REGTR: End-to-end Point Cloud Correspondences with Transformers

Zi Jian Yew  Gim Hee Lee
National University of Singapore
yewzijian@u.nus.edu  gimhee.lee@comp.nus.edu.sg

Abstract

Despite recent success in incorporating learning into point cloud registration, many works focus on learning feature descriptors and continue to rely on nearest-neighbor feature matching and outlier filtering through RANSAC to obtain the final set of correspondences for pose estimation. In this work, we conjecture that attention mechanisms can replace the role of explicit feature matching and RANSAC, and thus propose an end-to-end framework to directly predict the final set of correspondences. We use a network architecture consisting primarily of transformer layers containing self and cross attentions, and train it to predict the probability each point lies in the overlapping region and its corresponding position in the other point cloud. The required rigid transformation can then be estimated directly from the predicted correspondences without further post-processing. Despite its simplicity, our approach achieves state-of-the-art performance on 3DMatch and ModelNet benchmarks. Our source code can be found at https://github.com/yewzijian/RegTR.

1. Introduction

Rigid point cloud registration refers to the problem of finding the optimal rotation and translation parameters that align two point clouds. A common solution to point cloud registration follows the following pipeline: 1) detect salient keypoints, 2) compute feature descriptors for these keypoints, 3) obtain putative correspondences via nearest neighbor matching, and 4) estimate the rigid transformation, typically in a robust fashion using RANSAC. In recent years, researchers have applied learning to point cloud registration. Many of these works focus on learning the feature descriptors [14, 15, 54] and sometimes also the keypoint detection [2, 20, 49]. The final two steps generally remain unchanged and these approaches still require nearest neighbor matching and RANSAC to obtain the final transformation. These algorithms do not take the post-processing into account during training, and their performance can be sensitive to the post-processing choices to pick out the correct correspondences, e.g. number of sampled interest points or distance threshold in RANSAC.

Several works [44, 45, 50] avoid the non-differentiable nearest neighbor matching and RANSAC steps by estimating the alignment using soft correspondences computed from the local feature similarity scores. In this work, we take a slightly different approach. We observe that the learned local features in these works are mainly used to establish correspondences. Thus, we focus on having the network directly predict a set of clean correspondences without RANSAC.

Several works [44, 45, 50] avoid the non-differentiable nearest neighbor matching and RANSAC steps by estimating the alignment using soft correspondences computed from the local feature similarity scores. In this work, we take a slightly different approach. We observe that the learned local features in these works are mainly used to establish correspondences. Thus, we focus on having the network directly predict a set of clean correspondences instead of learning good features. We are motivated by the recent line of works [7, 30] which make use of transformer attention [43] layers to predict the final outputs for various tasks with minimal post-processing. Although attention mechanisms have previously been used in registration of both point clouds [20, 44] and images [35], these works utilize...
attention layers mainly to aggregate contextual information to learn more discriminative feature descriptors. A subsequent RANSAC or optimal transport step is still often used to obtain the final correspondences. In contrast, our Registration Transformer (REGTR) utilizes attention layers to directly output a consistent set of final point correspondences, as illustrated in Fig. 1. Since our network outputs clean correspondences, the required rigid transformation can be estimated directly without additional nearest neighbor matching and RANSAC steps.

Our REGTR first uses a point convolutional backbone [40] to extract a set of features while downsampling the input pair of point clouds. The features of both point clouds are passed into several transformer [43] layers consisting of multi-head self and cross attentions to allow for global information aggregation, while taking into account the point positions through positional encodings to allow the network to utilize rigidity constraints to correct bad correspondences. The resulting features are then used to predict the corresponding transformed locations of the downsampled points. We additionally predict overlap probability scores to weigh the predicted correspondences when computing the rigid transformation. Unlike the more common approach of computing correspondences via nearest neighbor feature matching, which requires interest points to be present at the same locations in both point clouds, our network is trained to directly predict corresponding point locations. As a result, we do not require sampling large number of interest points (e.g. in [20, 54]) or a keypoint detector (e.g. [26, 55]) that produces repeatable points. Instead, we establish correspondences on simple grid subsampled points.

