RPSRNet: End-to-End Trainable Rigid Point Set Registration Network using Barnes-Hut 2^D-Tree Representation

Sk Aziz Ali[1,2] Kerem Kahraman[1] Gerd Reis[2] Didier Stricker[1,2]
[1]TU Kaiserslautern [2]German Research Center for Artificial Intelligence (DFKI GmbH), Kaiserslautern

Abstract

We propose RPSRNet - a novel end-to-end trainable deep neural network for rigid point set registration. For this task, we use a novel 2^D-tree representation for the input point sets and a hierarchical deep feature embedding in the neural network. An iterative transformation refinement module of our network boosts the feature matching accuracy in the intermediate stages. We achieve an inference speed of ∼12-15 ms to register a pair of input point clouds as large as ∼250K. Extensive evaluations on (i) KITTI LiDAR-odometry and (ii) ModelNet-40 datasets show that our method outperforms prior state-of-the-art methods – e.g.*, on the KITTI dataset, DCP-v2 by 1.3 and 1.5 times, and PointNetLK by 1.8 and 1.9 times better rotational and translational accuracy respectively. Evaluation on ModelNet40 shows that RPSRNet is more robust than other benchmark methods when the samples contain a significant amount of noise and disturbance. RPSRNet accurately registers point clouds with non-uniform sampling densities,* e.g.*, LiDAR data, which cannot be processed by many existing deep-learning-based registration methods.*

1. Introduction

Rigid point set registration (RPSR) is indispensable in numerous computer vision and graphics applications – *e.g.*, camera pose estimation [2, 63], LiDAR-based odometry [74, 38], 3D reconstruction of partial scenes [30], simultaneous localization and mapping tasks [46], to name a few. An RPSR method estimates the rigid motion field, parameterized by rotation ($\mathbf{R} \in \mathcal{SO}(D)$) and translation ($\mathbf{t} \in \mathbb{R}^D$), of a moving sensor from the given pair of D-dimensional point clouds (*source* and *target*).

Generally, different types of input data from diverse application areas pose distinct challenges to registration methods, *e.g.*, (i) LiDAR data contains large number of points with non-uniform sampling density, (ii) partial scans obtained from structured-light sensors or multi-view camera systems contain a large number of points with small amount of overlap between the point clouds, (iii) RGB-D sensors,

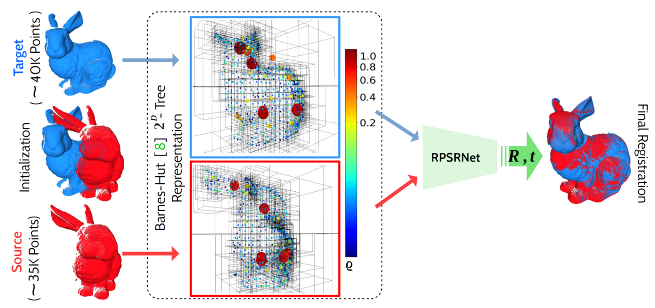


Figure 1. **Rigid Point Set Registration using Barnes-Hut (BH) $\mathbf{2^D}$-tree Representation.** The center-of-masses (CoMs) and point-densities (ϱ) of non-empty tree-nodes are computed for the respective BH-trees of the *source* and *target*. These two attributes are input to our RPSRNet which has global feature-embedding blocks employing tree-convolution operations on the nodes. Finally, we regress rigid rotation ($\mathbf{R} \in \mathcal{SO}(D)$) and translation ($\mathbf{t} \in \mathbb{R}^D$) parameters from output features.

such as Kinect, yield dense depth data with large displacement between consecutive frames.

Classical RPSR Methods. Iterative Closest Point (ICP) [10] and its many variants [29, 56, 36, 60, 70, 28, 23] are the most widely used methods that alternate the nearest correspondence search and transformation estimation steps at every iteration. A comparative study [50] shows that several ICP variants target a specific challenge, and many of them are prone to converge in bad local-minima. Coherent Point Drift (CPD) [44] is another state-of-the-art approach from the class of probabilistic RPSR methods. Similar to CPD, GMMReg [11], FilterReg [21], and HGMR [19] are also probabilistic RPSR methods that treat the data as Gaussian mixture models. The probabilistic models for rigid alignment gives more robustness against noisy input than ICP [10]. Recently physics-based approaches, *e.g.*, GA [25], BH-RGA [26], FGA [4], and [1, 3], are appearing computationally faster and more robust than ICP or CPD. These methods assume that the point clouds are astrophysical particles with masses and obtain the optimal alignment by defining a motion model for the source in a simulated gravitational field. However, most of the above methods run on CPUs and do not scale to register large point clouds, *e.g.*, LiDAR scans. Due to this, often handcrafted [18] or automatically extracted [58, 57] feature descriptors are iteratively sampled using RANSAC [53] to obtain true correspondence matches. On the other hand, Fast Global Registration (FGR) [77] replaces the RANSAC step with a robust global optimization technique to obtain true matches.

DNN-based Point Processing Registration Methods. A recent survey [39] shows that many contributions are made to the development of deep neural networks (DNNs) for point cloud processing (PCP) tasks [51, 52, 69, 5, 66, 67, 42, 30, 32, 13, 62, 71, 48, 47, 40, 61, 41], *e.g.*, classification and segmentation [51, 52, 69], correspondence and geometric feature matching [24, 35], up-sampling [72], and down-sampling [47]. Rigid point-set registration using DNNs appeared recently [20, 5, 66, 67, 42, 30, 48, 71, 13]. Elbaz *et al*. [20] introduce the LORAX algorithm for large scale point set registration using super-point representation. PointNetLK [5] is the first RPSR method which uses PointNet [51] (w/o its T-net component) to obtain the global feature-embedding of *source* and *target*. The method further uses the iterative Lucas and Kanade (LK) method to obtain the transformation optimizing the distance between global features. Unlike PointNetLK, the recent RPSR method Deep Closet Point (DCP) [66] chooses DGCNN [68] to obtain the feature embedding and a Transformer network [17] to learn contextual residuals between them. PRNet [67] is an extension of DCP [66] to solve a matching sharpness issue. Deep Global Registration (DGR) [13], [16], and 3DRegNet [48] are among the latest methods to report better accuracy than FGR [77] for registering partial-to-partial scans. DGR takes fully convolutional geometric features (FCGF) [15] and trains a high dimensional convolutional network [14] to classify inliers/outliers from the input FCGF descriptors. Finally, a weighted Procrustes [27] using inlier weights estimate the transformation. 3DRegNet [48] method also employs a classifier, similar to the inlier/outlier classification block of DGR, but using deep ResNet [31] layers, followed by differentiable Procrustes [27] to align the scans. Notably, both 3DRegNet and DGR have $\sim$4 times and $\sim$10 higher runtime than DCP [66]. Some other RPSR methods using DNNs are AlignNet-3D [30], DeepGMR [73] and DeepVCP [42].

Problems in DNN-based Point Cloud Processing. Generally, DNNs on 3D point clouds give more intuitive high dimensional geometric features learned from the given samples. Although, the convolution operations are not straight forward for DNNs on point clouds because they can be *unordered, irregular* and *unstructured*. Voxel-based [43] or shallow grid-based [54] representations use volumetric convolution which is very memory demanding ($\mathcal{O}(N^3)$ where N is the voxel resolution) and can only be applied to very small problems. In contrast, multilayer perceptron (MLP) based convolution in [51, 52, 69, 72] operate on sub-sampled versions of the point clouds. Thus, the local correlation between the latent-features are inefficiently established using random neighborhood search in MLP-based methods. Another critical disadvantage of methods relying on PointNet [51] (or its extension [52]) is that deconvolution is inapplicable. RPMNet [71], which is another PointNet [51] reliant RPSR method, shows that it requires points' normals to be computed beforehand for robust alignment. Additionally, the inference time, a crucial parameter for real-time applications, of several recent DNN-based methods [71, 48, 13] are not on a par with DCP [66] or DeepVCP [42]. Among the DNN based approaches of RPSR, no method is available so far which addresses the aforementioned issues.

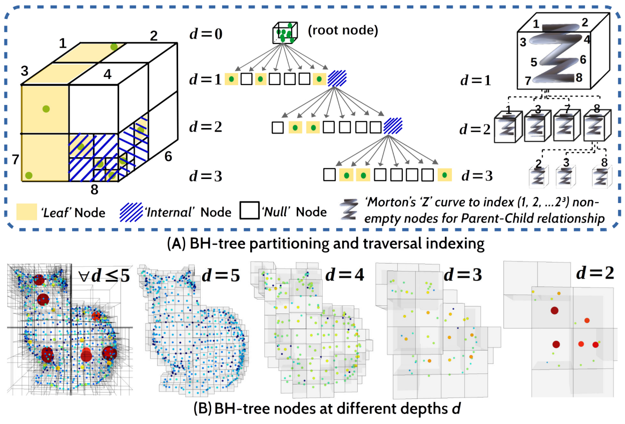
2^{D}-Tree Based Methods for Point Cloud Processing. 2^{D}-tree (*e.g.*, octree in 3D or quadtree in 2D) representation [76] of a point cloud is more memory efficient to encode local correlation among the points inside a tree-cell as they can share the same input signals. Unlike regular grids, 2^{D}-tree representation free up the empty cells, which are often more than 50% for sparse point clouds, from being computed. Moreover, it provides hierarchical correlations between the neighborhood cells. FGA [4] and BH-RGA [26] are the latest physics-based RPSR methods to use Barnes-Hut (BH) 2^{D}-tree representation of point clouds. FGA shows state-of-the-art alignment accuracy and the fastest speed among the classical methods. DNN-based methods for shape reconstruction from a single image [62], point cloud classification and segmentation (PCCS) [37, 65], real-time 3D scene analysis [64], shape retrieval [55], and large LiDAR data compression [32] have all appeared in last three years which claim 2^{D}-tree as more efficient learning representation for point clouds. Although, there is no RPSR method that addresses the aforementioned problems of previous learning-based approaches and simultaneously utilizes the efficacy of 2^{D}-tree representation.

In this paper, we present the first DNN-based RPSR method using a novel Barnes-Hut (BH) [8] 2^{D}-tree representation of input point clouds (see Fig. 1). At first, we build a BH-tree by recursively subdividing the normalized bounding space of an input point cloud up to a limiting depth d (Sec. 2). A tree with maximum depth $d = 6$ or 7 (*i.e.*, equivalent to 64^3 or 128^3 voxels in 3D) gives a fine level of granularity for our proposed DNN to process. Except from the root node, which contains all the points, the internal nodes of the 2^{D}-tree encapsulate varying number of points. Hence, the center-of-masses (CoMs) and the inverse densities (IDs) of the nodes, computed at each depth, are the representative attributes of a BH-tree. Besides, the neighbors of a given node are easily retrievable using established indexing and hash maps [7] for 2^{D}-tree. With this input representation, Sec. 3 describes the complete pipeline of RPSRNet. To this end, we design a single DNN block – *hierarchical feature extraction* (HFE) – with d hidden layers for global feature extraction. Two sub-blocks under HFE – namely *hierarchical position feature embedding* (HPFE) and *hierarchical density feature embedding* (HDFE) respectively – learn the positional and density features, respectively. We apply late-fusion between HPFE and HDFE that results in a density adaptive embedding. As a result, learned-features become homogeneous for the input point clouds with non-uniform point sampling densities (*e.g*., LiDAR scans). The final block of our RPSRNet, inspired by DCP-v2 [66], contains a relational network [59][i] with an integrated transformer [6], and a differentiable singular value decomposition (SVD)[ii] module. We further refine the estimation with an iterative rigid alignment architecture with multiple alignment-passes for a single pair of input sample (See Fig. 3 and Sec. 3.3).

Contributions. The overall contributions and promising characteristics of this work are as follows:

- A novel BH 2^D-tree representation of input point sets.
- An end-to-end trainable rigid registration network with real time inference on dense point clouds using the following components:
  1. HPFE and HDFE using 2^D-tree convolution
  2. Rigid transformation estimation using relational network and differentiable SVD.
  3. A multi-pass architecture for iterative transformation $(\mathbf{R}, \mathbf{t})$ refinement (similar to [67])

2. The Proposed BH 2^D-Tree Construction



d=0
(root node)
d=1
d=2
d=3
1
2
3
4
5
6
7
8
'Leaf' Node
'Internal' Node
'Null' Node
'Morton's 'Z' curve to index (1, 2, ...2³) non-empty nodes for Parent-Child relationship
(A) BH-tree partitioning and traversal indexing
∀d≤5
d=5
d=4
d=3
d=2
(B) BH-tree nodes at different depths d

Figure 2. Barnes-Hut $\mathbf{2^D}$-Tree construction and nodes.

