Efficient Point Cloud Alignment Using Point Feature Histograms and Iterative Closest Point

Yicheng Jiang
Robotics
University of Michigan - Ann Arbor
Ann Arbor, United States
valeska@umich.edu

Zijie Chen
Robotics
University of Michigan - Ann Arbor
Ann Arbor, United States
chenzj@umich.edu

***Abstract*—This project explores point-cloud processing by integrating two key techniques: Iterative Closest Point (ICP) and Point Feature Histogram (PFH). These methods are used for robust feature-based matching and alignment. The objective is to improve accuracy and efficiency in point registration tasks.**

I. Introduction

The field of robotics often relies on point cloud processing to interpret and interact with the environment. Accurate and efficient point cloud analysis is crucial for tasks such as object recognition, spatial mapping, and localization, which are essential for the development of autonomous systems.

Iterative Closest Point (ICP) is a widely-used algorithm that finds point correspondences between a source model and a target scene to align the two point clouds. The algorithm iteratively minimizes the distance between the corresponding points by optimizing a transformation consisting of rotation and translation. This makes ICP highly effective for tasks such as registering scans taken from different viewpoints, enabling the creation of a unified 3D model.

Despite its robustness, ICP has limitations, including sensitivity to initial alignment and potential convergence to local minima. To address these issues, feature-based methods such as the Point Feature Histogram (PFH) are often used in conjunction with ICP. PFH provides a geometric descriptor that captures the local geometry of a point and its neighbors, making it useful for identifying the corresponding regions between point clouds.

By combining ICP's alignment capabilities with PFH's robust feature matching, this project aims to improve processing efficiency and overcome the challenges of misalignment under various conditions, such as partial overlaps.

II. Implementation

A. Iterative Closest Point (ICP)

As implemented in the homework assignment, the ICP algorithm uses a Euclidean distance metric to iteratively match points from the source point cloud to the target point cloud. The complete algorithm is described in Algorithm 1.

The process begins with an initial guess for the transformation between the source and target point clouds, which includes both translation and rotation. In practical scenarios, a heuristic is often employed to estimate the initial guess. However, in the homework assignment, the default pose of the source model was directly used as the starting point.

Algorithm 1: Iterative Closest Point (ICP)

Input: P and Q (not necessarily the same size)
Output: P aligned to Q
Place P at some initial pose
while not Done **do**
  `// Compute Correspondences`
  $C \leftarrow \emptyset$
  for all $p_i \in P$ **do**
    Find the closest q_i
    $C \leftarrow C \cup \{p_i, q_i\}$
  end for
  `// Compute Transform with SVD`
  $R, t \leftarrow \text{GetTransform}(C_p, C_q)$
  if $\sum_{i=1}^{n} \|(Rp_i + t) - c_{q_i}\|^2 < \epsilon$ **then**
    return P
  end if
  `// Update all` P
  for all $p_i \in P$ **do**
    $p_i \leftarrow Rp_i + t$
  end for
end while

For each point in the source cloud, the algorithm identifies the closest point in the target cloud using a nearest-neighbor search based on the Euclidean distance metric. After establishing point correspondences, the algorithm computes the optimal rigid transformation (rotation and translation) that minimizes the sum of squared distances between the matched points. This transformation is typically calculated using Singular Value Decomposition (SVD).

The computed transformation is then applied to the source point cloud by aligning it incrementally closer to the target. The algorithm checks for convergence by evaluating whether the change in mean square error between iterations falls below a predefined threshold. If the convergence criteria are not met, the algorithm iteratively repeats the steps of point matching, transformation estimation, and transformation application.

This iterative process continues until the alignment meets the convergence criteria, which ensures a robust and refined alignment of the point clouds.

B. Point Feature Histogram (PFH)

Since the drawback of ICP algorithm lies in using a simple way to estimate correspondences of the points from source to target, the PFH is applied to improve the quality of the point correspondences based on the features of the point cloud. The PFH is a feature descriptor used to capture the local geometric properties of a point and its surrounding neighbors in a point cloud.