Although our REGTR is simple in design, it achieves state-of-the-art performance on the 3DMatch [54] and ModelNet [46] datasets. It also has fast run times since it does not require running RANSAC on a large number of putative correspondences. In summary, our contributions are:

- We directly predict a consistent set of final point correspondences via self and cross attention, without using the commonly used RANSAC or optimal transport layers.
- We evaluate on several datasets and demonstrate state-of-the-art performance, achieving precise alignments despite using a small number of correspondences.

2. Related Work

Correspondence-based registration. Correspondence-based approaches for point cloud registration first establish correspondences between salient keypoints, followed by robust estimation of the rigid transformation. To accomplish the first step, many keypoint detectors [38, 55] and feature descriptors [34, 41] have been handcrafted. Pioneered by 3DMatch [54], many researchers propose to improve feature descriptors [14, 15, 24, 54] and also keypoint detection [2, 26, 49] by learning from data. Recently, Predator [20] utilizes attention mechanisms to aggregate contextual information to learn more discriminative feature descriptors. Most of these works are trained by optimizing a variant of the contrastive loss [11, 37] between feature descriptors of matching and non-matching points, and rely on a subsequent nearest neighbor matching step and RANSAC to select the correct correspondences.

Learned direct registration methods. Instead of combining learned descriptors with robust pose estimation, some works incorporate the entire pose estimation into the training pipeline. Deep Closest Point (DCP) [44] proposes a learned version of Iterative Closest Point (ICP) [4, 8], and utilizes soft correspondences on learned pointwise features to compute the rigid transform in a differentiable manner. However, DCP cannot handle partial overlapping point clouds and thus later works overcome the limitation by detecting keypoints [45] or using optimal transport layers with an added slack row and column [17, 50]. IDAM [27] considers both feature and Euclidean space during its pairwise matching process. PCAM [6] multiplies the cross-attention matrices at multiple levels to fuse low and high-level contextual information. DeepGM [53] learns to compute point-to-distribution correspondences. A separate group of works [1, 21, 28, 36, 48] rely on global feature descriptors and circumvent the local feature correspondence step. PointNetLK [1] aligns two point clouds by minimizing the distances between their global PointNet [33] features, in a procedure similar to the Lucas-Kanade [3] algorithm. Li et al. [28] extends it to use analytical Jacobians to improve the generalization behavior. OMNet [48] incorporates masking into the global feature to better handle partial overlapping point clouds. Our method utilizes local features and is similar to e.g. [44, 45], but we focus on predicting accurate corresponding point locations via transformer attention layers.

Learned correspondence filtering. The putative correspondences obtained from correspondence-based methods contain outliers, and thus RANSAC is typically used to filter out wrong matches when estimating the required transformation. However, RANSAC is non-differentiable and cannot be used within a training pipeline. Recent works alleviate this problem by modifying RANSAC to enforce differentiability [5], or by learning to identify which of the putative correspondences are inliers [13, 18, 25, 51]. In addition to inlier classification, 3DRegNet [32] also regresses the rigid transformation parameters using a deep network. Different from these works, we directly predict the clean correspondences without explicitly computing the noisy putative correspondences.

Transformers. Transformers [43] propose a novel attention mechanism that makes use of multiple layers of self and cross multi-head attention to exchange information between the input and output. Although originally designed for NLP
tasks, the attention mechanism has recently been shown to be useful for many computer vision tasks [7,16,30,52], and we utilize it in our work to predict point correspondences.

3. Problem Definition

Consider two point clouds \( \mathbf{X} \in \mathbb{R}^{M \times 3} \) and \( \mathbf{Y} \in \mathbb{R}^{N \times 3} \), which we denote as the source and target, respectively. The objective of point cloud registration is to recover the unknown rigid transformation consisting of a rotation \( \mathbf{R} \in \text{SO}(3) \) and translation \( \mathbf{t} \in \mathbb{R}^3 \) that aligns \( \mathbf{X} \) to \( \mathbf{Y} \).