Given a pair of misaligned input point clouds, a source $\mathbf{Y} \in \mathbb{R}^{M\times(\text{D+c})}$ and a target $\mathbf{X} \in \mathbb{R}^{N\times(\text{D+c})}$, we build Barnes-Hut (BH) [8] $\mathbf{2^D}$-trees $\tau^{\mathbf{Y}}$ and $\tau^{\mathbf{X}}$ on them. The trees ($\tau^{\mathbf{Y}}$ and $\tau^{\mathbf{X}}$) are defined by their respective sets of nodes ($\mathbf{N}^{\mathbf{Y}}_d$ and $\mathbf{N}^{\mathbf{X}}_d$) and the parent-child relationships among them at different depths $d = 1, 2, \ldots$. Every point in $\mathbf{Y}$ and $\mathbf{X}$ is a D-dimensional position vector, often associated with an additional c-dimensional input feature channel. The input features can be RGB color channels, local point densities, or other attributes. Although BH-tree is generalizable for D-dimensional input data, we describe further details and notations w.r.t 3D data (*i.e.* D = 3) for easier understanding. The following steps describe how to build a BH octree (*i.e.* $2^3 = 8$ child octants per parent node) by recursively subdividing the Euclidean space of an input:

1. We determine the extreme point positions, *i.e*., $\min$, $\max$ values along x, y, and z-axes, and split the whole space (as root node) from its center into 2^3 cubical sub-cells as its child nodes. The *half-length* of the cube is called cell length r. Depending on the structure of the input point clouds, if not a regular grid structure, the child nodes can be *empty*, *non-empty with more than one point*, and *non-empty with exactly one point*. We call them *null*, *internal*, and *leaf* nodes (see Fig. 2-A).
2. For each *internal* node, we compute the center-of-mass (CoM) $\mu \in \mathbb{R}^3$ and inverse density (ID)[iii] $\varrho^- \in \mathbb{R}^+$ of the encapsulated points inside. These two attributes (CoM and ID) of a *leaf* node are equivalent to the point position and its local density.
3. We continue the subdivision of the *internal* nodes only up to a maximum depth d_0 if *leaf* node is not reached.

Fig.2-(B) illustrates the BH-tree CoMs of the nodes at different depths $d = 2, 3, 4$ and 5 with their color-coded (magnitude order: red > green > blue) point densities ϱ. Furthermore, we establish the indexing of the nodes ($\mathbf{N}_d$) as well as their neighborhood indices (κ_d) at every depth d using a label array (L_d) as hash key to retrieve the parent-child information. For instance, the child nodes at depth d of $\mathbf{N}_{d-1,i}$ (a non-empty parent node i at depth $d-1$) will be indexed in order of the Morton's 'Z' curve: $\{2^3 \cdot i + 1, \ldots, 2^3 \cdot i + 8\}$. The supplement has exemplary indexing.

3. The Proposed RPSRNet Method

In this section, we formulate the rigid alignment problem between two BH-trees constructed on the input point cloud pair, and then describe our RPSRNet model for this rigid alignment task by highlighting its different components.

3.1. Mathematical Formulation

Rigid registration of a movable 3D point cloud $\mathbf{Y} = \{\mathbf{y}_1, \ldots, \mathbf{y}_M\} \in \mathbb{R}^{M\times 3}$ (*i.e., source*) with another point

[i]to extract the correspondence weights from our hierarchical deep feature embedding of source and target

[ii]to estimate rotation and translation parameters.

[iii]inspired by Monte-Carlo Importance Sampling Density – if m random samples drawn from a meta-distribution function $q(x)$ of a distribution $p(x)$, then expectation of sample can be expressed as a fraction of weights $w(x) = \frac{p(x)}{q(x)}$, such that the normalized importance is $\sum_{i=1}^{m} w(x_i) = 1$.

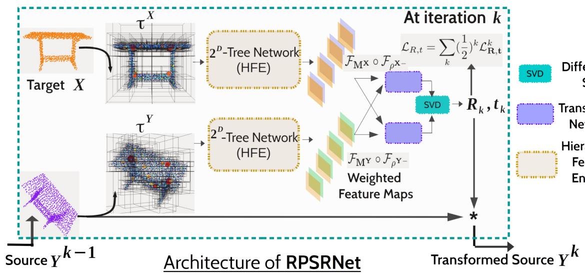

At iteration k
τ^X
τ^Y
Target X
2^D-Tree Network (HFE)
2^D-Tree Network (HFE)
$\mathcal{F}_{\mathbf{M}^{\mathbf{X}}} \circ \mathcal{F}_{\rho^{\mathbf{X}-}}$
$\mathcal{L}_{R,t} = \sum_k (\frac{1}{2})^k \mathcal{L}^k_{\mathbf{R},\mathbf{t}}$
SVD
R_k, t_k
$\mathcal{F}_{\mathbf{M}^{\mathbf{Y}}} \circ \mathcal{F}_{\rho^{\mathbf{Y}-}}$
Weighted Feature Maps
Differential SVD
Transformer Network
Hierarchical Feature Encoder
Source Y^{k-1}
Architecture of RPSRNet
Transformed Source Y^k

Figure 3. A schematic design of iterative RPSR network using 2^{D} tree representation of input point clouds.

cloud $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\} \in \mathbb{R}^{N\times 3}$ (*i.e.*, *target*) is to estimate a 6DoF transformation $\mathbf{T} = \left[\mathbf{R} \in \mathcal{SO}(3) | \mathbf{t} \in \mathbb{R}^3\right]$, which can bring both $\mathbf{Y}$ and $\mathbf{X}$ in a common reference frame. The problem is often formulated as optimization of cost function $\mathbf{U}(\mathbf{R}, \mathbf{t})$ in the form of globally multiply-linked correspondence distance errors:

$$\mathbf{U}(\mathbf{R}, \mathbf{t}, \mathbf{X}, \mathbf{Y}) = \sum_{i,j} \omega_{ij} \|(\mathbf{R}\mathbf{y}_i + \mathbf{t}) - \mathbf{x}_j\|_2^2, \tag{1}$$

where $\|\cdot\|$ denotes the ℓ_2-norm of the distance, and ω_{ij} is correspondence weight. Without ω_{ij} (or with constant values for all $(\mathbf{y}_i,\ \mathbf{x}_j)$ pairs), the influence of putative matches on Eq. (1) cannot be distinguished from spurious matches. In the context of our BH-tree representation of the inputs (*i.e.*, $\mathbf{Y} \rightarrow \tau^{\mathbf{Y}}$ and $\mathbf{X} \rightarrow \tau^{\mathbf{X}}$), we reformulate $\mathbf{U}(\mathbf{R}, \mathbf{t})$ as a multi-scale sum of mean-squared distance errors between the CoMs ($\mathbf{M}^{\mathbf{Y}}_d = \{\mu^y_{d,l}\}$ and $\mathbf{M}^{\mathbf{X}}_d = \{\mu^x_{d,l}\}$) of the respective sets of non-empty tree-nodes ($\mathbf{N}^{\mathbf{Y}}_d = \{\mathbf{n}^y_{d,l}\}$ and $\mathbf{N}^{\mathbf{X}}_d = \{\mathbf{n}^x_{d,l}\}$) at different depths $d \in \{1, 2, \dots, d_0\}$ and labels $l \in \{1, 2, .., (2^3)^d\}$), and weighted by their corresponding ID values $\rho^{\mathbf{Y}-}_d = \{\varrho^{y-}_{d,l}\}$ and $\rho^{\mathbf{X}-}_d = \{\varrho^{x-}_{d,l}\}$). Therefore, the multiply-linked ($\sum_{i,j}$) correspondence distance errors are now applicable on the CoMs of the non-empty nodes at every depth:

$$\mathbf{U}(\mathbf{R}, \mathbf{t}, \tau^{\mathbf{X}}, \tau^{\mathbf{Y}}) = \sum_d \sum_{l,\hat{l}} \varrho^{y-}_{d,l}\, \varrho^{x-}_{d,\hat{l}} \left\| \left(\mathbf{R}\mu^y_{d,l} + t\right) - \mu^x_{d,\hat{l}} \right\|_2^2. \tag{2}$$

Generalized Procrustes methods [34, 27] give a closed-form solution for $\hat{\mathbf{R}}, \hat{\mathbf{t}} = \arg\min_{\mathbf{R},\mathbf{t}} \mathbf{U}(\mathbf{R}, \mathbf{t}, \mathbf{X}, \mathbf{Y})$ in Eq. (1) without input weights ω_{ij} as $\hat{\mathbf{R}} = \mathbf{U}\mathbf{S}\mathbf{V}^{\mathrm{T}}$ and $\hat{\mathbf{t}} = \mathbf{X} - \mathbf{R}\mathbf{Y}$, where $\mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathrm{T}} = \mathrm{SVD}((\mathbf{X} - \overline{\mathbf{X}})(\mathbf{Y} - \overline{\mathbf{Y}})^{\mathrm{T}})$ and $\mathbf{S} = \mathit{diag}(1, \dots, 1, \det(\mathbf{U}\mathbf{V}))$. $\overline{\mathbf{X}}$ and $\overline{\mathbf{Y}}$ are the mean of the inputs $\square$. Analogously, the DGR [13] method shows a closed form solution for $\mathbf{R}, \mathbf{t}$ with correspondence weights ω_{ij} in Eq. (1)

Our RPSRNet is a deep-learning framework (see Fig. 3) to estimate rigid transformation $\mathbf{R}, \mathbf{t}$ in Eq. (2) by using high-dimensional features learned for the CoM and ID attributes of input trees. Our pipeline has three main stages:

Stage 1: We propose a hierarchical feature encoder (HFE) that returns feature maps $\mathcal{F}_{\tau^{\mathbf{Y}}}$ and $\mathcal{F}_{\tau^{\mathbf{X}}}$ for the input trees. The output feature maps from HFE are obtained by fusing (Hadamard product) positional features with features from inverse density channel. Note, that the fused feature maps (position $\circ$ density) $\mathcal{F}_{\mathbf{M}^{\mathbf{Y}}} \circ \mathcal{F}_{\rho^{\mathbf{Y}-}}$ and $\mathcal{F}_{\mathbf{M}^{\mathbf{X}}} \circ \mathcal{F}_{\rho^{\mathbf{X}-}}$ are computed separately.

Stage 2: In this stage, we use a Transformer network [6] to learn the contextual residuals $\phi(\mathcal{F}_{\tau^{\mathbf{Y}}}, \mathcal{F}_{\tau^{\mathbf{X}}})$ and $\phi(\mathcal{F}_{\tau^{\mathbf{X}}}, \mathcal{F}_{\tau^{\mathbf{Y}}})$, between the two embedding $\mathcal{F}_{\tau^{\mathbf{Y}}}$ and $\mathcal{F}_{\tau^{\mathbf{X}}}$. The contextual map $\phi : \mathbb{R}^{64\times 512} \times \mathbb{R}^{64\times 512} \rightarrow \mathbb{R}^{64\times 512}$ denotes the changes between two input embedding (see dimension of output feature from HFE in Sec. 3.2).

