Cascading Feature Extraction for Fast Point Cloud Registration

Yoichiro Hisadome
hisadome@hal.t.u-tokyo.ac.jp
Yusuke Matsui
matsui@hal.t.u-tokyo.ac.jp

Graduate School of Information Science and Technology
The University of Tokyo
Tokyo, Japan

Abstract

We propose a method for speeding up a 3D point cloud registration through a cascading feature extraction. The current approach with the highest accuracy is realized by iteratively executing feature extraction and registration using deep features. However, iterative feature extraction takes time. Our proposed method significantly reduces the computational cost using cascading shallow layers. Our idea is to omit redundant computations that do not always contribute to the final accuracy. The proposed approach is approximately three times faster than the existing methods without a loss of accuracy.

1 Introduction

Point cloud registration is the task of aligning two 3D point clouds into a single point cloud. The two point clouds to be aligned usually represent the same object but are taken from different angles. Point cloud registration is a fundamental building block for 3D data processing, such as simultaneous localization and mapping and 3D object reconstruction.

The current state-of-the-art algorithm [14] in terms of accuracy takes approximately 60 ms to align approximately 1,000 points on a standard GPU (NVIDIA Tesla V100). To achieve a real-time performance, the inference should be less than 30 ms, and less than 20 ms would be even more desirable.

In this paper, we propose a fast 3D point cloud registration method. Our idea is to omit the computation of a part of the iterative process that is not necessarily accurate. Existing methods repeat the same heavy processing during every iteration. With the proposed method, the heavy processing runs only once initially, and the light processing is repeated in the subsequent iterations. Through this approach, we can significantly reduce the overall computational cost.

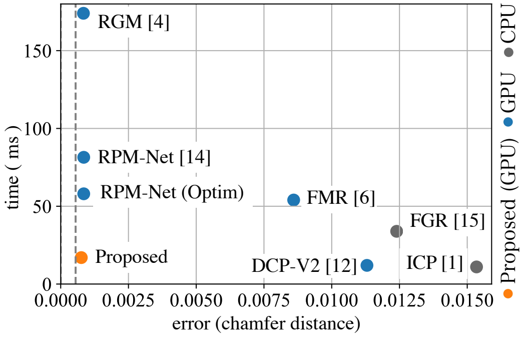

Figure 1: Diagram of accuracy and speed. The proposed method achieves both a high accuracy and speed (evaluated on modelnet40 [13]).

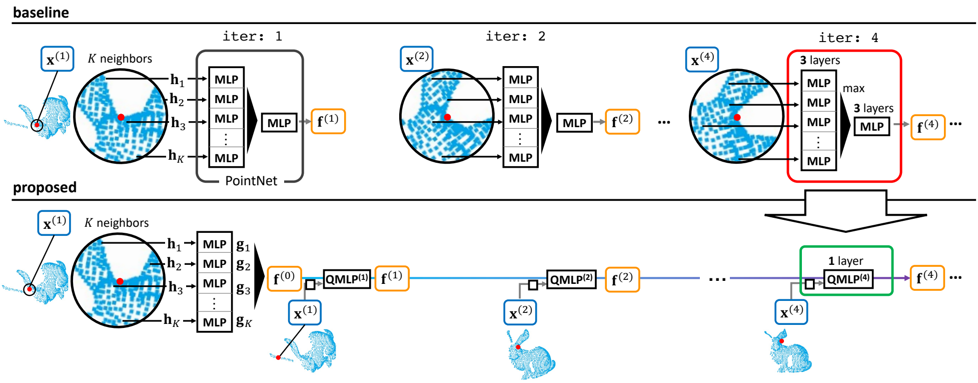

Figure 2: Overview of our method, with comparison to the baseline RPMNet [14]. The subscripts and superscripts denote the index and number of iterations, respectively.

RPMNet [14] and RGM [4] are the most accurate methods available today. RPMNet and RGM achieve the same level of accuracy, whereas RPMNet is more than twice as fast. Thus, we used RPMNet for our baseline in this study. The results and a comparison with other methods are shown in Fig. 1. The proposed method is three times faster than RPMNet without any loss of accuracy. In addition, our method is more accurate than RGM.

2 Related Work

Next, we show a general pipeline of modern point cloud registration methods [4, 12, 14, 15] and describe in detail the latest approach, RPMNet [14]. Let us assume that there are two point cloud sets, $\mathcal{X} \subset \mathbb{R}^3$ and $\mathcal{Y} \subset \mathbb{R}^3$. Many modern point cloud registration methods take the following steps.

(i) Obtain a feature vector for each point, for both $\mathcal{X}$ and $\mathcal{Y}$.

(ii) For all combinations of a feature vector in $\mathcal{X}$ and one in $\mathcal{Y}$, compare the distance in a brute force fashion and obtain the correspondence among $\mathcal{X}$ and $\mathcal{Y}$.