4. Our Approach

Figure 2 illustrates our overall framework. We first convert the input point clouds into a smaller set of downsampled keypoints \( \tilde{\mathbf{X}} \in \mathbb{R}^{M' \times 3} \) and \( \tilde{\mathbf{Y}} \in \mathbb{R}^{N' \times 3} \), which we denote as \( \mathbf{X} \) and \( \mathbf{Y} \), respectively. The KPConv backbone uses a series of ResNet-like blocks and strided convolutions to transform each input point cloud to a reduced set of keypoints \( \tilde{\mathbf{X}} \in \mathbb{R}^{M' \times 3} \) and \( \tilde{\mathbf{Y}} \in \mathbb{R}^{N' \times 3} \), and their associated features \( \tilde{\mathbf{F}}_{\tilde{\mathbf{X}}} \in \mathbb{R}^{M' \times D} \) and \( \tilde{\mathbf{F}}_{\tilde{\mathbf{Y}}} \in \mathbb{R}^{N' \times D} \). These projected features are then fed into several transformer cross-encoder layers (Sec. 4.2) before finally outputting the transformed keypoints \( \hat{\mathbf{X}} \in \mathbb{R}^{M' \times 3} \) and \( \hat{\mathbf{Y}} \in \mathbb{R}^{N' \times 3} \). The cross-attention enables the network to compare points from the two different point clouds, and the self-attention allows points to interact with other points within the same point cloud when predicting its own transformed position, e.g., using rigidity constraints. Note that the network weights are shared among the two point clouds.

4.1. Downsampling and Feature Extraction

We follow [2,20] to adopt the Kernel Point Convolution (KPConv) [40] backbone for feature extraction. The KPConv backbone uses a series of ResNet-like blocks and strided convolutions to transform each input point cloud into a reduced set of keypoints \( \tilde{\mathbf{X}} \in \mathbb{R}^{M' \times 3} \) and \( \tilde{\mathbf{Y}} \in \mathbb{R}^{N' \times 3} \), and their associated features \( \tilde{\mathbf{F}}_{\tilde{\mathbf{X}}} \in \mathbb{R}^{M' \times D} \) and \( \tilde{\mathbf{F}}_{\tilde{\mathbf{Y}}} \in \mathbb{R}^{N' \times D} \). In contrast to [2,20] that subsequently perform upsampling to obtain feature descriptors of the original point cloud resolution, our approach directly predicts the transformed keypoint locations using the downsampled features.

4.2. Transformer Cross-Encoder

The KPConv features from the previous step are linearly projected into a lower dimension \( d = 256 \). These projected features are then fed into \( L = 6 \) transformer cross-encoder layers. Each transformer cross-encoder layer has three sub-layers: 1) a multi-head self-attention layer operating on the two point clouds separately, 2) a multi-head cross-attention layer which updates the features using information from the other point cloud, and 3) a position-wise feed-forward network. The cross-attention enables the network to compare points from the two different point clouds, and the self-attention allows points to interact with other points within the same point cloud when predicting its own transformed position, e.g., using rigidity constraints. Note that the network weights are shared among the two point clouds.

\[ \text{Attention sub-layers.} \quad \text{The multi-head attention} [43] \quad \text{operation in each sub-layer is defined as follows:} \]

\[ \text{MHAAttn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = (\text{Head}_1 \oplus ... \oplus \text{Head}_H) \mathbf{W}^O \quad (1a) \]
\[ \text{Head}_h = \text{Attn} \left( QW^Q_h, KW^K_h, VW^K_h \right), \quad (1b) \]

where \( \oplus \) denotes concatenation over the channel dimension, \( W^Q_h, W^K_h, V^K_h \in \mathbb{R}^{d_{\text{head}} \times d} \) and \( W^O \in \mathbb{R}^{H \times d_{\text{head}} \times d} \) are learned projection matrices. We set the number of heads \( H \) to 8, and \( d_{\text{head}} = d/H \). Each attention head employs a single-head dot product attention:

\[ \text{Attn} (Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d}} \right) V. \quad (2) \]

Residual connections and layer normalization are applied to each sub-layer, and we use the “pre-LN” [47] ordering which we find to be easier to optimize.

The query, key, values are set to the same point cloud in the self-attention layers, i.e. \( \text{MHAttn} (\mathbf{F}_{\tilde{X}}, \mathbf{F}_{\tilde{X}}, \mathbf{F}_{\tilde{X}}) \) for the source point cloud (and likewise for the target point cloud). This allows points to attend to other parts within the same point cloud. For the cross-attention layers, the keys and values are set to be the features from the other point cloud, i.e. \( \text{MHAttn} (\mathbf{F}_{\tilde{X}}, \mathbf{F}_{\tilde{Y}}, \mathbf{F}_{\tilde{Y}}) \) for the source point cloud (and likewise for the target point cloud) to allow each point to interact with points in the other point cloud.