Stage 3: We use the differential SVD [49] block on the final score matrix as suggested by DCP [66]:

$$S = \mathrm{SoftMax}([\mathcal{F}_{\tau^{\mathbf{X}}} + \phi(\mathcal{F}_{\tau^{\mathbf{X}}}, \mathcal{F}_{\tau^{\mathbf{Y}}})]\,[\mathcal{F}_{\tau^{\mathbf{Y}}} + \phi(\mathcal{F}_{\tau^{\mathbf{Y}}}, \mathcal{F}_{\tau^{\mathbf{X}}})]^{\mathrm{T}}) \tag{3}$$

3.2. Hierarchical Feature Encoder

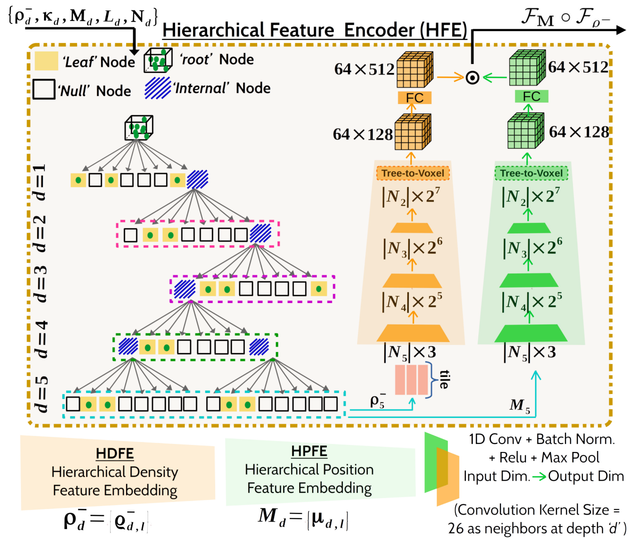

$\{\rho^-_d, \kappa_d, \mathbf{M}_d, L_d, \mathbf{N}_d\}$
Hierarchical Feature Encoder (HFE)
$\mathcal{F}_{\mathbf{M}} \circ \mathcal{F}_{\rho^-}$
'Leaf' Node
'root' Node
'Null' Node
'Internal' Node
64×512
64×512
FC
FC
64×128
64×128
Tree-to-Voxel
Tree-to-Voxel
$|N_2|\times 2^7$
$|N_3|\times 2^6$
$|N_4|\times 2^5$
$|N_5|\times 3$
tile
ρ^-_5
M_5
$d=1$
$d=2$
$d=3$
$d=4$
$d=5$
HDFE
Hierarchical Density Feature Embedding
$\rho^-_d = \{\varrho^-_{d,l}\}$
HPFE
Hierarchical Position Feature Embedding
$M_d = \{\mu_{d,l}\}$
1D Conv + Batch Norm. + Relu + Max Pool
Input Dim. → Output Dim
(Convolution Kernel Size = 26 as neighbors at depth 'd')

Figure 4. Nodes at different depth of Barnes-Hut tree as the input of our hierarchical feature encoder (HFE).

Our HFE (see Fig. 4) takes the BH 2^3-tree attributes – the CoMs, IDs, labels of nodes, and list of 26 neighbours of all non-empty nodes at every depth as input. HFE has has two embedding layers – one to encode the features from the position of CoMs, and the other from the IDs – named as HPFE and HDFE, respectively. The encoder has four hidden layers. *Three* of them encode the features only at the non-empty ($\mathbf{N}^{\mathbf{Y}}_d$, $\mathbf{N}^{\mathbf{Y}}_d$) at depths $d = 5, 4, 3,$, and the *fourth* is *Tree-to-Voxel* layer which converts the non-empty tree nodes at depth 2 into a voxel structure, *i.e.*, $2^3 \cdot 2^3 = 4^3$. The empty locations are zero-filled. The final layer is a *fully connected layer* to up-sample the feature maps. Each embedding layer does a set of operations, *i.e.*, 1D convolution + Batch Normalization [33] + ReLU [45] + Max-Pooling operations. We call this as a basic unit and named HDFE(d) or HPFE(d) based on their input attributes.

Barnes-Hut 2^3-Tree Convolution / Pooling. To perform the tree convolution operation, we use smart indexing of tree nodes, parent-child relation order, and adjacency system of their neighborhoods. Fig. 2-(A) shows a 'Z'-curve (known as Morton's curve) traverse through nodes at the current depth d and increase their label index l by 1 leaving the empty node label as -1. If the parent node was empty, the 'Z'curve skips that octant. Hence, we keep an array for the labels L_d at every depth to serve as hash map (See supplementary material). Next, for the convolution operation on every *non-empty* node, 26 adjacent nodes from the same depth are fetched. Notably adjacent sibling nodes can be empty. At run time, we dispatch a *zero filling* operation on those empty neighbors for indexed based (L_d) 1D convolution. In the reverse case, while Max-Pooling, information flows from child to parent nodes. Hence the same operation of zero filling is done on the empty siblings of a non-empty parent (See supplement for for detailed example indexing on a tree and operations like zero-filling and child-filling). The following is the input and output feature dimensions at depth d using weight filter W after 1D convolution:

$$\mathcal{F}_{out} = \text{Conv1D}(W \in \mathbb{R}^{|\mathbf{N_d}|\times 26+1}, \mathcal{F}_{in} \in \mathbb{R}^{|\mathbf{N_d}|\times 2^{(10-d)}}) \quad (4)$$

3.3. Iterative Registration Loss

RPSRNet predicts the final transformation in multiple steps. On every internal iteration k, a combined loss $\mathcal{L}^k_{\mathbf{R},\mathbf{t}}$ on rotation: $\mathcal{L}^k_{\mathbf{R}} = \|(\mathbf{R}^k_{\text{pred}})^{\text{T}}\mathbf{R}_{\text{gt}} - \mathbf{I}\|^2$, and translation: $\mathcal{L}^k_{\mathbf{t}} = \|(\mathbf{t}^k_{\text{pred}}) - \mathbf{t}_{\text{gt}}\|^2$ is minimized. Our combined loss $\mathcal{L}^k_{\mathbf{R},\mathbf{t}}$ and total loss $\mathcal{L}_{R,t}$ using learnable scale parameters $\sigma_{\mathbf{R}}$ and $\sigma_{\mathbf{t}}$ (for balanced learning of the components) are:

$$\mathcal{L}^k_{\mathbf{R},\mathbf{t}} = \exp(-\sigma_{\mathbf{R}})\mathcal{L}^k_{\mathbf{R}} + \sigma_{\mathbf{R}} + \exp(-\sigma_{\mathbf{t}})\mathcal{L}^k_{\mathbf{t}} + \sigma_{\mathbf{t}} \quad (5)$$

$$\text{and } \mathcal{L}_{R,t} = \sum_k (\frac{1}{2})^k \mathcal{L}^k_{\mathbf{R},\mathbf{t}}, \text{ where } k_0 = \text{max iteration.}$$

The scale parameters $\sigma_{\mathbf{R}}$ and $\sigma_{\mathbf{t}}$ help the network to learn the wide range of transformations.

4. Data Preparation and Evaluation Method

We evaluate on synthetic ModelNet40 [75] dataset which contains 9843 training and 2468 test samples of CAD models under 40 different categories, and also KITTI LiDAR-Odometry [22, 9] as several driving sequences.

ModelNet40 Dataset. In one setup **(M1)**, we use all the ~9.8K training and ~2.4K testing samples under all 40 different object categories. In another setup **(M2)**, to evaluate the generalizability of our network, we choose a training set $\mathcal{T}_M$ with shapes belonging to the first twenty categories, and testing set $\mathcal{T}^*_M$ where shapes are from the other twenty categories that are not seen during training. All input samples are scaled between $[-1, 1]$. The target point clouds are obtained after transforming its clone by random orientations $(\theta_x, \theta_y, \theta_z) \in (0°, 45°]$ and a random linear translation $(\mathbf{t}_x, \mathbf{t}_y, \mathbf{t}_z) \in [-0.5, 0.5]$. To help the deep networks better cope with the data disturbances, another 950 training and 240 testing samples are randomly selected to be preprocessed by four different settings of noise or data disturbances on source point clouds – (i) adding Gaussian $\sim\mathcal{N}(0, 0.02)$ and (ii) uniformly distributed noise $\sim\mathcal{U}(-1.0, 1.0)$ which are 20% of the total points in a sample, (iii) cropping a chunk (approx. 20%) of data, and (iv) jitter each point's position with a displacement tolerance 0.03. The choice of applying one of the four options is random. We prepare five instances of the validation sets with increasing level of noise $(1\%, 5\%, 10\%, 20\%, \text{and } 40\%)$, jitter (increasing displacement threshold), and crops $(1\%, 10\%, 20\%, 30\%, \text{and } 40\%)$.

KITTI Dataset. There are 22 driving sequences in KITTI LiDAR odometry dataset. We prepare two setups – in the first setup **(K1-w/o)**, the ground points are removed from each sample using the label information from SemanticKITTI [9], and in the second setup **(K2-w)** samples remain unchanged. The samples from each driving sequence 00 to 07 are split into 70%, 20%, and 10% as training, testing and validation sets and then merged. The number of frames in the sequences are $4541, 1101, 4661, 801, 271, 2761, 1101,$ and 1101 respectively. Source point clouds from any of these sets are randomly selected frame-indices, whereas the corresponding targets are with next fifth frame-indices. Our RPSRNet can process point clouds with actual size. Due to the memory and scalability issues, PointNetLK [5], DCP-v2 [66], and CPD [44] use inputs down-sampled to 2048 points.

Evaluation Baselines. We compare state-of-the-art neural network-based methods – DCP-v2 [66] and PointNetLK [5] against our RPSRNet. We also evaluate several unsupervised methods – ICP [10], FilterReg [21], CPD [44], FGR [77], GA[i] [25, 26] – for broader analysis (see Sec. A of the supplement for details on training and parameter settings) and ignore some recent CNN-based methods [48, 71, 13] that have higher run time (> 150 milliseconds).

Evaluation Metrics. We use angular deviation φ between the ground truth and predicted rotation matrices $(\mathbf{R}_{gt}, \mathbf{R})$, and similarly, the Euclidean distance error $\mathbf{\Delta t}$ between the ground truth and predicted translations $(\mathbf{t}_{gt}, \mathbf{t})$ as:

$$\varphi = \cos^{-1}\left(0.5(\text{tr}\left(\mathbf{R}^T_{gt}\mathbf{R}\right) - 1)\right), \ \mathbf{\Delta t} = \|\mathbf{t}_{gt} - \mathbf{t}\|. \quad (6)$$

5. Experiments and Results

5.1. Indoor Scenes: Synthetic ModelNet40

Since the DCP and PointNetLK provide the pre-trained model only on the clean version of the ModelNet40 [75] dataset, we retrain the networks on our augmented versions

[i]A GPU implementation

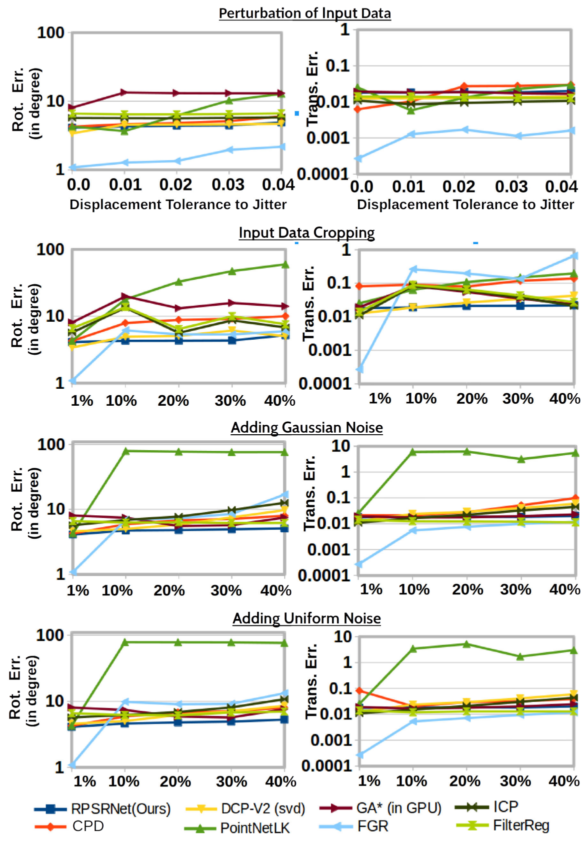


Figure 5. Transformation error evaluated by DCP-v2 [66], PointNetLK [5], GA [25], FGR [77], FilterReg [21] , ICP [10] and our RPSRNet on corrupted ModelNet40 [75]. Five increasing level of disturbances for each of the four types of disturbances. RPSRNet is stable performer and clear winner for most of the test instances.

(M1) and **(M2)**. RPSRNet, DCP-v2 [66], PointNetLK [5], ICP [10], CPD [44], FilterReg [21], FGR [77], and GA [25] are evaluated on all twenty validation sets (for each different type and level of data disturbances). Despite training with the additional 950 samples, both DCP and PointNetLK show a common generalizability issue. The error plots in Fig. 5 show increasing nature prediction inaccuracies for DCP and PointNetLK, with the increasing level of data disturbances. For instance, the transformation error of PointNetLK jumps several times higher when the noise level increases from 1% to 5% – for Gaussian noise, the rotational error φ_{rmse} increases from $2.267°$ to $82.69°$ and the translational error $\mathbf{\Delta t}_{rmse}$ increases from 0.1844 to 5.945. The same increment also occurs for the Uniform noise. When the input data is clean, FGR [77] performs the best with the lowest transformation error (φ_{rmse} = $1.082°$ and $\mathbf{\Delta t}_{rmse}$ = 0.000267). FGR is also consistently superior to all other competing methods at every level of data perturbation, but its translation error is significantly higher in case of partial data registration. With the increasing level of noise, FGR's performance deteriorates further from being the best (at 1% noise level) to the second-worst (at 40% noise level). DCP approach is more robust than PointNetLK but the noise intolerance issue is still pertinent for both methods. RPSRNet is far more robust and stable than the competing methods (Fig. 6 shows qualitative results on few evaluation samples).