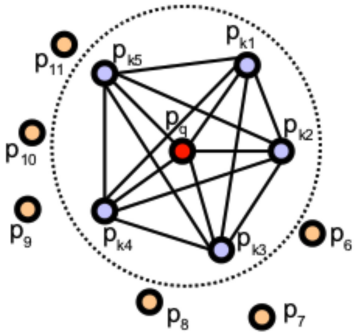


Fig. 1: Visualization of k neighbors within radius r in PFH. Adapted from [2].

The process begins with the estimation of surface normals for each point. This is achieved by identifying a set of neighboring points within a predefined radius, as illustrated in Fig. 1, and fitting a plane to these points to compute the normal vector. Principal Component Analysis (PCA) with Singular Value Decomposition (SVD) was applied to identify the direction of minimum variance among the data points, corresponding to the plane's normal vector. It is essential that the estimated normals are consistently oriented in the same direction. To ensure this, we reorient the normals so that they point away from the mean position of the points. For each point p, a vector v is computed from p to the mean of the entire dataset. Subsequently, for the normal vector n associated with p, the following operation is applied:

$$n = \begin{cases} -n & \text{if } n \cdot v > 0, \\ n & \text{otherwise.} \end{cases} \tag{1}$$

For every pair of points p_i and p_j in the k-neighborhood of p within radius r, a source point p_s and a target point p_t [1] are selected based on their corresponding normals n_i and n_j. The source is defined as the point with the smaller angle between its associated surface normal and the line connecting the two points. The target refers to the other point in the pair:

$$p_s = \begin{cases} p_i & \text{if } acos(\langle n_i, p_j - p_i \rangle) \leq acos(\langle n_j, p_i - p_j \rangle), \\ p_j & \text{otherwise.} \end{cases} \tag{2}$$

After selecting the source and target points in each pair, the Darboux frame [1] can be defined with the origin as the source point p_s:

$$\begin{aligned} \mathbf{u} &= n_s \\ \mathbf{v} &= (p_t - p_s) \times \mathbf{u} \\ \mathbf{w} &= \mathbf{u} \times \mathbf{v} \end{aligned} \tag{3}$$

With the Darboux frame, a set of four features [1] can be quantified and computed for p_s and p_t:

$$\begin{aligned} f_1 &= \mathbf{v} \cdot n_t \\ f_2 &= \|p_t - p_s\| \\ f_3 &= \frac{\mathbf{u} \cdot (p_t - p_s)}{f_2} \\ f_4 &= atan(\mathbf{w} \cdot n_t, \mathbf{u} \cdot n_t) \end{aligned} \tag{4}$$

In robotics, 2.5D data are commonly acquired, where the spacing between points increases with distance from the viewpoint [2]. In scenarios where point density varies with distance, excluding the second feature f_2, which represents the Euclidean distance between two points, is advantageous, as these distances do not accurately represent intrinsic features.

The values of f_1 and f_3 lie within the range of ± 1, as they are derived from dot product computations. The value of f_4 falls within the range of $\pm \frac{\pi}{2}$ due to its dependence on the arctangent function.

To represent these three features more compactly, their values are categorized and grouped into histograms [1]. For instance, when categorizing values into two groups, the threshold s_i is defined as the midpoint of the range of f_i. Accordingly, a *step*(s_i, f_i) function is defined as:

$$step(s_i, f_i) = \begin{cases} 0, & \text{if } f_i < s_i, \\ 1, & \text{otherwise.} \end{cases} \tag{5}$$

Assuming we are categorizing all four features, the index *idx* that represents the category of f_i is then calculated as:

$$idx = \sum_{i=1}^{i \leq 4} \text{step}(s_i, f_i) \cdot 2^{i-1} \tag{6}$$

This equation uses two divisions (*div* = 2) for the feature values. To apply more divisions, replace the value 2 in Eq. 6 with the desired value of *div*. Consequently, the histogram contains div^4 bins when all four features are included, or div^3 bins when the second feature, f_2, is excluded. Performance comparisons for different values of *div* are provided in Section IV.

