2.11	0.00	-557.24	1.16	0.16
Time Change (%)	-2.35	0.00	124.70	314.86	-93.37

configuration is highly dependent on the geometric characteristics of the underlying point cloud, and that heterogeneous binning can provide a flexible trade-off between accuracy and computational efficiency.

4.2.2 *Binning Method*

The performance difference between percentile-based binning and equal-width binning is marginal. Equal-width binning achieves a slight improvement in accuracy of approximately 0.15%, at the cost of an increased matching time of about 1.23%.

4.2.3 *Correspondence Measurement*

We evaluate the impact of different correspondence measurement strategies on matching performance by comparing five distance or similarity metrics applied uniformly across all feature dimensions: ℓ_1, ℓ_2, chi-square distance, Jensen–Shannon Divergence (JSD), and cosine similarity. All configurations use identical feature generation and binning settings, and performance is reported relative to the baseline configuration, which employs the ℓ_2-norm.

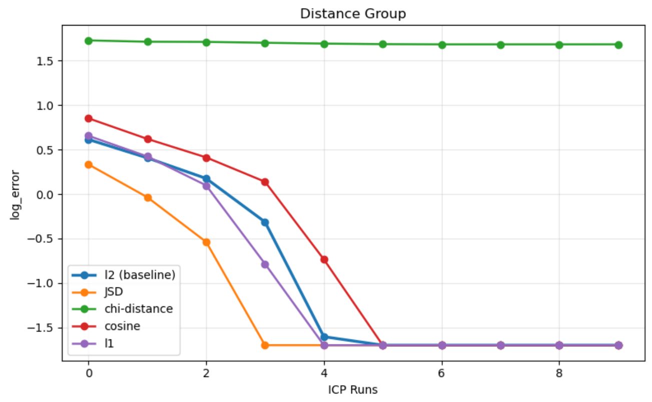


Figure 8: Log error convergence versus ICP iterations under different correspondence measurement strategies.

As shown in Table 5, correspondence measurement has a noticeable impact on registration accuracy. Overall, the accuracy ranking follows

$$\ell_1 \approx \text{JSD} > \text{Cosine} > \ell_2 > \text{Chi-square}.$$

This trend suggests that treating histogram differences across all bins with equal importance plays a critical role in effective correspondence estimation. Both ℓ_1 distance and JSD directly compare full empirical distributions, allowing all three geometric features to jointly contribute to discrimination. In contrast, chi-square distance excessively amplifies discrepancies in low-frequency bins, which can be dominated by noise and lead to unstable correspondences.

Cosine similarity, while robust to global scale variations, primarily captures directional agreement between histograms and may overlook magnitude differences that encode meaningful geometric information. The inferior performance of the ℓ_2 metric further indicates that squared penalties can overemphasize large bin deviations and reduce robustness in the presence of noise.

From a computational perspective, JSD incurs substantially higher overhead due to the evaluation of logarithmic operations and the lack of efficient vectorized acceleration comparable to ℓ_1, ℓ_2, or cosine similarity. As a result, although JSD provides competitive accuracy, its practical applicability is limited when computational efficiency is a primary concern.

4.2.4 *Modified PFH and FPFH*

We compare our modified PFH against FPFH under identical experimental settings to evaluate their relative performance in initial alignment and subsequent refinement. Using FPFH as the baseline, our modified PFH achieves a 2.39% improvement in registration accuracy while reducing feature matching time by 2.47%. In addition, the improved initial alignment quality leads to faster convergence during the ICP refinement stage. Compared to FPFH, our method further reduces the ICP computation time by 1.45%.

Although the overall performance gains are modest, the results consistently indicate that restricting feature computation to informative point pairs can improve both efficiency and alignment quality. These findings suggest that, under practical settings with filtered or moderately sized neighborhoods, carefully designed PFH-based descriptors remain competitive with FPFH while incurring lower computational overhead.

5 Conclusion

5.1 *Preprocessing Method*

After performing the comparison between grid grid-based method and Open3D's voxel downsizing method, we found that when the grid/voxel size is small, both outcomes are relatively the same. However, when the grid size becomes larger, the output point cloud's distribution is slightly worse for grid grid-based method compared to voxel downsizing. The main reason is that the grid method uses a flooring algorithm, while the voxel method uniformly downsizes. Moreover, we compared the matching outcome between our density filter and Open3D's ISS function. We found out that the more point data the input cloud has, the more stable the ISS function becomes. If the data points decrease to a certain low level, ISS would only output several keypoints, which would cause failure during the matching processes. On the other hand, our density filter outperformed the ISS function with a much higher success rate and easier parameters to tune. In addition, we tested the impact of implementing our preprocessing method. The computational time was drastically reduced by utilizing downsizing and density filter. What's more, the preprocessing method can also improve the success rate and matching error when the point cloud size gets bigger compared to the method of only using FPFH.

5.2 *PFH Setting*

In this report, we studied PFH-based feature descriptors for point cloud registration, focusing on feature discretization, correspondence measurement, and descriptor construction. Experiments on multiple real-world point cloud datasets show that both histogram bin configuration and distance metrics significantly affect matching accuracy and computational efficiency.

By adopting the query–neighbor pairing strategy of FPFH while preserving the core PFH formulation, our modified PFH consistently outperforms FPFH under identical settings, improving registration accuracy and reducing both feature matching and ICP refinement time, albeit with modest gains.

Overall, these results indicate that restricting feature computation to informative point pairs, together with appropriate discretization and correspondence measurement, offers a practical trade-off between accuracy and efficiency. This study further suggests that hybrid PFH-based designs remain a promising direction for efficient and robust point cloud registration.

References

[1] 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*, 2008, pp. 3384–3391.

[2] 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*, 2009, pp. 3212–3217.

[3] Open3D Team, "Intrinsic shape signatures (iss) — open3d documentation," https://www.open3d.org/docs/latest/tutorial/Advanced/iss_keypoint_detector.html, accessed: 2025-12-14.

[4] V. Asha, N. Bhajantri, and P. Nagabhushan, "Glcm-based chi-square histogram distance for automatic detection of defects on patterned textures," *International Journal of Computational Vision and Robotics*, vol. 2, no. 4, p. 302, 2011. [Online]. Available: http://dx.doi.org/10.1504/IJCVR.2011.045267

[5] J. K. Hoyos-Osorio and L. G. Sanchez-Giraldo, "The representation jensen-shannon divergence," 2024. [Online]. Available: https://arxiv.org/abs/2305.16446

[6] F. Tessari, K. Yao, and N. Hogan, "Surpassing cosine similarity for multidimensional comparisons: Dimension insensitive euclidean metric," 2025. [Online]. Available: https://arxiv.org/abs/2407.08623