5.2. Outdoor Scene: KITTI LiDAR Dataset

Experiments show most unsupervised alignment methods ICP [10], CPD [44], FilterReg [21], and GA [25] all fail to recover mainly the correct translation difference between *source* and *target*, and gets trapped into bad local minima. The evaluation Table 1, reports the final RMSE values on orientations and translations averaged[iv] over *eight* sequences (00 - 07) for all baseline methods. The upper and lower sub-rows indicate **(K1-w/o)** and **(K2-w)** setups respectively. Like in ModelNet40 experiment, FGR [77] performs better among the unsupervised approaches. Our RPSRNet (the last two columns) outperforms all competing methods. For instance, in **K2-w** setup, unsupervided methods ICP, FilterReg, CPD and GA reports Δt_{rmse} as 1.08, 0.77, 1.08, and 1.0 which are 1.87, 1.33, 1.87, and 1.72 times higher than RPSRNet's error value 0.58. In the **K1-w/o** setup, all methods record higher transformation errors, especially on translation part, than RPSRNet. A small rotational inaccuracy in range of $0.5°$ to $3°$ after registration is acceptable because further refinement using ICP or other fast alignment [4] methods reduce such difference. But a prediction error beyond 70 cm for the translation part denotes large dispute. In **K1-w/o** setup: RPSRNet^3 has 1.31 and 1.33 times lower rotational and translational errors than the respective second best methods DCP-v2 and FilterReg. On the other hand, in **K2-w** setup: the same error for rotation is 1.35 lower than second best candidate GA [25], but the translation error is 1.1 times higher as the second best behind DCP-v2. The Fig. 8 shows some qualitative results from our RPSRNet^3 compared to other competing methods on some challenging frames from sequence 03 and 06.

Frame-to-Frame LiDAR Registration. Using the trained model of RPSRNet, we predict the relative transformations on all consecutive pairs of frames in a given sequence. Lets assume that the relative transformation between the sensor poses from frame f to $f+1$ is $\mathbf{T}^f_{4\times4} = [\mathbf{R}_f\, \mathbf{t}_f]$ w.r.t the initial sensor pose $[\mathbf{R}_{\text{init}}, \mathbf{t}_{\text{init}}]$. The trajectory of a point $\vec{p} = (0,0,0,1)^T$ shown in the Fig. 7 is the locus of its spatial positions starting from frame 1 as $\vec{p}^1$ till the frame f as

$$\vec{p}^f = [\mathbf{R}_{\text{init}}\mathbf{t}_{\text{init}}] \cdot ([\mathbf{R}_f\, \mathbf{t}_f] \cdot [\mathbf{R}_{f-1}\, \mathbf{t}_{f-1}] \cdots [\mathbf{R}_1\, \mathbf{t}_1])^{-1} \cdot \vec{p}. \quad (7)$$

for four different sequences. Our measurements are close to the ground-truth.

[iv]The supplementary document provides a detailed evaluation on the individual sequences

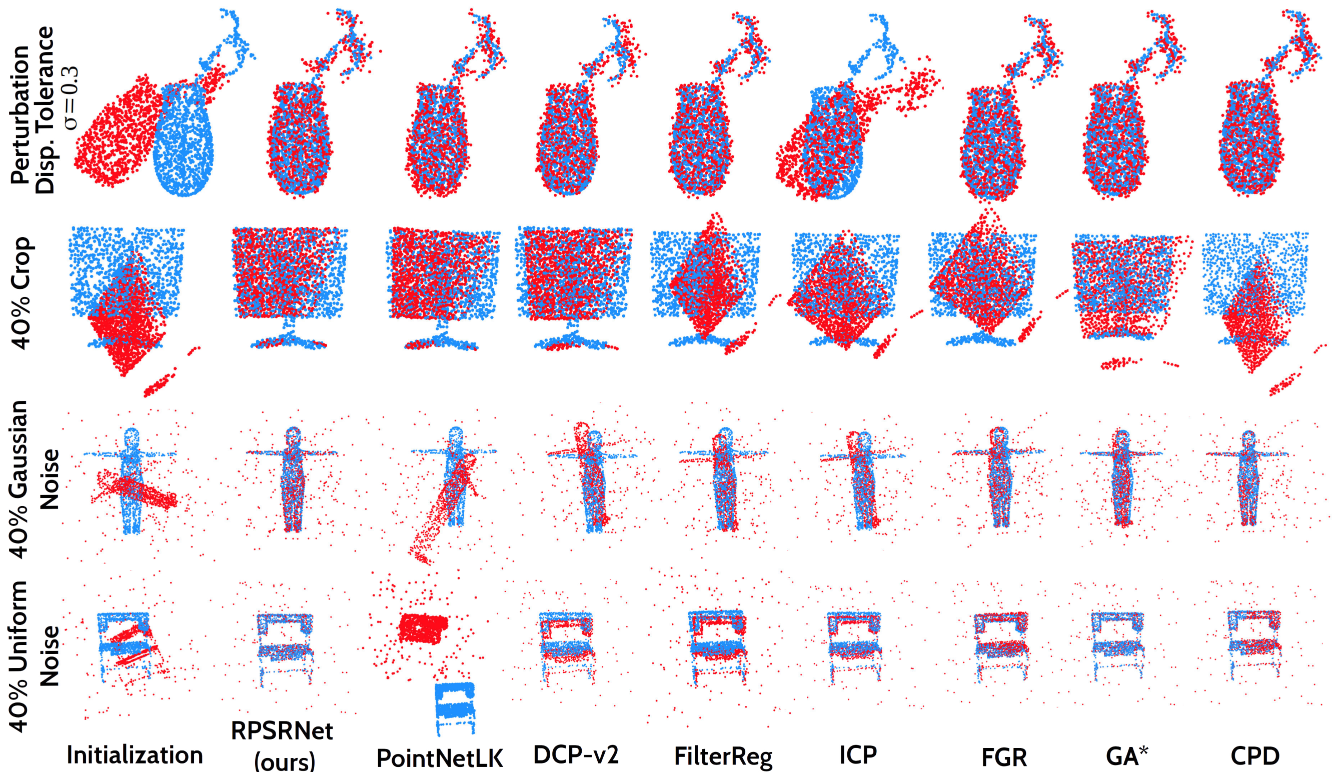


Figure 6. Qualitative registration outcomes on few samples fetched during evaluation of competing methods against noisy data.

On KITTI [22] Dataset									
	CPD [44]	**GA*** [25]	**FGR** [77]	**ICP** [10]	**FilterReg** [21]	**DCP-v2** [66]	**PointNetLK** [5]	**RPSRNet**^{1} **(ours)**	**RPSRNet**^{3} **(ours)**
Seq.	$\varphi_{\text{rmse}}, \mathbf{\Delta t}_{\text{rmse}}$	$\varphi_{\text{rmse}}, \mathbf{\Delta t}_{\text{rmse}}$	$\varphi_{\text{rmse}}, \mathbf{\Delta t}_{\text{rmse}}$	$\varphi_{\text{rmse}}, \mathbf{\Delta t}_{\text{rmse}}$	$\varphi_{\text{rmse}}, \mathbf{\Delta t}_{\text{rmse}}$	$\varphi_{\text{rmse}}, \mathbf{\Delta t}_{\text{rmse}}$	$\varphi_{\text{rmse}}, \mathbf{\Delta t}_{\text{rmse}}$	$\varphi_{\text{rmse}}, \mathbf{\Delta t}_{\text{rmse}}$	$\varphi_{\text{rmse}}, \mathbf{\Delta t}_{\text{rmse}}$
mean	3.55, 1.08	3.30, 1.0	3.29, 0.85	3.15, 1.08	3.08, 0.77	2.92, 0.89	4.02, 1.12	3.13, 0.88	**2.22, 0.58**
	3.03, 1.07	*2.94, 1.02*	*3.25, 1.11*	*3.06, 1.20*	*3.26, 1.20*	*2.96,* ***<u>0.76</u>***	*5.17, 1.20*	*3.03, 1.01*	***<u>2.18</u>****, 0.84*

Table 1. Evaluation on KITTI [22] LiDAR sequences 00 to 07. Each cell in the table denotes the RMSE on angular and translational deviations from ground-truth when averaged over all 8 sequences. The nested-rows (upper and lower) denote the **(K1-w/o)** and **(K2-w)** setups. The lowest transformation errors achieved by any method is highlighted in **bold** or underlined ***<u>bold</u>*** font.

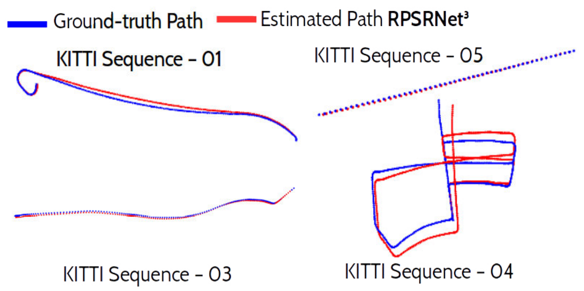


Figure 7. Vehicle trajectories for the KITTI LiDAR sequences are obtained by registering consecutive pairs of frames.

5.3. Runtime and Efficiency

RPSRNet is computationally efficient with the fastest inference time among the competing methods on all tested datasets (see Fig. 9). RPSRNet, FGR [77], ICP [10], FilterReg [21] and GA [25] takes the input with its actual point size for KITTI whereas other methods – DCP-v2 [66], PointNetLK [5] and CPD [44] takes downsampled version (2048 points / frame) of the point clouds otherwise special GPU or CPU memory cards are required. The input point size (2048 points / sample) is constant for all methods in case of ModelNet40 dataset. RPSRNet, DCP-v2, PointNetLK, GA run on an NVIDIA Titan 1080 GPU, whereas CPD, FGR, ICP and FilterReg are run on a 3.0GHz Intel Xeon CPU. Training time per epoch for RPSRNet, DCP-v2, PointNetLK are 5, 18, and 12 minutes, respectively.

5.4. Ablation Study

Why not DCP-v2 with iterative refinement? In this ablation, we implement two hybrid versions of DCP-v2 – (i) DCP-v3: it uses DCP-v2 architecture with additional loss functions (cyclic consistency and embedding disparity)

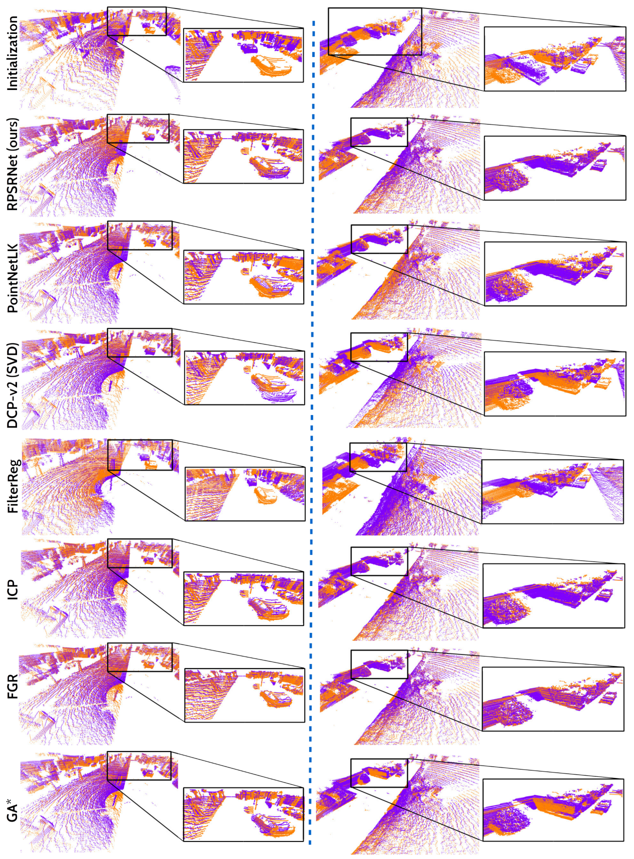


Figure 8. Results of our RPSRNet^3 on some samples from the validation set of KITTI [22, 9] dataset (*left column*: frame 000477 (as $\mathbf{Y}$) and 000482 (as $\mathbf{X}$) of seq-03 and *right column*: frame 000019 (as $\mathbf{Y}$) and 000024 (as $\mathbf{X}$) from the seq-06). Zoomed parts of each image highlights how other competing methods perform in aligning static target objects – *e.g.*, cars and bushes.