The value of each feature for every point in the k-neighborhood is processed using the described approach to determine its corresponding *idx* in the histogram. The determined *idx* is incremented by 1. For a query point p, each pair of p_i and p_j within the k-neighborhood of p contributes to generating a histogram. By aggregating all histograms within the k-neighborhood and normalizing the result, a feature histogram for the point p is obtained.

Instead of using L2 norm, we adopted the Chi-squared distance as the similarity metric because it emphasizes differences between bins relative to their combined magnitude, making it effective for comparing normalized histograms while reducing the influence of bins with small values. For two histograms, A and B, their Chi-squared distance is calculated as:

$$\chi^2(A,B) = \sum_i \frac{(A_i - B_i)^2}{A_i + B_i} \tag{7}$$

C. Fast Point Feature Histogram (FPFH)

FPFH is an optimized variant of PFH and offers improved computational efficiency [2]. PFH generates a detailed feature histogram for a query point p by computing and aggregating features for every pair of points within the k-neighborhood of p. This process is illustrated by the solid lines connecting points inside the dotted circle in Fig. 1. When n neighbors are used for constructing the PFH, the computational complexity is $\mathcal{O}(k^2)$. In comparison, FPFH employs a simplified approach to compute feature descriptors.

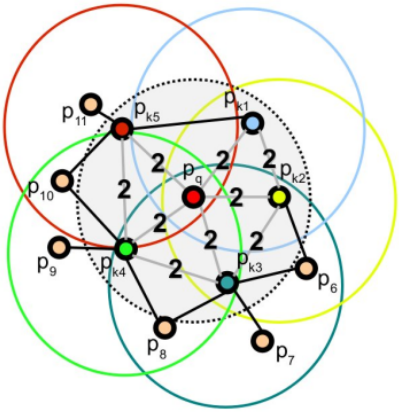


Fig. 2: Visualization of constructing a FPFH for query point p_q. Adapted from [2].

First, the Simplified Point Feature Histogram (SPFH) is introduced. For a given point p, SPFH considers only the pairs formed between the point itself and its k nearest neighbors, resulting in k pairs used to construct a simplified feature histogram for p. The Fast Point Feature Histogram (FPFH) algorithm extends this by applying SPFH to each of the k-nearest neighbors of p. The resulting SPFH values of the neighbors are then weighted and combined to compute the final histogram for p. A diagram illustrating the FPFH construction is provided in Fig. 2, and its computation is formally defined as follows:

$$FPFH(p) = SPFH(p) + \frac{1}{k}\sum_{i=1}^{k}\frac{1}{\omega_i}\cdot SPFH(q_i) \tag{8}$$

In Eq. 8, the weight ω_i denotes the distance between the query point p and its neighboring point p_i, calculated using a specified distance metric. In our case, we used the Euclidean distance because it is simple and fast.

Compared to the $\mathcal{O}(k^2)$ computational complexity of PFH, FPFH reduces the complexity to $\mathcal{O}(k)$ while retaining most of the descriptive power of PFH [2]. A runtime comparison of PFH and FPFH is provided in Section IV.

D. Pipeline

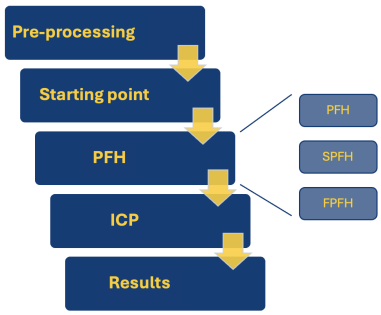


Fig. 3: Steps of our processing algorithm.