(iii) Estimate the rigid transformation T that minimizes the sum of the squared distances between the corresponding points.

Several methods, including RPMNet, take the following procedures in addition to (i)∼(iii) above.

(iv) Apply the rigid transformation to one of the objects $\mathcal{Y} \leftarrow T(\mathcal{Y})$ and return to (i).

Let us describe this in detail for RPMNet. The upper part of Fig. 2 shows the configuration of RPMNet. First, we describe the feature extraction in (i). Let us consider the extraction of a 96-dimensional feature $\mathbf{f} \in \mathbb{R}^{96}$ from a point $\mathbf{x} \in \mathbb{R}^3$ in a point cloud. We collect the K-nearest neighbors of $\mathbf{x}$ and compute a classical feature vector $\{\mathbf{h}_k\}_{k=1}^K$. Here, $\mathbf{h}_k \in \mathbb{R}^{10}$ is a 10-dimensional feature that consists of (1) the angles between the k-th neighbor of $\mathbf{x}$ and $\mathbf{x}$, (2) the absolute coordinate of the k-th neighbor of $\mathbf{x}$, and (3) the relative coordinate of the k-th neighbor of $\mathbf{x}$ with $\mathbf{x}$ as the coordinate reference. We obtain a single 96-dimensional feature $\mathbf{f} \in \mathbb{R}^{96}$ by inputting $\{\mathbf{h}_k\}_{k=1}^K$ into PointNet (PN):

$$\mathbf{f} = \mathrm{PN}(\{\mathbf{h}_k\}_{k=1}^K). \tag{1}$$

PointNet [2] is a feature extraction module that consists of a multilayer perceptron (MLP) and maxpool.

Next, we describe the computation of the correspondence in (ii). In the process of (ii) for RPMNet, a soft registration is conducted using simulated annealing. This makes their method quite robust to noise. Let $\mathcal{X} = \{\mathbf{x}_i\}_{i=1}^{N} \subset \mathbb{R}^3 and \mathcal{Y} = \{\mathbf{y}_i\}_{i=1}^{M} \subset \mathbb{R}^3$ be the two point clouds to be aligned. Let $d_{i,j}$ be the Euclidean distance between the features of $\mathbf{x}_i$ and $\mathbf{y}_j$. Herein, we construct a matrix $\mathbf{M} \in \mathbb{R}^{N \times M}$, where the (i, j)-th element $m_{i,j} \in \mathbf{M}$ is defined as follows:

$$m_{i,j} \quad = \quad e^{-\beta(d_{i,j}^2 - \alpha)}. \tag{2}$$

The parameters α and β are positive scalars. An independent network estimates the optimal values of α and β. If β is large, $\mathbf{M}$ represents a harder (discrete) correspondence; however, if it is small, $\mathbf{M}$ represents a softer (ambiguous) correspondence. In addition, β takes a larger value for each iteration. Moreover, α serves as a distance threshold, and distances of less than α are considered sufficiently close. If $d_{i,j}$ is small, $m_{i,j}$ takes a larger value and can be regarded as the similarity between $\mathbf{x}_i$ and $\mathbf{y}_j$. Next, Sinkhorn's algorithm [11] normalizes $\mathbf{M}$. From the normalized $\mathbf{M}$, we obtain the correspondence between the points.

The orthogonal Procrustes method [10] analytically calculates the optimal rigid transformation T (iii). To deal with a partial overlap, we adopted the same strategy as in [12, 14]. We used $\sum_i^N m_{i,j}$ to weigh the correspondences. A point with no correspondence has a weight of nearly 0. Then, repeating the procedure from the feature extraction to applying a rigid transformation improves the accuracy (iv). In RPMNet, a feature extraction is a bottleneck process. Our experiments using an NVIDIA Tesla V100 confirmed that RPMNet takes approximately 80 ms to align approximately 1,000 objects, and feature extraction occupies more than 50% of the computational time.

3 Proposed method

Our main idea is to reduce the computational cost by replacing PointNet with an extremely lightweight layer and a single linear layer with ReLU. We call this single linear layer with ReLU a quasi-MLP (QMLP) because it is equivalent to an MLP when it is cascaded. One may wonder if replacing PointNet with such a shallow layer would result in a loss of accuracy. However, we found that it is possible to achieve the same level of accuracy by cascading the QMLPs during each iteration. First, we describe the proposed cascade feature extraction method in Sec. 3.1, and in Sec. 3.2, explain why QMLP has the same expressive power as an MLP.