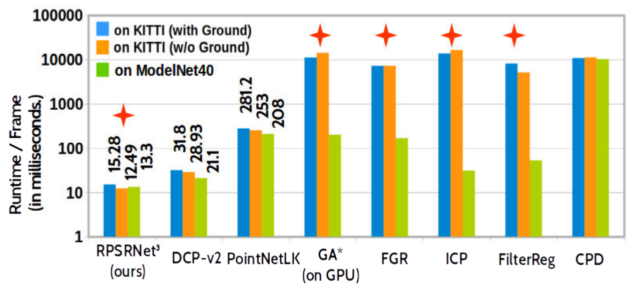


Figure 9. Runtime of the competing methods on KITTI and ModelNet40 with values for the deep-learning based methods. The methods with star marks above take the whole point cloud, whereas others use subsampled version of ~2K.

from its extension PRNet [67] and (ii) DCP-v4: this is DCP-v2 with two internal alignment network blocks. Fig. 10 shows that DCP-v2 and DCP-v3 diverge when tested on clean ModelNet40 with increasing number of external iterations for prediction refinement. On noisy version of ModelNet40, all three versions of DCP (v2, v3, v4) diverge with increasing errors. This test states that an iterative approach similar to us with additional loss functions does not help DCP-v2 [66]. It also reasserts how our novel representation, 2^D-tree convolution, and weighted feature embedding improve the overall accuracy.

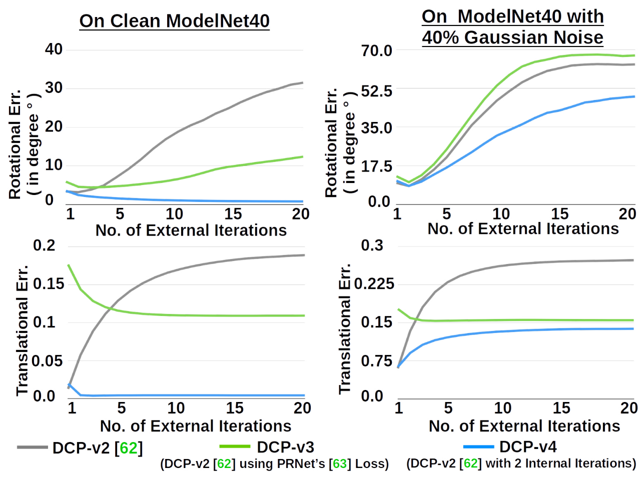


Figure 10. On clean ModelNet40 [75] (*left*) : If we use iterative refinement, the transformation errors of DCP-v2 [66] and DCP-v3 (modified version of DCP-v2 including additional loss functions from PRNet [67]) start diverging to increase. The same observation also found on ModelNet40 with 40% Gaussian noise for all three methods (*right*).

6. Conclusions and Discussion

Our work presents a novel input representation and an end-to-end learning framework for rigid point set registration using 2^3-tree convolution. Extensive experiments on different types of datasets show RPSRNet outperforms competing methods in terms of inference speed and accuracy. The proposed network has limitations to tackle partial-to-partial data registration which is a higher level challenge.

We plan to integrate loss functions which deal with partially overlapping data, *e.g.*, Chamfer distance, and also extend our network architecture for estimating non-rigid displacement fields, *e.g.* scene-flow. Hence, in future, we will implement the deconvolution blocks against each layer of the encoder part in the current network. At present, we build the BH-trees of the input point clouds on CPU (in 4-5 ms./sample) before training starts and pass its attributes to the network during training time as arrays. It disrupts a single application flow. We will integrate a GPU implementation of BH-Tree [12] for direct input data processing.

Acknowledgement

This work was partially supported by the project VIDETE (01IW18002) of the German Federal Ministry of Education and Research (BMBF).

References

[1] Swapna Agarwal and Brojeshwar Bhowmick. 3d point cloud registration with shape constraint. *International Conference on Image Processing (ICIP)*, pages 2199–2203, 2017. 1

[2] Aitor Aldoma, Markus Vincze, Nico Blodow, David Gossow, Suat Gedikli, Radu Bogdan Rusu, and Gary Bradski. Cad-model recognition and 6dof pose estimation using 3d cues. In *International Conference on Computer Vision (ICCV) Workshops*, 2011. 1

[3] Sk Aziz Ali, Vladislav Golyanik, and Didier Stricker. NRGA: Gravitational Approach for Non-rigid Point Set Registration. In *International Conference on 3D Vision (3DV)*, 2018. 1

[4] Sk Aziz Ali, Kerem Kahraman, Christian Theobalt, Didier Stricker, and Vladislav Golyanik. Fast Gravitational Approach for Rigid Point Set Registration with Ordinary Differential Equations. *arXiv e-prints*, 2020. 1, 2, 6

[5] Yasuhiro Aoki, Hunter Goforth, Rangaprasad Arun Srivatsan, and Simon Lucey. Pointnetlk: Robust and efficient point cloud registration using pointnet. In *CVPR*, 2019. 2, 5, 6, 7

[6] Noam Shazeer Ashish Vaswani, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, , and Illia Polo-sukhin. Attention is all you need. In *Advances in Neural Information Processing (NIPS)*, 2017. 3, 4

[7] Michael Bader. *Space-filling curves: an introduction with applications in scientific computing*, volume 9. Springer Science & Business Media, 2012. 2

[8] J. Barnes and P. Hut. A hierarchical o(n log n) force-calculation algorithm. *Nature*, 324:446–449, 1986. 2, 3

[9] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, C. Stachniss, and J. Gall. SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences. In *International Conf. on Computer Vision (ICCV)*, 2019. 5, 8

[10] P. J. Besl and N. D. McKay. A method for registration of 3-d shapes. *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 14(2):239–256, 1992. 1, 5, 6, 7

[11] Bing Jian and B. C. Vemuri. A robust algorithm for point set registration using mixture of gaussians. In *International Conference on Computer Vision (ICCV)*, 2005. 1

[12] Martin Burtscher and Keshav Pingali. Chapter 6 - an efficient cuda implementation of the tree-based barnes hut n-body algorithm. In *GPU Computing Gems Emerald Edition*, Applications of GPU Computing Series, pages 75 – 92. Boston, 2011. 8

[13] Christopher Choy, Wei Dong, and Vladlen Koltun. Deep global registration. In *Computer Vision and Pattern Recognition (CVPR)*, 2020. 2, 4, 5

[14] Christopher Choy, Junha Lee, Rene Ranftl, Jaesik Park, and Vladlen Koltun. High-dimensional convolutional networks for geometric pattern recognition. In *Computer Vision and Pattern Recognition (CVPR)*, 2020. 2

[15] Christopher Choy, Jaesik Park, and Vladlen Koltun. Fully convolutional geometric features. In *International Conference on Computer Vision*, pages 8958–8966, 2019. 2

[16] H. Deng, T. Birdal, and S. Ilic. 3d local features for direct pairwise registration. In *Computer Vision and Pattern Recognition (CVPR)*, 2019. 2

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. 2

[18] Zhen Dong, Bisheng Yang, Yuan Liu, Fuxun Liang, Bijun Li, and Yufu Zang. A novel binary shape context for 3d local surface description. 130:431 – 452, 2017. 2

[19] Benjamin Eckart, Kihwan Kim, and Jan Kautz. Hgmr: Hierarchical gaussian mixtures for adaptive 3d registration. In *European Conference on Computer Vision (ECCV)*, 2018. 1

[20] Gil Elbaz, Tamar Avraham, and Anath Fischer. 3d point cloud registration for localization using a deep neural network auto-encoder. *Computer Vision and Pattern Recognition (CVPR)*, pages 2472–2481, 2017. 2

[21] Wei Gao and Russ Tedrake. Filterreg: Robust and efficient probabilistic point-set registration using gaussian filter and twist parameterization. In *Computer Vision and Pattern Recognition (CVPR)*, 2019. 1, 5, 6, 7

[22] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013. 5, 7, 8

[23] N. Gelfand, L. Ikemoto, S. Rusinkiewicz, and M. Levoy. Geometrically stable sampling for the icp algorithm. In *International Conference on 3-D Digital Imaging and Modeling, 2003. 3DIM 2003. Proceedings.*, pages 260–267, 2003. 1

[24] Zan Gojcic, Caifa Zhou, Jan D Wegner, and Andreas Wieser. The perfect match: 3d point cloud matching with smoothed densities. In *Computer Vision and Pattern Recognition (CVPR)*, pages 5545–5554, 2019. 2

[25] Vladislav Golyanik, Sk Aziz Ali, and Didier Stricker. Gravitational approach for point set registration. *Computer Vision and Pattern Recognition (CVPR)*, 2016. 1, 5, 6, 7

[26] Vladislav Golyanik, Christian Theobalt, and Didier Stricker. Accelerated gravitational point set alignment with altered physical laws. In *International Conference on Computer Vision (ICCV)*, 2019. 1, 2, 5

[27] John C Gower. Generalized procrustes analysis. *Psychometrika*, 40(1):33–51, 1975. 2, 4

[28] Sébastien Granger and Xavier Pennec. Multi-scale em-icp: A fast and robust approach for surface registration. In *European Conference on Computer Vision (ECCV)*. Springer Berlin Heidelberg, 2002. 1

[29] MA Greenspan and M. Yurick. Approximate k-d tree search for efficient icp. In *International Conference on Recent Advances in 3D Digital Imaging and Modeling (3DIM)*, 2003. 1

[30] Johannes Groß, Aljoša Ošep, and Bastian Leibe. Alignnet-3d: Fast point cloud registration of partially observed objects. In *International Conference on 3D Vision (3DV)*, pages 623–632. IEEE, 2019. 1, 2

[31] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Computer Vision and Pattern Recognition (CVPR)*, 2016. 2

[32] Lila Huang, Shenlong Wang, Kelvin Wong, Jerry Liu, and Raquel Urtasun. Octsqueeze: Octree-structured entropy model for lidar compression. In *Computer Vision and Pattern Recognition (CVPR)*, 2020. 2

[33] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, 2015. 4

[34] Wolfgang Kabsch. A solution for the best rotation to relate two sets of vectors. *Acta Crystallographica Section A: Crystal Physics, Diffraction, Theoretical and General Crystallography*, 32(5):922–923, 1976. 4

[35] Marc Khoury, Qian-Yi Zhou, and Vladlen Koltun. Learning compact geometric features. In *International Conference on Computer Vision (ICCV)*, pages 153–161, 2017. 2

[36] Michael Korn, Martin Holzkothen, and Josef Pauli. Color supported generalized-icp. *Computer Vision Theory and Applications (VISAPP)*, 3:592–599, 2014. 1

[37] Huan Lei, Naveed Akhtar, and Ajmal Mian. Octree guided cnn with spherical kernels for 3d point clouds. *Computer Vision and Pattern Recognition (CVPR)*, 2019. 2

[38] Qing Li, Shaoyang Chen, Cheng Wang, Xin Li, Chenglu Wen, Ming Cheng, and Jonathan Li. Lo-net: Deep real-time lidar odometry. In *Computer Vision and Pattern Recognition (CVPR)*, 2019. 1

[39] Weiping Liu, Jia Sun, Wanyi Li, Ting Hu, and Peng Wang. Deep learning on point clouds and its application: A survey. *Sensors*, 19(19), 2019. 2

[40] Yongcheng Liu, Bin Fan, Gaofeng Meng, Jiwen Lu, Shiming Xiang, and Chunhong Pan. Densepoint: Learning densely contextual representation for efficient point cloud processing. In *International Conference on Computer Vision (ICCV)*, 2019. 2

[41] Yongcheng Liu, Bin Fan, Shiming Xiang, and Chunhong Pan. Relation-shape convolutional neural network for point cloud analysis. In *Computer Vision and Pattern Recognition (CVPR)*, 2019. 2

[42] Weixin Lu, Guowei Wan, Yao Zhou, Xiangyu Fu, Pengfei Yuan, and Shiyu Song. Deepvcp: An end-to-end deep neural network for point cloud registration. In *Computer Vision and Pattern Recognition (CVPR)*, pages 12–21, 2019. 2