As shown in Fig. 3, our processing algorithm begins with the pre-processing of raw experimental data and concludes with results that maximize the correspondence between the target and source. Within this pipeline, we employ a variant of the PFH and ICP algorithms sequentially to compute and execute the alignment. The PFH variant determines point correspondences based on feature descriptors and performs an initial coarse alignment, while ICP refines the alignment by iteratively identifying corresponding points using Euclidean distance. Both methods establish point-to-point correspondences and calculate transformations by minimizing the Euclidean distance between corresponding pairs. The PFH ensures robust global alignment, whereas ICP achieves precise local refinement.

Another key feature we add to PFH is feature extraction based on curvature. A threshold for curvature can be predefined, and the curvature is calculated as:

$$\kappa = \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2} \tag{9}$$

where λ_i are eigenvalues obtained from Singular Value Decomposition (SVD), and $\lambda_0 < \lambda_1 < \lambda_2$.

If a point p has a curvature $kappa$ exceeding a specified threshold, p is categorized as a feature point. Feature points are queried to compute PFH or FPFH, and their correspondences are established with feature points in the target. Conversely, if $kappa$ is below the threshold, p is categorized as a regular point. Regular points are processed similarly to the steps in standard ICP. The goal is to optimize computation by filtering out unnecessary points during histogram calculation.

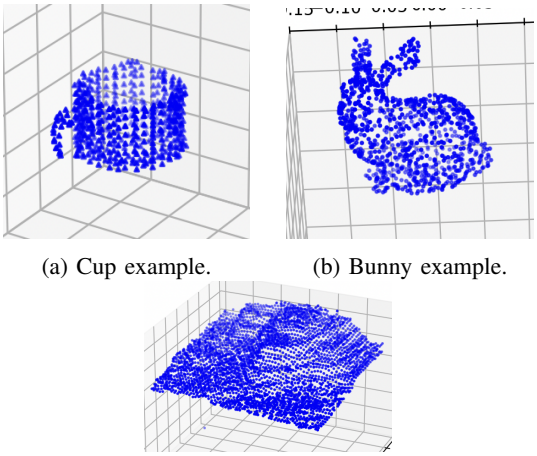
However, the filtered points are not disregarded entirely, as they contain global information, whereas feature points capture local geometric details. Consequently, finding correspondences for regular points remains essential, but is performed using Euclidean distance. Additionally, feature points in the target are exclusively reserved for establishing correspondences with

feature points in the source. Once correspondences are determined for all feature points, regular points can establish correspondences with the remaining available points.

The described method for curvature-based filtering is integrated into the PFH variant within the pipeline. The effect of adjusting the curvature threshold on performance is analyzed in Section IV.

III. Dataset and Assumptions

The dataset we used in our testing consists of generally two sources, point clouds generated by the mesh of 3D models with noise added, a cup and a bunny, and point clouds from the real terrain, originally formatted as .pcd file, shown in Fig 4. The cup point cloud is adapted from the course *EECS 465/ROB 422: Introduction to Algorithmic Robotics*, instructed by Prof. Dmitry Berenson. The reconstructed models of the Stanford bunny was obtained from the Stanford 3D Scanning Repository [3]. The terrain point cloud is retrieved from the GitHub repository provided by the Point Cloud Library (PCL) [4].

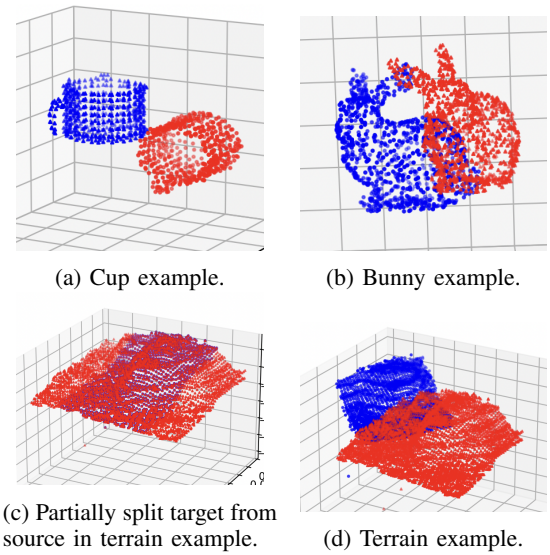


(a) Cup example. (b) Bunny example.