3.1 Cascading feature extraction

First, we describe the feature extraction in the first iteration. The bottom part of Fig. 2 shows the configuration of our network. Similar to the baseline, for feature extraction of a single point, the proposed method collects K-nearest neighbor points and calculates a classical feature vector $\mathbf{h}$ for each point. However, unlike RPMNet, the proposed method does not include the absolute coordinate information in $\mathbf{h}$, and thus $\mathbf{h} \in \mathbb{R}^7$. First, we input $\{\mathbf{h}_k\}_{k=1}^{K}$ individually into $\mathrm{MLP} : \mathbb{R}^7 \to \mathbb{R}^{96}$ to obtain a 96-dimensional feature vector.

$$\mathbf{g}_k = \mathrm{MLP}(\mathbf{h}_k). \tag{3}$$

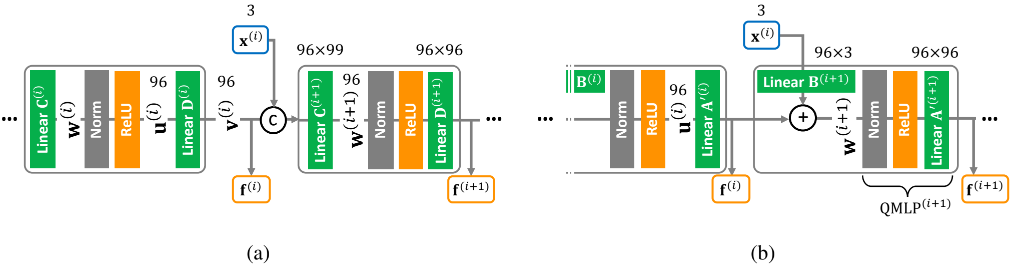

Figure 3: (a) Configuration of a cascaded feature extractor using an ordinary MLP (i.e., PointNet). The linear layers are successive between iterations. (b) Feature extractor constructed using the proposed QMLP. ⓒ indicates a concatenation, and ⊕ denotes an addition.

We then take the maximum value of the obtained feature vectors for each dimension and obtain an intermediate feature vector $\mathbf{f}^{(0)} \in \mathbb{R}^{96}$.

$$\mathbf{f}^{(0)} = \mathrm{maxpool}(\mathcal{G}) \in \mathbb{R}^{96}, \tag{4}$$

where $\mathcal{G} = \{\mathbf{g}_k\}_{k=1}^{K} \subset \mathbb{R}^{96}$. This is the process for a single point. By doing this for all points in the point cloud, we obtain a set of feature vectors $\mathcal{F}^{(0)} \subset \mathbb{R}^{96}$. Note that the superscript (i) denotes the number of iterations.

Next, we describe the feature extraction for the second and subsequent iterations. In the $i+1$ iteration of the proposed method, the set of feature vectors $\mathcal{F}^{(i)}$ and the absolute positions of points $\mathcal{X}^{(i)} \subset \mathbb{R}^3$ obtained in the i-th iteration are input into $\mathrm{QMLP}^{(i+1)} : \mathbb{R}^{96} \to \mathbb{R}^{96}$. Let $\mathbf{x}^{(i)} \in \mathcal{X}^{(i)}$ be the absolute position of a point, and $\mathbf{f}^{(i)} \in \mathcal{F}^{(i)}$ be the corresponding set of feature vectors at the i-th iteration. We obtain a feature vector $\mathbf{f}^{(i+1)}$ for the $i+1$-th iteration as follows:

$$\mathbf{f}^{(i+1)} = \mathrm{QMLP}^{(i+1)}\left(\mathbf{f}^{(i)} + \mathbf{B}^{(i+1)}\mathbf{x}^{(i)}\right). \tag{5}$$

Here, $\mathbf{B}^{(i+1)} \in \mathbb{R}^{96\times 3}$ is a thin matrix, which will be discussed in Sec. 3.2.

The critical differences between our design and that of RPMNet are two-fold. (1) We use $\mathbf{f}^{(i)}$ to calculate $\mathbf{f}^{(i+1)}$. This design is the origin of our paper titled “Cascading feature extraction” (2) As detailed in Secs. 3.1 and 3.2, the number of operations in our scheme is much smaller than that in RPMNet.

The proposed cascading feature extraction method utilizes the nature of simulated annealing. Features are not always required to be fully discriminative at the beginning of the iterations because we apply only a coarse alignment. Therefore, even if QMLP replaces several layered MLPs, the final accuracy often remains unchanged. By contrast, at the end of the iterations, because the feature vectors passed through several QMLPs in the previous iterations, a single QMLP can conduct an equivalent computation by passing through the MLPs of the deeper layers. Thus, QMLP, the computational cost of which is low, replaces PointNet.

3.2 QMLP

In terms of matrix operations, we show that cascading QMLPs are equivalent to the cascading ordinal MLPs used in PointNet and other applications. QMLP consists of one linear transformation and one ReLU layer. In the following, the matrix size and other values are the same as those in the actual implementation. The MLP used in PointNet contains at least two linear transformations. Each layer connects the input layer to the hidden layer and the hidden layer to the output layer.