**Position-wise Feed-forward Network.** This sub-layer operates on the features of each keypoint individually. Following its usual implementation [43], we use a two-layer feed-forward network with a ReLU activation function after the first layer. Similar to the attention sub-layers, residual connections and layer normalization are applied.

**Positional encodings.** Unlike previous works [20, 44] that use attentions to learn discriminative features, our transformer layers replace the role of RANSAC and thus requires information of the point positions. Specifically, we incorporate positional information by adding sinusoidal positional encodings [43] to the inputs at each transformer layer.

The outputs of the transformer cross-encoder layers are features \( \tilde{\mathbf{F}}_{\tilde{X}} \in \mathbb{R}^{M' \times d}, \tilde{\mathbf{F}}_{\tilde{Y}} \in \mathbb{R}^{N' \times d} \) which are conditioned on the other point cloud.

### 4.3. Output Decoding

The conditioned features can now be used to predict the coordinates of transformed keypoints. To this end, we use a two-layer MLP to regress the required coordinates. Particularly, the corresponding locations \( \hat{\mathbf{Y}} \in \mathbb{R}^{M' \times 3} \) of the source keypoints \( \tilde{\mathbf{X}} \) in the target point cloud are given as:

\[ \hat{\mathbf{Y}} = \text{ReLU}(\tilde{\mathbf{F}}_{\tilde{X}} \mathbf{W}_1 + \mathbf{b}_1) \mathbf{W}_2 + \mathbf{b}_2, \quad (3) \]

where \( \mathbf{W}_1, \mathbf{W}_2 \) and \( \mathbf{b}_1, \mathbf{b}_2 \) are learnable weights and biases, respectively. We use the hat accent (\( \hat{\cdot} \)) to indicate predicted quantities. A similar procedure is used to obtain the predicted transformed locations \( \hat{\mathbf{X}} \) of the target keypoints.

Alternatively, we also explore the use of a single-head attention layer (\( \text{cf. Tab. 4} \)), where the predicted locations \( \hat{\mathbf{Y}} \) are a weighted sum of the target keypoint coordinates \( \hat{\mathbf{Y}} \):

\[ \hat{\mathbf{Y}} = \text{Attn}(\tilde{\mathbf{F}}_{X} W^Q_{\text{out}}, \tilde{\mathbf{F}}_{Y} W^K_{\text{out}}, \hat{\mathbf{Y}}), \quad (4) \]

where \( \text{Attn} (\cdot) \) is defined previously in Eq. (2), and \( W^Q_{\text{out}}, W^K_{\text{out}} \in \mathbb{R}^{d \times d} \) are learned projection matrices.

In parallel, we separately predict the overlap confidence \( \tilde{o}_X \in \mathbb{R}^{M' \times 1}, \tilde{o}_Y \in \mathbb{R}^{N' \times 1} \) using a single fully connected layer with sigmoid activation. This is used to mask out the influence of correspondences outside the overlap region which are not predicted as accurately.

### 4.4. Estimation of Rigid Transformation

The predicted transformed locations in both directions are first concatenated to obtain the final set of \( M' + N' \) correspondences:

\[ \hat{\mathbf{X}}_{\text{corr}} = \begin{bmatrix} \hat{\mathbf{X}} \\ \hat{\mathbf{Y}} \end{bmatrix}, \quad \hat{\mathbf{Y}}_{\text{corr}} = \begin{bmatrix} \hat{\mathbf{Y}} \\ \hat{\mathbf{Y}} \end{bmatrix}, \quad \hat{o}_{\text{corr}} = \begin{bmatrix} \hat{o}_X \\ \hat{o}_Y \end{bmatrix}. \quad (5) \]

The required rigid transformation can be estimated from the estimated correspondences by solving the following:

\[ \hat{\mathbf{R}}, \hat{\mathbf{t}} = \arg \min_{\mathbf{R}, \mathbf{t}} \sum_i \| \mathbf{R} \hat{\mathbf{x}}_i + \mathbf{t} - \hat{\mathbf{y}}_i \|^2, \quad (6) \]

where \( \hat{\mathbf{x}}_i, \hat{\mathbf{y}}_i, \hat{o}_i \) denote the \( i \)th row of \( \hat{\mathbf{X}}_{\text{corr}}, \hat{\mathbf{Y}}_{\text{corr}}, \hat{o}_{\text{corr}} \), respectively. We follow [18, 50] to solve Eq. (6) in closed form using a weighted variant of the Kabsch-Umezawa [22, 42] algorithm.