(c) Terrain example.

Fig. 4: The cup, bunny and terrain 3D pointcloud examples.

To test the performance of our method, also to make the comparison between each method, we split the testing examples into two groups. First group called *fully-overlapped group* consists of the cup and the bunny, and terrain in the second group called *partially-overlapped group*. As for the first group, we duplicate the source with rotating, translating and noise added as the target, demonstrated in Fig 5a, 5b, where the goal is to coincide the target and the source. The second group however, designed to be more complex and more challenging for the algorithm, where the point cloud is split into two parts that are partially overlapped with larger data volume, depicted in Fig 5c, and randomly place the target apart from the source, shown in Fig 5d. In all examples, the target is rendered in red, while the source is in blue. We visualize all the results and also compare the performance of each algorithm by the total error and run time.



(a) Cup example. (b) Bunny example.

(c) Partially split target from source in terrain example. (d) Terrain example.

Fig. 5: Starting point of the source and target used in testing.

IV. Results

A. Experimental Settings

Our experiments were conducted on a 2022 MacBook Air, with Apple M2 chip, 8 GB memory. The software environment used is Python 3.11.10, numpy 2.1.3, matplotlib 3.9.3, scipy 1.14.1. The cup example is originally in .csv format, no need for pre-processing. The bunny example is in .ply format, which needs to generate points according to the mesh of the 3D model. The terrain example is in .pcd format, where downsampling and format conversion are used to get point clouds in valid format in pre-processing.

B. Variables Declaration

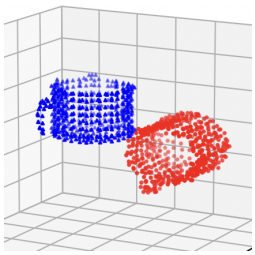
Variable	Description
div	The size of histogram bin.
k	Number of neighbors of a certain point.
χ	Percentile of featured points used by PFH.
r	Radius of seeking for the neighbors.
t	Total runtime of the algorithm.
ϵ	Total error of the result with the original target.

TABLE I: Definition of Variables Used in the Model

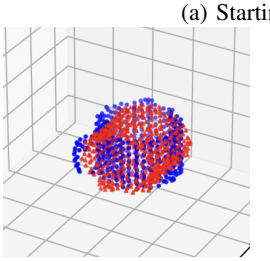
C. Test on fully overlapped examples

We applied ICP and FPFH-ICP respectively to both the cup and the bunny example to compare the performance between these two methods, or the necessity of the use of PFH algorithm, as shown in Fig 6 and 7. From the same starting pose for each example, in Fig 6a and 7a, both of the examples turns out that the result of the ICP is not satisfactory in Fig 6b and 7b, while FPFH-ICP correctly matches the target

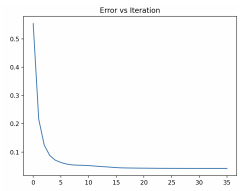
with the source in Fig 6d and 7d. Therefore, it proves that the application of PFH greatly enhances the performance of ICP.



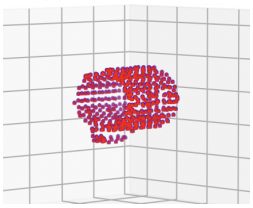
(a) Starting point.



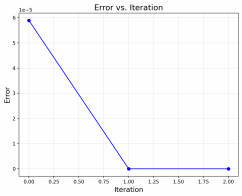
(b) Result of ICP.