We found that a cascaded feature extractor with an ordinary MLP is redundant. Let us first model the ordinary MLP as follows. Here, two linear transformations continue in succession between iterations, as shown in Fig. 3(a). This is a typical configuration of PointNet. In Figs. 3(a) and 3(b), each block represents an MLP in the i-th iteration. The input to the MLP passes through the first linear layer $\mathbf{C}^{(i)} \in \mathbb{R}^{96\times 99}$, resulting in a vector $\mathbf{w}^{(i)} \in \mathbb{R}^{96}$. Next, normalization and ReLU layers are applied to $\mathbf{w}^{(i)}$ to obtain a hidden vector $\mathbf{u}^{(i)} \in \mathbb{R}^{96}$. Subsequently, the second linear layer $\mathbf{D}^{(i)} \in \mathbb{R}^{96\times 96}$ transforms $\mathbf{u}^{(i)}$ into the final output $\mathbf{v}^{(i)} \in \mathbb{R}^{96}$. The input to the MLP of the next iteration is a stack of $\mathbf{x}^{(i)}$ and $\mathbf{v}^{(i)}$.

Here, we show that we can reduce the number of calculations by rewriting the operation. Recall from Fig. 3(a) that we can obtain $\mathbf{w}^{(i+1)}$ from $\mathbf{C}^{(i+1)}$, $\mathbf{v}^{(i)}$, and $\mathbf{x}^{(i)}$. The equation can then be rewritten as follows:

$$
\begin{aligned}
\mathbf{w}^{(i+1)} &= \mathbf{C}^{(i+1)} \begin{bmatrix} \mathbf{v}^{(i)} \\ \mathbf{x}^{(i)} \end{bmatrix} && (6)\\
&= \begin{bmatrix} \mathbf{A}^{(i+1)} \,|\, \mathbf{B}^{(i+1)} \end{bmatrix} \begin{bmatrix} \mathbf{v}^{(i)} \\ \mathbf{x}^{(i)} \end{bmatrix} && (7)\\
&= \mathbf{A}^{(i+1)}\mathbf{v}^{(i)} + \mathbf{B}^{(i+1)}\mathbf{x}^{(i)} && (8)\\
&= \mathbf{A}^{(i+1)}\mathbf{D}^{(i)}\mathbf{u}^{(i)} + \mathbf{B}^{(i+1)}\mathbf{x}^{(i)} && (9)\\
&= \mathbf{A}'^{(i)}\mathbf{u}^{(i)} + \mathbf{B}^{(i+1)}\mathbf{x}^{(i)}. && (10)
\end{aligned}
$$

In this study, we decompose $\mathbf{C}^{(i+1)} \in \mathbb{R}^{96\times 99}$ into smaller matrices $\mathbf{A}^{(i+1)} \in \mathbb{R}^{96\times 96}$ and $\mathbf{B}^{(i+1)} \in \mathbb{R}^{96\times 3}$. We then define a new matrix $\mathbf{A}'^{(i)} = \mathbf{A}^{(i+1)}\mathbf{D}^{(i)} \in \mathbb{R}^{96\times 96}$. The final representation of Eq. 10 is our proposed module, which we visualize in Fig. 3(b). Equation 10 implies that if we smartly configure the matrices, we can compute $\mathbf{w}^{(i+1)}$ simply by (1) the output of the previous step ($\mathbf{A}'^{(i)}\mathbf{u}^{(i)}$) and (2) the current point position $\mathbf{x}^{(i)}$ with a shallow transformation $\mathbf{B}^{(i+1)}$.

In a conventional MLP, two matrix operations using matrices **C** and **D** are conducted on the input vector (the green modules in Fig 3(a)). By contrast, with the proposed method, the costly operation uses only $\mathbf{A}^{'}$ (the green modules in Fig 3(b)). This is because the size of **B** is much smaller than that of $\mathbf{A}^{'}$.

These results show that the proposed cascaded feature extractor can achieve the same performance as an ordinary MLP while reducing the computational cost through the use of QMLP.

3.3 Analysis

Accurate Correspondence Next, we show why the proposed method does not degrade the accuracy, it occasionally improves it. The patterns that we should focus on differ depending on the registration stage. First, at the beginning of the iterations, QMLP needs to focus on the local feature of $\mathbf{f}^{(i)}$ rather than the positional feature derived from $\mathbf{x}^{(i)}$. This is because the information of the absolute position does not help find the correspondence during this stage. By contrast, at the end of the iterations, focusing on both the local and positional features leads to a correct correspondence. Fig. 4 illustrates this. There are many points whose local