### 4.5. Loss Functions

We train our network end-to-end with the ground truth poses \( \{ \mathbf{R}^*, \mathbf{t}^* \} \) for supervision using the following losses:

**Overlap loss.** The predicted overlap scores are supervised using the binary cross entropy loss. The loss for the source point cloud \( \mathbf{X} \) is given by:

\[ \mathcal{L}_o^{\mathbf{X}} = - \frac{1}{M'} \sum_i^{M'} \hat{o}_{\mathbf{x}_i} \log \hat{o}_{\mathbf{x}_i} + (1 - \hat{o}_{\mathbf{x}_i}) \log (1 - \hat{o}_{\mathbf{x}_i}). \quad (7) \]

To obtain the ground truth overlap labels \( o_{\mathbf{x}_i} \), we first compute the dense ground truth labels for the original point cloud in a similar fashion as [20]. Specifically, the ground truth label for point \( \mathbf{x}_i \in \mathbf{X} \) is defined as:

\[ o_{\mathbf{x}_i} = \begin{cases} 1, & \| \mathbf{T}^*(\mathbf{x}_i) - \text{NN}(\mathbf{T}^*(\mathbf{x}_i), \mathbf{Y}) \| < r_o, \\ 0, & \text{otherwise.} \end{cases} \quad (8) \]

where \( \mathbf{T}^*(\mathbf{x}_i) \) denotes the application of the ground truth rigid transform \( \{ \mathbf{R}^*, \mathbf{t}^* \} \), \( \text{NN}(\cdot) \) denotes the spatial nearest neighbor and \( r_o \) is a predefined overlap threshold. We
We thus get a total overlap loss of:

\[ L^X_o = L^X_c + L^X_o. \]

**Correspondence loss.** We apply a \( \ell^1 \) loss on the predicted transformed locations for keypoints in the overlapping region:

\[ L^X_c = \frac{1}{M} \sum_{i} o^X_i \sum_{j} o^Y_j | T^*(\tilde{x}_i) - \tilde{y}_j |, \quad (9) \]

and similarly for the target point cloud. We thus get a total correspondence loss of: \( L_c = L^X_c + L^X_o. \)

**Feature loss.** To encourage the network to take into account geometric properties when computing the correspondences, we apply an InfoNCE [31] loss on the conditioned features. Considering the set of points \( x \in \tilde{X} \) with a correspondence in \( \tilde{Y} \), the InfoNCE loss for the source point cloud is:

\[ L^X_f = -\mathbb{E}_{x \in \tilde{X}} \left[ \log \frac{f(x, p_x)}{f(x, p_x) + \sum_{n_x} f(x, n_x)} \right], \quad (10) \]

where we follow [31] to use a log-bilinear model for \( f(\cdot, \cdot) \):

\[ f(x, c) = \exp(\tilde{f}_x^T W f_c). \quad (11) \]

\( \tilde{f}_x \) denotes the conditioned feature for point \( x \). \( p_x \) and \( n_x \) denote keypoints in \( \tilde{Y} \) which match and do not match \( x \), respectively. They are determined using the positive and negative margins \((r_p, r_n)\) which are set as \((m, 2m)\). \( m \) is the voxel distance used in the final downsampling layer in the KPConv backbone, and all negative points that falls outside the negative margin are utilized for \( n_x \). Since the two point clouds contain the same type of features, we enforce the learnable linear transformation \( W_f \) to be symmetrical by parameterizing it as the sum of a upper triangular matrix \( U_f \) and its transpose, \( \text{i.e. } W_f = U_f + U_f^T \). As explained in [31], Eq. (10) maximizes the mutual information between features for matching points. Unlike [2, 20], we do not use the circle loss [39] that requires the matching features to be similar (w.r.t. \( \ell^2 \) or cosine distance). This is unsuitable since: 1) our conditioned features contains information about the transformed positions, and 2) our keypoints are sparse and are unlikely to be at the same location in the two point clouds, and therefore the geometric features are also different.