(c) Error of ICP.



(d) Result of FPFH-ICP.



(e) Error of FPFH-ICP.

Fig. 6: Comparison between ICP and FPFH-ICP in the cup example.

D. Test on partially overlapped examples

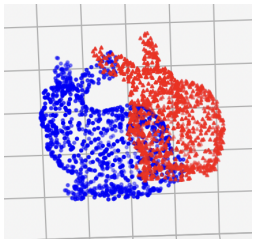
We applied the FPFH-ICP algorithm on the real-world terrain point cloud as well. The results are shown in Fig 8 step by step. The result after FPFH in Fig 8b, shows the global rough matching of the point clouds by FPFH, but misalignment still exists. After ICP applied, in Fig 8c and 8d, the target and the source are completely aligned with each other.

In Section IV-E, comparison of the performance in terms of the variables (div, k, χ) is conducted, and the evaluation of PFH, SPFH and FPFH is made quantitatively, evaluated by the runtime t and the total error ϵ.

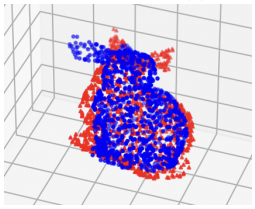
E. Algorithm Performance

As discussed above, we evaluated the performance of three different PFH algorithms by varying the variables that are div, k and χ (declared in Section IV-B). The performance is quantitatively evaluated by the *runtime* and the *error*.

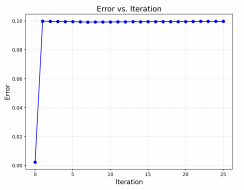
First of all, the percentile of featured points invoked by PFH, χ, is chosen to be the independent variable, varying from 0 to 100. In Fig 9, the performance is in terms of percentile χ and bin size div, and in Fig 10, the performance is in terms of percentile χ and number of neighbor k. The results turn out that the percentile χ affects the runtime t and error ϵ greatly on both cases, while bin size div and number of neighbor k does



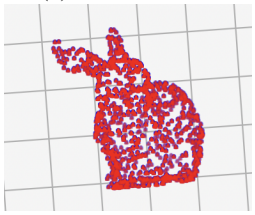
(a) Starting point.



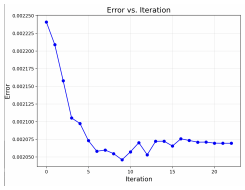
(b) Result of ICP.



(c) Error of ICP.

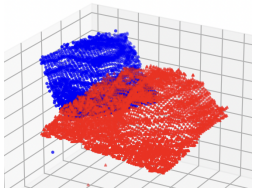


(d) Result of FPFH-ICP.

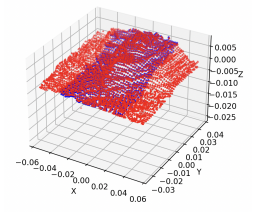


(e) Error of FPFH-ICP.

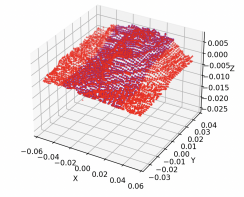
Fig. 7: Comparison between ICP and FPFH-ICP in the bunny example.



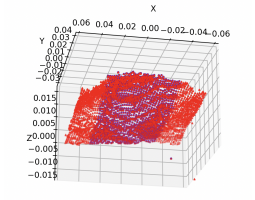
(a) Starting point.



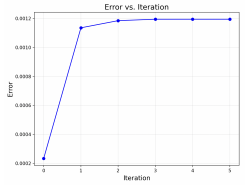
(b) Result of FPFH.



(c) Result of FPFH-ICP.



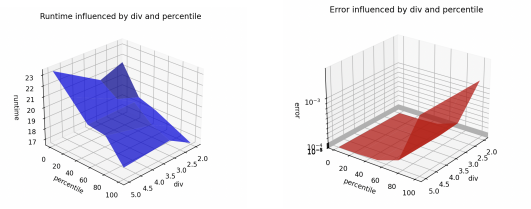
(d) Result of FPFH-ICP (another view).