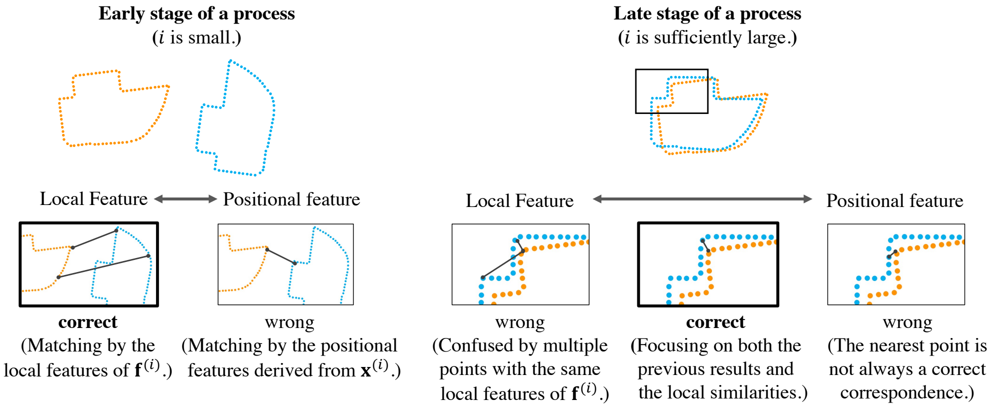
Early stage of a process
(i is small.)
Local Feature
Positional feature
correct
(Matching by the local features of $\mathbf{f}^{(i)}$.)
wrong
(Matching by the positional features derived from $\mathbf{x}^{(i)}$.)
Late stage of a process
(i is sufficiently large.)
Local Feature
Positional feature
wrong
(Confused by multiple points with the same local features of $\mathbf{f}^{(i)}$.)
correct
(Focusing on both the previous results and the local similarities.)
wrong
(The nearest point is not always a correct correspondence.)

Figure 4: Change in pattern to be identified between the beginning and end of the iteration. At the end of the iteration, only the close points in the absolute position are in correspondence.

features are similar. However, because we roughly aligned the point clouds at the beginning of the iterations, the only correct counterparts are already close in position.

The baseline used PointNet, which has the same weight parameters for all iterations. This means that it focuses on the same pattern during all stages of alignment. By contrast, with our proposed method, the feature extractor parameters used in each iteration are all different. Therefore, our proposed method can adaptively focus on different patterns at different stages of alignment.

Computational Cost The baseline uses K-nearest neighbors for the feature extraction of a single point. It applies the same feature extraction every iteration, and thus has a computational complexity of $\mathcal{O}(D^2KL)$. The dimensionality of the feature vectors is D. The number of neighbors used for feature extraction is K. The number of iterations is L. Herein, we approximate the running time of PointNet, where the maximum width of D is $\mathcal{O}(D^2)$. By contrast, the proposed method has a computational complexity of $\mathcal{O}(KD^2)$ in the first iteration, as in PointNet, and $\mathcal{O}(D^2)$ in the second and subsequent iterations, and thus the total computational complexity is $\mathcal{O}(D^2(K+L))$. This is indeed smaller than $\mathcal{O}(D^2KL)$.

3.4 Other Ingenuities for Speedup

We propose two other ingenuities to accelerate the method: batch annealing parameter extraction and Sinkhorn normalization.

Batch Prediction of Annealing Parameters A network separate from the feature extraction module predicts the appropriate parameters (α and β) at each iteration in the baseline. Thus, the cost of annealing parameter prediction increases according to the number of iterations. However, with the proposed method, the parameters for each iteration are estimated simultaneously in the first iteration. Therefore, we were also able to reduce the computational complexity of the parameter estimation for use in the annealing. This reduced the computational cost of the annealing parameter prediction by a factor of five.

Adaptive Iteration for Sinkhorn Normalization The number of iterations of Sinkhorn normalization is constant in the baseline. By contrast, we changed the number of iterations depending on the alignment stage to speed up the process. Initially, we reduce the number of iterations, and then increase the number to a sufficient value. In our implementation, we empirically set the number to equal the number of current registration iterations (i.e., if $i = 4$, the number of iterations of Sinkhorn normalization is also set to four). This operation accelerates Sinkhorn normalization 1.5-fold. This is to take advantage of the fact that the early stages of alignment do not require precision.

3.5 Code-Level Optimization

We found that the original implementation of RPM-Net has room to accelerate at the implementation level. Thus, we improved RPM-Net [14] for three points: Sinkhorn normalization, singular value decomposition (SVD), and a nearest neighbor search. We call the result "RPM-Net (Optim)", which is 20 ms faster than the original implementation (see Fig. 1). We use RPM-Net (Optim) for our comparison, which is the basis of our proposed method.

Sinkhorn Normalization We accelerated the runtime of Sinkhorn normalization by three-fold through the following modification. As the baseline, the authors integrated Sinkhorn normalization and the calculation of the similarities using the exponential, following the method of Mena *et al.* [8]. The implementation is shown in Algorithm 1. However, the computation costs of the logarithmic and exponential functions are much higher than those of addition and division. Therefore, we reimplemented the Sinkhorn normalization in Algorithm 2, where l is the number of iterations of the Sinkhorn normalization.