[43] Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 922–928. IEEE. 2

[44] A. Myronenko and X. Song. Point set registration: Coherent point drift. *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 32(12):2262–2275, 2010. 1, 5, 6, 7

[45] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *International Conference on Machine Learning (ICML)*, 2010. 4

[46] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *International Symposium on Mixed and Augmented Reality (ISMAR)*, 2011. 1

[47] Ehsan Nezhadarya, Ehsan Taghavi, Ryan Razani, Bingbing Liu, and Jun Luo. Adaptive hierarchical down-sampling for point cloud classification. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, June. 2

[48] G. Dias Pais, Pedro Miraldo, Srikumar Ramalingam, Jacinto C. Nascimento, Venu Madhav Govindu, and Rama Chellappa. 3dregnet: A deep neural network for 3d point registration. 2019. 2, 5

[49] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, , and Adam Lerer. Automatic differentiation in pytorch. 2017. 4

[50] Francis Pomerleau, Françoisand Colas and Stéphane Siegwart, Rolandand Magnenat. Comparing icp variants on real-world data sets. *Autonomous Robots*, 34(3):133–148, 2013. 1

[51] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Computer Vision and Pattern Recognition (CVPR)*, pages 652–660, 2017. 2

[52] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in neural information processing systems*, pages 5099–5108, 2017. 2

[53] Rahul Raguram, Jan-Michael Frahm, and Marc Pollefeys. A comparative analysis of ransac techniques leading to adaptive real-time random sample consensus. In *European Conference on Computer Vision (ECCV)*, 2008. 2

[54] Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. Octnet: Learning deep 3d representations at high resolutions. In *Computer Vision and Pattern Recognition (CVPR)*, pages 3577–3586, 2017. 2

[55] Gernot Riegler, Osman Ulusoy, and Andreas Geiger. Octnet: Learning deep 3d representations at high resolutions. In *Computer Vision and Pattern Recognition (CVPR)*, 2017. 2

[56] Szymon Rusinkiewicz and Marc Levoy. Efficient variants of the ICP algorithm. In *International Conference on 3D Digital Imaging and Modeling (3DIM)*, 2001. 1

[57] R. B. Rusu, N. Blodow, and M. Beetz. Fast point feature histograms (fpfh) for 3d registration. In *International Conference on Robotics and Automation (ICRA)*, 2009. 2

[58] R. B. Rusu, N. Blodow, Z. C. Marton, and M. Beetz. Aligning point cloud views using persistent feature histograms. In *International Conference on Intelligent Robots and Systems (IROS)*, 2008. 2

[59] Adam Santoro, David Raposo, David G Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. A simple neural network module for relational reasoning. In *Advances in neural information processing systems*, pages 4967–4976, 2017. 3

[60] Aleksandr Segal, Dirk Hähnel, and Sebastian Thrun. Generalized-icp. In *Robotics: Science and Systems V, University of Washington, Seattle, USA, June 28 - July 1, 2009*, 2009. 1

[61] Hang Su, Varun Jampani, Deqing Sun, Subhransu Maji, Evangelos Kalogerakis, Ming-Hsuan Yang, and Jan Kautz. SPLATNet: Sparse lattice networks for point cloud processing. In *Computer Vision and Pattern Recognition (CVPR)*, 2018. 2

[62] Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. Octree generating networks: Efficient convolutional archi-

tectures for high-resolution 3d outputs. In *International Conference on Computer Vision*, 2017. 2

[63] S. Umeyama. Least-squares estimation of transformation parameters between two point patterns. *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 13(4):376–380, 1991. 1

[64] F. Wang, Y. Zhuang, H. Gu, and H. Hu. Octreenet: A novel sparse 3-d convolutional neural network for real-time 3-d outdoor scene analysis. *IEEE Transactions on Automation Science and Engineering*, 17(2):735–747, 2020. 2

[65] Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun, and Xin Tong. O-CNN: Octree-based Convolutional Neural Networks for 3D Shape Analysis. *ACM Transactions on Graphics (SIGGRAPH)*, 36(4), 2017. 2

[66] Yue Wang and Justin Solomon. Deep closest point: Learning representations for point cloud registration. In *International Conference on Computer Vision (ICCV)*, pages 3523–3532, 2019. 2, 3, 4, 5, 6, 7, 8

[67] Yue Wang and Justin Solomon. Prnet: Self-supervised learning for partial-to-partial registration. In *Advances in Neural Information Processing Systems (NIPS)*, pages 8812–8824, 2019. 2, 3, 8

[68] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (TOG)*, 38(5):1–12, 2019. 2

[69] Wenxuan Wu, Zhongang Qi, and Li Fuxin. Pointconv: Deep convolutional networks on 3d point clouds. In *Computer Vision and Pattern Recognition*, pages 9621–9630, 2019. 2

[70] J. Yang, H. Li, and Y. Jia. Go-icp: Solving 3d registration efficiently and globally optimally. In *International Conference on Computer Vision (ICCV)*, 2013. 1

[71] Z. J. Yew and G. H. Lee. Rpm-net: Robust point matching using learned features. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. 2, 5

[72] Wang Yifan, Shihao Wu, Hui Huang, Daniel Cohen-Or, and Olga Sorkine-Hornung. Patch-based progressive 3d point set upsampling. In *Computer Vision and Pattern Recognition (CVPR)*, pages 5958–5967, 2019. 2

[73] Wentao Yuan, Benjamin Eckart, Kihwan Kim, Varun Jampani, Dieter Fox, and Jan Kautz. Deepgmr: Learning latent gaussian mixture models for registration. In *European Conference on Computer Vision (ECCV)*. Springer, 2020. 2

[74] Ji Zhang and Sanjiv Singh. Low-drift and real-time lidar odometry and mapping. *Autonomous Robots*, 41(2):401416, 2016. 1

[75] Zhirong Wu, S. Song, A. Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Computer Vision and Pattern Recognition (CVPR)*, 2015. 5, 6, 8

[76] Kun Zhou, Minmin Gong, Xin Huang, and Baining Guo. Data-parallel octrees for surface reconstruction. *Transactions on visualization and computer graphics*, 17(5):669–681, 2010. 2

[77] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Fast global registration. In *European Conference on Computer Vision (ECCV)*, 2016. 2, 5, 6, 7

— Supplementary Material —

RPSRNet: End-to-End Trainable Rigid Point Set Registration Network using Barnes-Hut 2^{D}-Tree Representation

Sk Aziz Ali[1,2] Kerem Kahraman[1] Gerd Reis[2] Didier Stricker[1,2]
[1]TU Kaiserslautern [2]German Research Center for Artificial Intelligence (DFKI GmbH), Kaiserslautern

This supplementary document summarizes several relevant details for deeper understanding about our RPSRNet in three main Secs. 1, 2 and 3. The Sec. 1 lists down the parameter settings and training protocols of all benchmark methods alongside our proposed RPSRNet. We also show the effects of different batch size and scaled loss learning on the prediction accuracy of the network. The next Sec. 2 gives insights to our BH 2^D-tree convolution operation using linearized node-indexing. In the end, Sec. 3 provides more insightful results and evaluation of our method on ModelNet40 [12] and KITTI [5] datasets.

1. Evaluation Method and Training Details.

We train the registration networks of DCP [10], PRNet [11], and PointNetLK [1] methods all from scratch (w/o using their pre-trained models) following standard training protocols from the respective studies. Both DCP and its extended version PRNet, and PointNetLK take 250 epochs to train the network with a learning rate of 10^{-3} and using ADAM optimizer. During experiments, we find that PRNet has inherent issues of ill-conditioned feature embedding matrix computations which causes random crashes during training. Hence, we do not evaluate PRNet. While a batch size 10 is set for the DCP, PointNetLK is trained with batch size 32 and internal alignment iterations 10. The optimal hyper-parameters for our RPSRNet are – *required number of epochs*: 250, *batch size*: 16 (see the reason in Sec. 1.1), *learning rate:* 0.0001, *loss dissipation factor* (β): 0.2, *rotational/translational scale* ($\sigma_{\mathbf{R}}/\sigma_{\mathbf{t}}$): -5.0 / -2.5 for KITTI dataset and -9.0 / -10.0 for ModelNet40 dataset, and a *dropout*: 0.2. On the other hand, the maximum number of iterations for all the unsupervised iterative alignment methods ICP [3], CPD [7], FilterReg [4], FGR [13], and GA [6], is set to 200. The mean $\mu = 0.0$ and Gaussian spread $\sigma = 0.05$ are set for FilterReg and CPD (data-based automatic initialization of σ is also possible). FGR method uses Fast Point Feature Histogram (FPFH) [9, 8] descriptor of the input source and target point clouds and use them in their global optimization step to estimate rigid transformation. Hence, it takes a search radius to localize descriptor space for every point. We set the *search-radius* parameter to 0.15. To evaluate the GA, we set its astrophysical parameter settings as – *Gravitational constant* (G): $6.67 \cdot 10^{-3}$, *force softening length* (ϵ): 0.05, *time step* (Δt): 0.05, and *energy dissipation rate*: 0.3.

1.1. Importance of Batch-Size and Scaled Loss

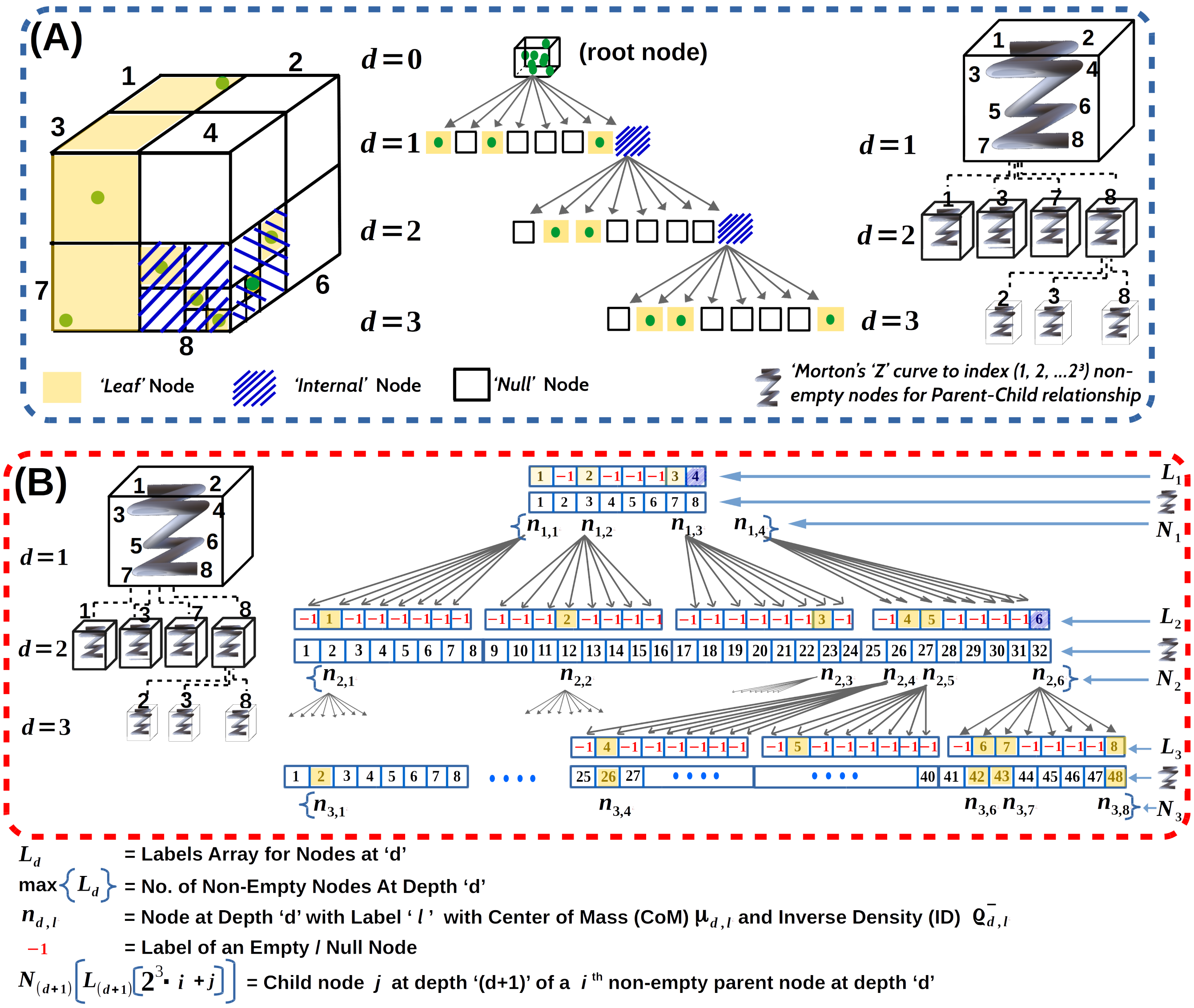
On KITTI [5] and ModelNet40 [12] Datasets					
Batch Sizes	SLL $\mathbb{1}$	(K2-w) $\varphi_{\mathrm{rmse}}, \boldsymbol{\Delta}\mathbf{t}_{\mathrm{rmse}}$	**(K1-w/o)** $\varphi_{\mathrm{rmse}}, \boldsymbol{\Delta}\mathbf{t}_{\mathrm{rmse}}$	**(M1-*seen*)** $\varphi_{\mathrm{rmse}}, \boldsymbol{\Delta}\mathbf{t}_{\mathrm{rmse}}$	**(M2-*unseen*)** $\varphi_{\mathrm{rmse}}, \boldsymbol{\Delta}\mathbf{t}_{\mathrm{rmse}}$
8	✗	1.4394, 2.19	1.481, 1.53	5.82 , 0.04452	5.878, 0.0468
	✓	0.9747, 0.87	0.9234, 0.84	4.3111, 0.0221	4.667, 0.0281
16	✗	1.5823 , 2.24	1.4645, 1.39	6.1468 , 0.0472	6.2004, 0.0423
	✓	0.9889, **0.81**	0.9651, **0.7**	4.588, 0.02377	5.221, 0.0252
32	✗	1.5452 , 2.25	1.3855, 1.44	7.006 , 0.0536	8.613, 0.0542
	✓	**0.9407**, 1.03	**0.9025, 0.7**	**4.1318, 0.0189**	**4.203, 0.0195**