(e) Error of FPFH-ICP.

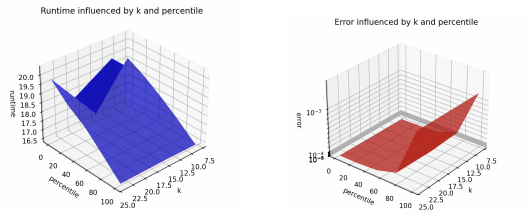
Fig. 8: Result of FPFH-ICP in the terrain example.

slightly. The higher the χ is, the lower the runtime becomes and the higher total error becomes, namely the slightly faster and much less accurate the performance becomes. In our application, the accuracy is in a more important position than runtime. Thus, we use $\chi = 0$ in the tests afterwards.



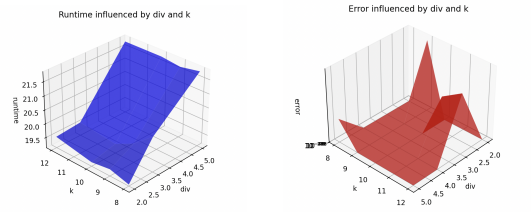
(a) Runtime influenced by χ and div. (b) Total error influenced by χ and div.

Fig. 9: Plot of performance influenced by χ and div.



(a) Runtime influenced by χ and div. (b) Total error influenced by χ and div.

Fig. 10: Plot of performance influenced by χ and k.

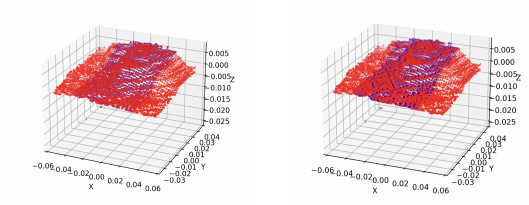


(a) Runtime influenced by div and k. (b) Total error influenced by div and k.

Fig. 11: Plot of performance influenced by div and k using FPFH.

We set the percentile $\chi = 0$ and radius $r = 0.5$ in the following evaluation of PFH and FPFH.

As for the FPFH approach, in Fig 11, we evaluate the influences of div and k on the performance runtime and total error. Depicted in Fig 11a, the runtime t is greatly sensitive to the bin size div, but slightly affected by number of neighbors k. As for the total error of the result ϵ, it generally decreases as the bin size div increases. It is worth mentioning that the total errors shown here are in logarithmic scale, from 10^{-35} to 10^{-7}, which means that it is satisfactory enough to real life cases. The visualization of the results with smallest and largest ϵ are shown below in Fig 12, where Fig 12a represents the smallest total error $\epsilon = 4.96 \times 10^{-35}$ when $div = 4$, $k = 12$,

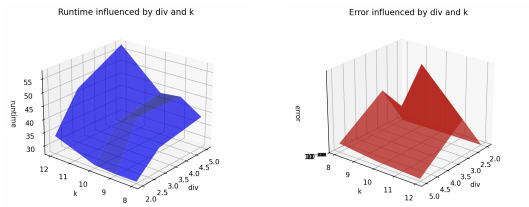


(a) Result with smallest ϵ (b) Result with largest ϵ

Fig. 12: Plot of results with smallest ϵ and largest ϵ using FPFH.

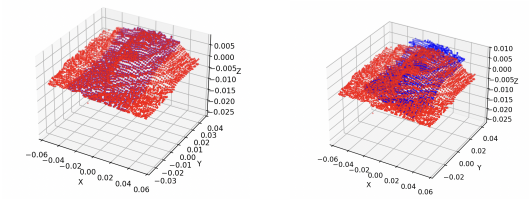
and Fig 12b represents the largest total error $\epsilon = 1.18 \times 10^{-6}$ when $div = 2$, $k = 9$.