SVD We accelerated the SVD by computing it on a CPU instead of a GPU. In the existing code, all registration algorithms, including SVD, were executed on a GPU. We profiled the execution time of the SVD on a GPU and found that it took approximately 2 ms to process a 3×3 matrix. We studied this process in detail and found that SVD is faster on a CPU even after including the transfer cost to the CPU.

The reason for this may be the difference in performance between the GPU and CPU on a single core. If a problem is small and difficult to parallelize, GPUs are slower than CPUs. Here, the size of the covariance matrix of the point cloud handled by the orthogonal Procrustes method is 3×3, which cannot be parallelized. Because the matrix to be transferred is extremely small, the communication time between the CPU and GPU is negligible. With this revision, the processing time of the Orthogonal Procrustes algorithm was halved.

Nearest Neighbor Search We used Faiss [7] to search for the K-nearest neighbors for the feature extraction of each point. Faiss is a fast library, particularly for large-scale data, and we therefore applied it during the experiment using KITTI [5]. When the number of points is 12,000, the K-nearest neighbor search becomes six times faster than the top-k function of PyTorch [9].

Algorithm 1 Implementation by Mena *et al.* [8]

```
while $c < l$ do
    $d_{i,j} \leftarrow d_{i,j} - \log(\sum_i e^{d_{i,j}})$ , $\forall i, j$
    $d_{i,j} \leftarrow d_{i,j} - \log(\sum_j e^{d_{i,j}})$ , $\forall i, j$
    $c \leftarrow c + 1$
end while
$m_{i,j} \leftarrow \exp(d_{i,j}{}^2)$, $\forall i, j$
```

Algorithm 2 Standard implementation of Sinkhorn normalization

```
$m_{i,j} \leftarrow \exp(d_{i,j}{}^2)$, $\forall i, j$
while $c < l$ do
    $m_{i,j} \leftarrow m_{i,j} / \sum_i m_{i,j}$, $\forall i, j$
    $m_{i,j} \leftarrow m_{i,j} / \sum_j m_{i,j}$, $\forall i, j$
    $c \leftarrow c + 1$
end while
```

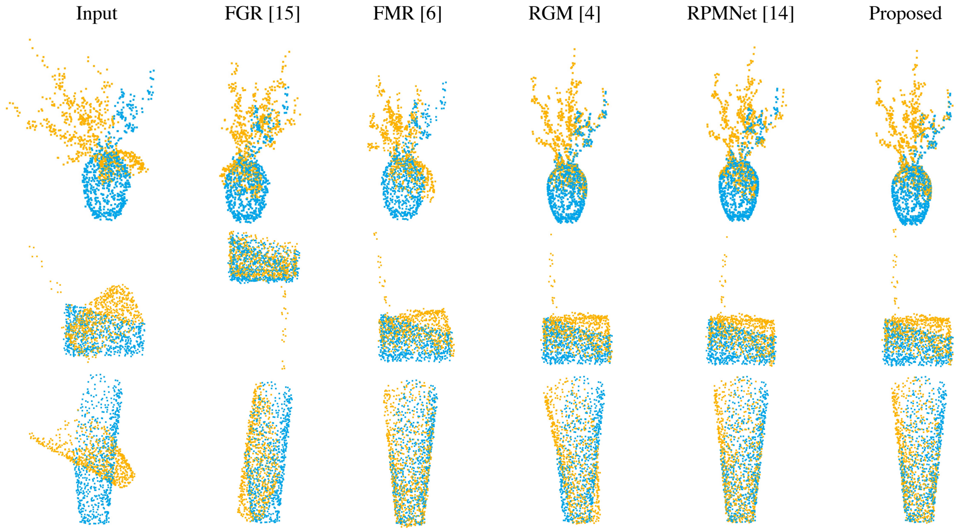

Figure 5: Qualitative comparison

4 Experiment

4.1 Environment

We used the GPU cluster of the National Institute of Advanced Industrial Science and Technology, AI Bridging Cloud Infrastructure (ABCI), to measure the speed of the model. The GPU is an NVIDIA Tesla V100, the CPU is an Intel Xeon Gold 6148, and the main memory is 32-GB DDR4 2666-MHz RDIMM (ECC).

4.2 Datasets

The datasets used in our experiments are modelnet40 [13] and KITTI [5]. Here, modelnet40 is a 40-class 3D point cloud object. We preprocessed modelnet40 in a similar manner as the methods developed by Yew and Lee [14] and Wang and Solomon [12]. We used approximately 5,100 point cloud objects from the first 20 classes for training, approximately 1,200 for validation, and approximately 1,300 from the second 20 classes for the test. The number of points in each object was 2,048, and 1,024 points were randomly selected for the test.