Table 1. Importance of different batch sizes and scaled loss learning (SLL) in RPSRNet: Our method using scaled loss learning (SLL) with 32 input samples in a batch reports minimum rotational and translation RMSE on both the KITTI and ModelNet40 datasets There is a notable exception on $\boldsymbol{\Delta}\mathbf{t}_{\mathrm{rmse}}$ while using the batch size 16 for KITTI (K2-w) setup.

We investigate thoroughly how different batch sizes, scaled version of our loss function (Sec.**3.3** of main matter) and pre-processing step of removing ground points impact on the final prediction accuracies. The evaluation Table 1 shows that with scaling, there are notable improvements on translation error for KITTI [5] dataset. The similar improvement on rotational error is clear for ModelNet40 [12] dataset. For the experiments in the main matter, we select batch size 16.

2. BH 2^{D}-tree Convolution and Node Indexing

This section describes the compact storage mechanism and input signal processing of our BH 2^{D}-tree representation of point cloud. To this end, we have already saved several lists of attributes of the tree nodes at every depth d:


(A)
d=0
(root node)
d=1
d=2
d=3
'Leaf' Node
'Internal' Node
'Null' Node
'Morton's 'Z' curve to index (1, 2, ...2³) non-empty nodes for Parent-Child relationship
(B)
L1
N1
L2
N2
L3
N3

L_d = Labels Array for Nodes at 'd'

$\max\{L_d\}$ = No. of Non-Empty Nodes At Depth 'd'

$n_{d,l}$ = Node at Depth 'd' with Label 'l' with Center of Mass (CoM) $\mu_{d,l}$ and Inverse Density (ID) $\bar{\varrho}_{d,l}$

-1 = Label of an Empty / Null Node

$N_{(d+1)}\left[L_{(d+1)}\left[2^3 \cdot i + j\right]\right]$ = Child node j at depth '(d+1)' of a i^{th} non-empty parent node at depth 'd'

Figure 1. **An exemplary Barnes-Hut 2^D-Tree:** (A) Different types of BH-tree nodes and space-filling Morton's 'Z'-curve for tree traversal. (B) Overview of the labeling, indexing, neighborhood system through parent-child relationships.

(i) $\mathbf{M}_d$, $\mathbf{N}_d$, and ρ_d^- – the center of masses (CoMs) $\mathbf{M}_d$ and the inverse densities (IDs) ρ_d^- of the *non-empty nodes* $\mathbf{N}_d$ linearly store the respective values
(ii) $\mathbf{L}_d$ – *a label array* $\mathbf{L}_d$ stores the sorted indices of non-empty nodes in an increasing order. This array holds **-1** value to those positions where the nodes are empty. The Fig. 1 gives the complete overview of depth-wise labeling and indexing of the nodes.
(iii) κ_d – Finally, we have a set of 26[i] neighbors $\kappa_d(n_{d,l})$ for every non-empty node $n_{d,l} \in \mathbf{N}_d$ at depth $d \in \{1, 2, \ldots\}$ and label $l \in \left\{1, 2, \ldots, (2^3)^d\right\}$. The Fig. 2 describes the adjacency system of a given node.

2^D-tree Convolution. The first hierarchical embedding block (HFE) for positional features takes the CoMs array $\mathbf{M}_d$ which has x, y, and, z coordinates as 3 input channels. The other dimension is the number of non-empty nodes, *i.e.*, $|\mathbf{N}_d|$. Another separate HFE for density features takes the IDs as 1D scalar array. For this we first tile the input to match the same dimension of position channel, *i.e.*, $|\mathbf{N}_d| \times 3$. The integer array $\mathbf{L}_d$ has the storage size of $2^3 \cdot |\mathbf{N}_d|$. For 1D convolution on all $n_{d,l} \in \mathbf{N}_d$ the with kernel size 26 and stride 26, we first perform a *neighbor-fill* operation. To keep our input tensor of fixed size equal to $26 \cdot |\mathbf{N}_d|$, we fill the positions of the array with zero which are empty.

[i]or 27 neighbors including itself

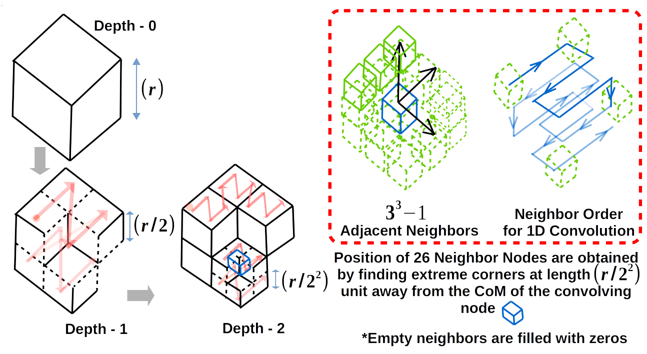

Depth - 0
(r)
(r/2)
Depth - 1
Depth - 2
(r/2²)
3³−1 Adjacent Neighbors
Neighbor Order for 1D Convolution
Position of 26 Neighbor Nodes are obtained by finding extreme corners at length (r/2²) unit away from the CoM of the convolving node
*Empty neighbors are filled with zeros

Figure 2. **Neighbors of a non-empty node at depth d.** 26 adjacent nodes $\kappa_d(n_{d,l})$ of $n_{d,l}$ (*always at the current depth of convolving node*) are the ones which are at the extreme end by $\frac{r}{2^d}$ length from $n_{d,l}$. The *zero-fill* operation sets zero values to the neighbors which are found empty. A linear array stores the neighbors contagiously following the order shown at *right*. Therefore, our 1D convolution kernel size is 26.

Hierarchical Pooling. Pooling is a converse operation of convolution where we fetch the information from child nodes to the parent nodes or in other words from nodes at the higher depth of our BH-tree to the lower depth, *e.g.*, from $d = 3$ 2. After a convolution, the parent-child information needs to be reconstructed for Max-Pooling. Lets observe the parent-child relationship using $\mathbf{L}_d$. The j^{th} child $C_j(P_i)$ of a non-empty parent node $P_i = n_{d,i} = \mathbf{N}_d[i]$ can be retrieved from the array $\mathbf{N}_{(d+1)}$, using the formula:

$$C_j(P_i) = N_{d+1}[L_{d+1}[i \cdot 8 + j]] \qquad (1)$$

Since we store the labels at every depth and do not recompute the neighbors at runtime, we first perform a *child-fill* operation which is to fill attributes' values of the empty neighbor nodes by zero at the current depth. Therefore our input-tensor will have the size of $2^3 \cdot |\mathbf{N}_d|$ as of the label array L_d. For the *neighbor-fill* and *child-fill* operations, we write custom functions in CUDA/C++ for Pytorch-binding.

3. Insightful Results and Evaluations.

We present more insightful registration results for KITTI [5] LiDAR odometry and ModelNet40 [12] datasets (see Table. 2, Fig. 3, and Fig. 4 respectively). The quantitative results on KITTI dataset give important few remarks about other methods:

1. Only on seq-00 and seq-05, FGR and ICP wins the contest of lowest translation error. Although, their difference from RPSRNet is significantly small.
2. In the **(K2)** setup, all methods record higher transformation errors, especially on translational part, on most of the sequences.
3. On seq-01, GA [6], FGR [13], ICP [3] FilterReg [4], PointNetLK [1], and our RPSRNet[1] (single internal iteration) report 39%, 119%, 6%, 171%, 86% and 6.90% increase in the translational error when evaluated on **(K2)** setup compared to the same on **(K1)** setup.

On the other hand, Fig. 3 and 4 show registration outcomes from competing methods on some randomly selected samples. The final alignments applying different methods show that most methods struggle to find globally optimal transformation when input data is largely incomplete.

References

[1] Yasuhiro Aoki, Hunter Goforth, Rangaprasad Arun Srivatsan, and Simon Lucey. Pointnetlk: Robust and efficient point cloud registration using pointnet. In *CVPR*, 2019. 1, 3, 4

[2] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, C. Stachniss, and J. Gall. SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences. In *International Conf. on Computer Vision (ICCV)*, 2019. 5

[3] P. J. Besl and N. D. McKay. A method for registration of 3-d shapes. *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 14(2):239–256, 1992. 1, 3, 4

[4] Wei Gao and Russ Tedrake. Filterreg: Robust and efficient probabilistic point-set registration using gaussian filter and twist parameterization. In *Computer Vision and Pattern Recognition (CVPR)*, 2019. 1, 3, 4

[5] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013. 1, 3, 4, 5

[6] Vladislav Golyanik, Sk Aziz Ali, and Didier Stricker. Gravitational approach for point set registration. *Computer Vision and Pattern Recognition (CVPR)*, 2016. 1, 3, 4

[7] A. Myronenko and X. Song. Point set registration: Coherent point drift. *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 32(12):2262–2275, 2010. 1, 4

[8] R. B. Rusu, N. Blodow, and M. Beetz. Fast point feature histograms (fpfh) for 3d registration. In *International Conference on Robotics and Automation (ICRA)*, 2009. 1

[9] R. B. Rusu, N. Blodow, Z. C. Marton, and M. Beetz. Aligning point cloud views using persistent feature histograms. In *International Conference on Intelligent Robots and Systems (IROS)*, 2008. 1

[10] Yue Wang and Justin Solomon. Deep closest point: Learning representations for point cloud registration. In *International Conference on Computer Vision (ICCV)*, pages 3523–3532, 2019. 1, 4

[11] Yue Wang and Justin Solomon. Prnet: Self-supervised learning for partial-to-partial registration. In *Advances in Neural Information Processing Systems (NIPS)*, pages 8812–8824, 2019. 1

[12] Zhirong Wu, S. Song, A. Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Computer Vision and Pattern Recognition (CVPR)*, 2015. 1, 3

[13] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Fast global registration. In *European Conference on Computer Vision (ECCV)*, 2016. 1, 3, 4