As for the PFH approach, in Fig 13, similar features are concluded that the runtime t increases with increasing div and k, while the total error ϵ generally decreases with increasing div and k, but outliers still exist. The visualization of the results with smallest and largest ϵ are demonstrated in Fig 15, where Fig 14a represents the smallest total error $\epsilon = 2.87 \times 10^{-34}$ when $div = 4$, $k = 10$, and Fig 14b represents the largest total error $\epsilon = 1.88 \times 10^{-5}$ when $div = 3$, $k = 8$.



(a) Runtime influenced by div and k. (b) Total error influenced by div and k.

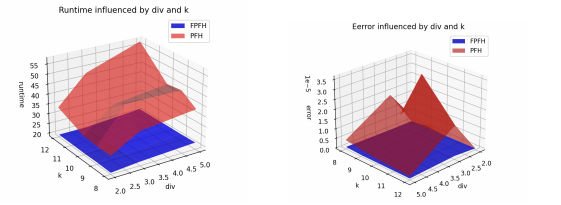
Fig. 13: Plot of performance influenced by div and k using PFH.



(a) Result with smallest ϵ (b) Result with largest ϵ

Fig. 14: Plot of results with smallest ϵ and largest ϵ using PFH.

The results of each of the methods turn out to be a trade off between the speed (runtime t) and the accuracy (total error ϵ). The more bin size div and number of neighbors k, the more accurate the result is, but meanwhile, the slower it takes to figure out. Under the same condition (same div and ϵ), the runtime of PFH method is much slower than that of FPFH, and the accuracy is also much lower than that of FPFH, in Fig 15.



(a) Comparison of runtime. (b) Comparison of error.

Fig. 15: Comparison of performance between FPFH and PFH.

V. Conclusion

We have implemented the Point Feature Histogram, Fast Point Feature Histogram, Simple Point Feature Histogram, Iterative Closest Point Algorithms. Our algorithm of processing point cloud is evaluated by synthetic and real-world data, fully overlapped and partially overlapped cases, all test cases showing success and great improvement on the performance. The comparison between different algorithms has been made, indicating the FPFH-ICP algorithm stands out from the other algorithms and variants. Also, the comparison on the processing performance with various parameters has been conducted, inferring that the trade off between processing speed and processing accuracy does exist throughout all algorithms and scenarios. Thus, it's critical to select proper parameters to tune the algorithm based on different application scenarios. With the help of the FPFH-ICP algorithm to process the point clouds, the importance of point cloud processing method will keep increasing in robotic fields of surroundings perception, SLAM, self driving etc.

Acknowledgment

The authors would like to express their sincere gratitude to Professor Dmitry Berenson and GSI Frank Lai for their invaluable guidance, constructive feedback, and continuous support throughout this work. Their expertise in motion planning and point cloud processing has been instrumental in shaping the ideas and methodologies presented in this paper.

The authors also wish to thank the Robotics Department at University of Michigan for providing access to the resources and tools necessary for implementing and evaluating the proposed methods. Lastly, we acknowledge the contributions of our colleagues and peers for their helpful discussions and insights on ICP and PFH algorithms, which greatly enriched the scope of this research.

References

[1] R. B. Rusu, Z. C. Marton, N. Blodow, and M. Beetz, "Persistent Point Feature Histograms for 3D Point Clouds," in Intelligent Autonomous Systems 10, pp. 119–128, 2008.

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

[3] "The Stanford 3D Scanning Repository," [Online]. Available: http://graphics.stanford.edu/data/3Dscanrep/. [Accessed: Dec. 12, 2024].

[4] Point Cloud Library (PCL), GitHub repository, [Online]. Available: https://github.com/PointCloudLibrary/data/tree/master/terrain. [Accessed: Dec. 12, 2024].