For the input data, approximately 70% of the point clouds were selected from a single point cloud object, and the point clouds to be aligned have an overlap of approximately 40%. Furthermore, Gaussian noise was applied to each point position of the objects, and one of the objects was rotated and translated. The rotations are within the range [0, 45°] for three Euler angles. Translations are within the range [-0.5, 0.5] in modelnet40, and [-15, 15] in KITTI.

KITTI is made up of point cloud data from an urban area. KITTI point cloud data come with GPS location information. Based on the location information, we can create the teacher data for positioning. However, owing to the large error in GPS positioning, we follow the method of Choy *et al.* [3] and refine the data using the ICP algorithm. During each stage, one of the point clouds was rotated randomly to the initial position. The model trained by modelnet40 was retrained using KITTI and evaluated.

4.3 Models

In this section, we describe the test parameters of the proposed and the comparison methods. There are 64 K-nearest neighbor points for feature extraction and five registration iterations.

To make a fair comparison, we first conducted a code-level acceleration on the existing RPMNet, which we call RPM-Net (Optim). Although this achieves the same accuracy as RPMNet, it is 40% to 200% faster. This speedup includes the optimization described in Sec. 3.5.

We compared our approach with ICP [1], FGR [15], DCP [12], FMR [6], and RGM [4]. Among these methods, ICP and FGR are classical methods that are not learning-based and are computed on a CPU. The other methods are based on deep learning and are executed on a GPU. In addition, FMR does not use a feature matching scheme, and RGM is the latest method to apply graph matching.

4.4 Results and Evaluation

For the above dataset, we conducted a point cloud registration and measured its accuracy and speed. Table 1 and Fig. 1 show the results. We used the modified chamfer error introduced in [14] ("CD" in Table 1). The accuracies of RPMNet, RGM, and our proposed method are much higher than those of the other methods. We can confirm that our proposed method is faster than the other models with high accuracy.

method	RE ↓	TE ↓	CD ↓	time ↓
ICP [1] (CPU)	27.25 °	28 cm	0.015	11 ms
FGR [15] (CPU)	31.43 °	20 cm	0.012	34 ms
DCP-V2 [12]	12.61 °	17 cm	0.011	12 ms
FMR [6]	12.14 °	17 cm	0.0086	54 ms
RPMNet (Optim) [14]	1.71 °	1.8 cm	0.00085	58 ms
RGM [4]	1.56 °	1.5 cm	0.00084	174 ms
Proposed	1.23 °	1.3 cm	0.00076	17 ms

Table 1: Speed and accuracy table for modelnet40. The proposed method is more accurate than the baseline.

Table 1 also shows that the proposed method outperforms the baseline and RGM in all evaluation metrics, i.e., rotation error, translation error, and chamfer distance. This can be attributed to the fact that the proposed method extracts features with different weight parameters for each iteration and can adaptively focus on different patterns depending on the alignment stage.

			4,000 pts	12,000 pts
method	RE ↓	TE ↓	time ↓	time ↓
RPMNet (Optim)	-	-	200 ms	750 ms
FGR	5.52 °	3.53 m	250 ms	1600 ms
FMR	1.89 °	1.18 m	67 ms	96 ms
Proposed	1.66 °	0.92 m	56 ms	220 ms

Table 2: Evaluation on KITTI dataset

Fig. 5 shows a qualitative comparison. Our proposed method and RPMNet achieved similar results, whereas our method is more accurate in terms of the details (e.g., branch of a tree in the first row) and three times faster. Both qualitative and quantitative comparisons show that our method is more accurate than RGM.

Table 3 shows an ablation study of the accuracy, using (1) separate weight parameters for each iteration and (2) the same weight parameters for all iterations. The result of "Proposed w/o Sec 3.1" is less accurate than that of "Proposed w/ Sec 3.1." This result justifies the explanation of Sec 3.1.

method	CD ↓
Proposed w/ Sec 3.1	0.00076
Proposed w/o Sec 3.1	0.0015

Table 3: Ablation study of cascading feature extraction.

Fig. 6 shows a comparison of the speed for each module. Code-optimization mainly improves the speed of the SVD and Sinkhorn normalization. Our proposed cascading feature extraction significantly speeds up the feature extraction, which has been a bottleneck.

Table 2 shows the speed and accuracy of the experiments using KITTI. Fig. 7 shows the results of the actual registration. We can confirm that the registration is accurate. The size of the point cloud used for the accuracy evaluation is 4,000. The proposed method is three times faster than the baseline, even for large point clouds such as KITTI.