On KITTI [5] Dataset									
	CPD [7]	**GA*** [6]	**FGR** [13]	**ICP** [3]	**FilterReg** [4]	**DCP-v2** [10]	**PointNetLK** [1]	**RPSRNet**^{1} **(ours)**	**RPSRNet**^{3} **(ours)**
Seq.	$\varphi_{\text{rmse}}, \mathbf{\Delta t}_{\text{rmse}}$	$\varphi_{\text{rmse}}, \mathbf{\Delta t}_{\text{rmse}}$	$\varphi_{\text{rmse}}, \mathbf{\Delta t}_{\text{rmse}}$	$\varphi_{\text{rmse}}, \mathbf{\Delta t}_{\text{rmse}}$	$\varphi_{\text{rmse}}, \mathbf{\Delta t}_{\text{rmse}}$	$\varphi_{\text{rmse}}, \mathbf{\Delta t}_{\text{rmse}}$	$\varphi_{\text{rmse}}, \mathbf{\Delta t}_{\text{rmse}}$	$\varphi_{\text{rmse}}, \mathbf{\Delta t}_{\text{rmse}}$	$\varphi_{\text{rmse}}, \mathbf{\Delta t}_{\text{rmse}}$
00	4.99, 1.12	4.82, 1.09	4.77, 0.82	4.72, **0.80**	4.77, 0.80	4.87, 1.07	5.44, 1.27	4.48, 1.01	**3.30**, 0.81
	4.91, 1.39	*4.78, 0.93*	*4.78, 0.83*	*4.70, 0.84*	*4.92, 0.88*	*4.78, 0.80*	*6.11, 1.27*	*4.56,* ***0.73***	***3.31****, 1.0*
01	3.18, 1.54	2.99, 1.74	3.06, 1.10	3.06, 2.27	3.02, 0.89	3.0, 0.95	4.50, 1.33	2.79, 1.28	**2.13, 0.48**
	3.15, 1.88	*3.06, 2.42*	*2.88, 2.42*	*2.83, 2.41*	*2.77, 2.41*	***1.80****,* ***0.68***	*5.22, 1.18*	*3.04, 1.38*	*2.0, 1.03*
02	3.87, 1.28	3.67, 0.98	3.71, 1.02	3.63, 1.0	3.42, 0.69	3.52, 0.76	4.21, 1.0	3.69, 0.73	**2.66, 0.6**
	3.0, 1.11	*3.71, 1.09*	*3.66, 0.9*	*3.66, 0.91*	*3.81, 1.1*	*3.52,* ***0.53***	*5.77, 0.96*	*3.73, 1.10*	***2.88****, 1.11*
03	0.38, 0.88	0.34, 0.72	0.31, 0.59	0.14, 0.56	0.14, 0.49	0.24, 0.66	0.90, 0.81	0.14, 0.72	**0.10, 0.41**
	0.23, 0.58	*0.18, 0.43*	*0.20, 0.74*	*0.13, 0.91*	*0.35, 0.78*	*0.18,* ***0.3395***	*2.1, 1.01*	*0.16, 0.85*	***0.08****, 0.68*
04	2.58, 1.09	2.64, 1.12	2.65, 1.06	2.64, 1.07	1.97, 0.89	2.20, 1.37	3.88, 1.31	2.74, 0.55	**1.11, 0.50**
	2.77, 0.91	*2.64, 0.85*	*2.64, 1.09*	*2.65, 1.11*	*2.11, 1.18*	*2.07, 1.01*	*4.86, 1.27*	*2.28, 0.38*	***1.19****,* ***0.21***
05	3.81, 0.79	3.41, 0.7135	3.29, **0.4142**	2.95, 0.75	3.16, 0.62	2.01, 0.42	4.09, 0.79	3.09, 0.87	**1.91, 0.71**
	3.0, 0.69	*3.30, 0.64*	*3.27, 0.80*	*2.8, 1.12*	*3.89, 0.99*	*3.15,* ***0.63***	*4.2, 1.12*	*3.35, 1.06*	***1.11****, 0.91*
06	4.67, 0.96	4.13, 0.85	4.04, 0.87	3.64, 1.20	4.04, 0.88	3.02, 0.69	3.08, 0.99	4.01, 0.64	**3.0, 0.50**
	2.85, 1.28	***1.55****, 1.02*	*4.13, 1.21*	*3.26, 1.32*	*4.03, 1.29*	*3.74,* ***0.48***	*4.87, 0.97*	*2.64, 0.48*	*3.94, 0.69*
07	4.89, 1.0	4.39, 0.78	4.46, 0.91	4.41, 1.03	4.11, 0.99	4.48, 1.22	6.04, 1.44	4.06, 0.81	**3.58, 0.61**
	4.31, ***0.71***	*4.34, 0.77*	*4.46, 0.88*	*4.37, 0.95*	*4.22, 0.99*	*4.45, 1.58*	*8.21, 1.81*	*4.45, 1.08*	***3.11****, 1.07*

Table 2. Evaluation on KITTI [5] LiDAR sequences 00 to 07. The first row under each sequence reports the RMSE on angular and translational deviations from ground-truth measured on its validation samples in **(K1-w/o)** setup: without 'ground points'. The *next* row under the same sequence reports the same error (with gray-colored and *italic* font-style for better distinction) on the same validation set in **(K2-w)** setup: with 'ground points'. The lowest transformation errors achieved by a method is highlighted in **bold** or underlined ***bold*** font.

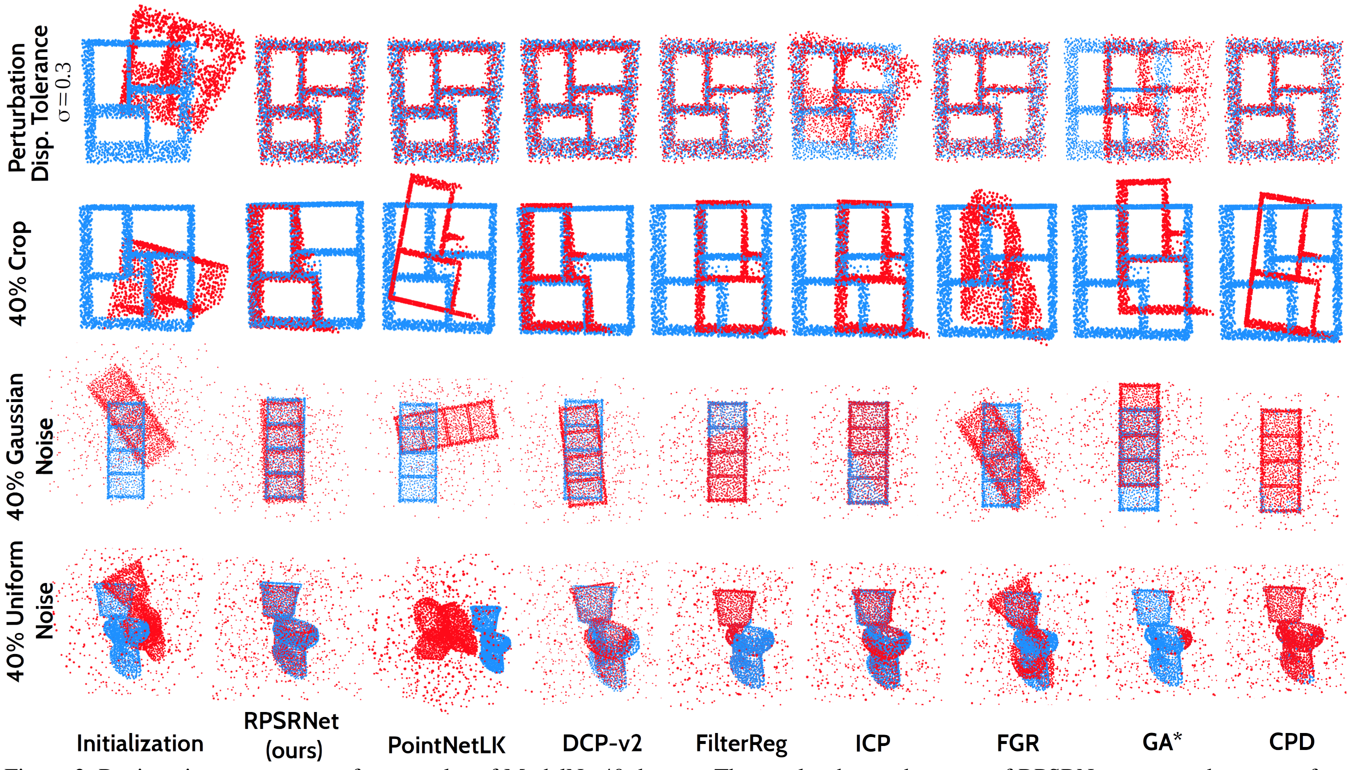


Figure 3. Registration outcomes on few samples of ModelNet40 dataset. The results show robustness of RPSRNet compared to state-of-the-art methods.

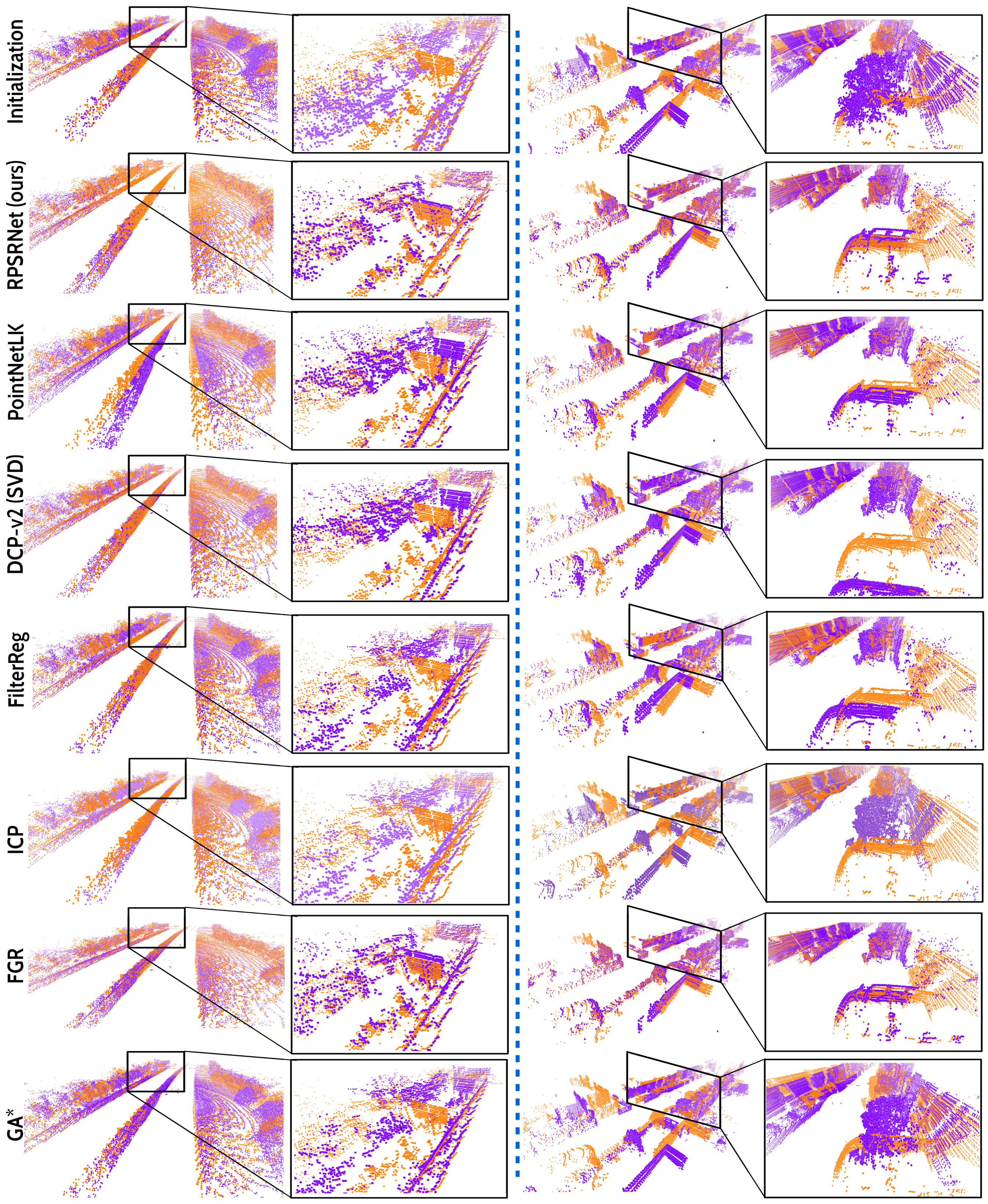


Figure 4. Results of our RPSRNet[3] on some randomly selected samples from the validation set of KITTI [5, 2] dataset (*left column*: frame 000131 (as $\mathbf{Y}$) and 000136 (as $\mathbf{X}$) of seq-01 and *right column*: frame 001721 (as $\mathbf{Y}$) and 001726 (as $\mathbf{X}$) from the seq-05). Zoomed parts of each image highlights how other competing methods perform in aligning static objects of target scene.