Because the memory consumption of the baseline, RPMNet, is extremely large and difficult to train, we did not evaluate its accuracy on a large point cloud but only measured its inference time. Notably, FMR is the fastest for 12,000 points. FMR aligns two point clouds such that the feature vectors of the points are close. Specifically, it operates as follows: (1) The point cloud is perturbated; (2) A transformation is found and applied to the point cloud to minimize the error between the feature vectors of the point clouds; (3) The point cloud is perturbated again and the process is repeated until convergence is achieved. Thus, FMR does not explicitly depend on the number of points. This speeds up the FMR for a massive number of points. However, the overheads of (1) and (2) are significant even if the number of points is small. This overhead makes FMR relatively slower for small- and middle-scale data.

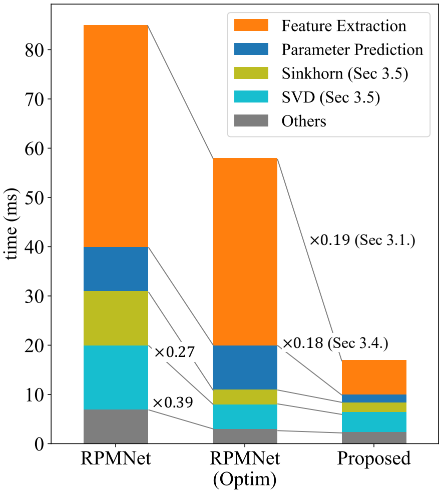

Figure 6: Module-wise speed comparison.

4.5 Limitation

Our method has two limitations. As the first condition, because we prepared individual weight parameters for each iteration, we need to specify the number of alignment iterations during the training phase. The baseline RPMNet does not have this limitation.

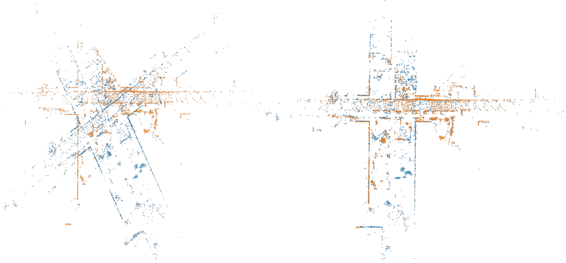

Figure 7: Example registration result of KITTI

The second edition is scalability. Similar to other existing methods such as RPMNet [14] and RGM [4], our method also uses Sinkhorn normalization, which has a computational complexity of the square of the number of points. Thus, it becomes a bottleneck when we increase the number of points, as with other methods.

5 Conclusion

We proposed a method for speeding up an iterative 3D point cloud registration. As our main idea, we do not have to compute the feature vectors until they are fully discriminative. We showed that our cascading feature extraction is computationally efficient. The inference speed is three times faster than that of the baseline. Furthermore, the accuracy is higher than that of the latest models, such as RPMNet and RGM.

6 Acknowledgements

This work was supported by JST, PRESTO, Japan, Grant Number JPMJPR 1936.

References

[1] K. S. Arun, T. S. Huang, and S. D. Blostein. Least-squares fitting of two 3-d point sets. volume PAMI-9, pages 698–700, 1987. doi: 10.1109/TPAMI.1987.4767965.

[2] R. Q. Charles, H. Su, M. Kaichun, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 77–85, 2017.

[3] Christopher Choy, Jaesik Park, and Vladlen Koltun. Fully convolutional geometric features. In *Proceedings of the IEEE International Conference on Computer Vision (CVPR)*, pages 8958–8966, 2019.

[4] Kexue Fu, Shaolei Liu, Xiaoyuan Luo, and Manning Wang. Robust point cloud registration framework based on deep graph matching. *Internaltional Conference on Computer Vision and Pattern Recogintion (CVPR)*, 2021.

[5] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[6] Xiaoshui Huang, Guofeng Mei, and Jian Zhang. Feature-metric registration: A fast semi-supervised approach for robust point cloud registration without correspondences. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[7] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, pages 1–1, 2019. doi: 10.1109/TBDATA.2019.2921572.

[8] Gonzalo Mena, David Belanger, Scott Linderman, and Jasper Snoek. Learning latent permutations with gumbel-sinkhorn networks. In *International Conference on Learning Representations (ICLR)*, 2018.

[9] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[10] Peter Schönemann. A generalized solution of the orthogonal procrustes problem. *Psychometrika*, 31(1):1–10, 1966.

[11] Richard Sinkhorn. A relationship between arbitrary positive matrices and doubly stochastic matrices. *Ann. Math. Statist.*, 35(2):876–879, 1964. doi: 10.1214/aoms/1177703591.

[12] Yue Wang and Justin M. Solomon. Deep closest point: Learning representations for point cloud registration. In *The IEEE International Conference on Computer Vision (ICCV)*, 2019.

[13] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1912–1920. IEEE Computer Society, 2015. ISBN 978-1-4673-6964-0.

[14] Zi Jian Yew and Gim Hee Lee. Rpm-net: Robust point matching using learned features. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[15] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Fast global registration. In *2016 European Conference on Computer Vision (ECCV)*, pages 766–782, 2016.