Our final loss is a weighted sum of the three components:

\[ L = L_c + \lambda_o L_o + \lambda_f L_f, \]

where we set \( \lambda_o = 1.0 \) and \( \lambda_f = 0.1 \) for all experiments.

## 5. Experiments

### 5.1. Implementation Details

We train our network using AdamW [29] optimizer with a initial learning rate of 0.0001 and weight decay of 0.0001. Gradients are clipped at 0.1. For the 3DMatch dataset, we train for 60 epochs with a batch size of 2, halving the learning rate every 20 epochs. We train on the ModelNet40 dataset for 400 epochs with a batch size of 4, and halving the learning rate every 100 epochs. Training requires around 2.5 and 2 days for 3DMatch and ModelNet40 on a single Nvidia Titan RTX, respectively.

### 5.2. Datasets and Results

**3DMatch.** The 3DMatch dataset [54] contains 46 train, 8 validation and 8 test scenes. We use the preprocessed data from [20] containing voxel-grid downsampled point clouds, and follow them to evaluate on both pairs with \( > 30\% \) overlap (3DMatch) and \( 10 - 30\% \) overlap (3DLtMatch). Each input point cloud contains an average of about 20,000 points, which are downsampled to an average of 345 points by our KPConv backbone. We perform training data augmentation by applying small rigid perturbations, jittering of the point locations and shuffling of points.

Following [2, 10, 20], we evaluate using Registration Recall (RR) which measures the fraction of successfully registered pairs, defined as having a correspondence RMSE below 0.2m. We also evaluate on the Relative Rotation Errors (RRE) and Relative Translation Errors (RTE) that measures the accuracy of successful registrations. We follow [20] and compare against several recent learned correspondence-based algorithms [2, 14, 19, 20]. These algorithms tend to perform better with a larger number of sampled interest points, and therefore we only show the results for the maximum number (5000) of sampled points. Since Predator [20] obtains the highest registration recall when using 1000 interest points, we also reran their open-source code.

There was a bug in [20] which decreased the performance, and we list the improved results using its corrected version.

---

\( ^2 \)The initial Predator code had a bug which decreased the performance.
Table 1. Performance on 3DMatch and 3DLoMatch datasets. Results for 3DSN, FCGF, D3Feat and Predator-5k are from [20].

| Method          | 3DMatch RR(%) | 3DMatch RRE(◦) | 3DMatch RTE(m) | 3DLoMatch RR(%) | 3DLoMatch RRE(◦) | 3DLoMatch RTE(m) |
|-----------------|---------------|----------------|----------------|----------------|------------------|------------------|
| 3DSN [19]       | 78.4          | 2.199          | 0.071          | 33.0           | 3.528            | 0.103            |
| FCGF [14]       | 85.1          | 1.949          | 0.066          | 40.1           | 3.147            | 0.100            |
| D3Feat [2]      | 81.6          | 2.161          | 0.067          | 37.2           | 3.361            | 0.103            |
| Predator-5k [20]| 89.0          | 2.029          | 0.064          | 59.8           | 3.048            | 0.093            |
| Predator-1k [20]| 90.5          | 2.062          | 0.068          | 62.5           | 3.159            | 0.096            |
| Predator-NR [20]| 62.7          | 2.582          | 0.075          | 24.0           | 5.886            | 0.148            |
| OMNet [48]      | 35.9          | 4.166          | 0.105          | 8.4            | 7.299            | 0.151            |
| DGR [13]        | 85.3          | 2.103          | 0.067          | 48.7           | 3.954            | 0.113            |
| PCAM [6]        | 85.5          | 1.808          | 0.059          | 54.9           | 3.529            | 0.099            |
| Ours            | 92.0          | 1.567          | 0.049          | 64.8           | 2.827            | 0.077            |

Table 2. Evaluation results on ModelNet40 dataset. Results of DCP-v2, RPM-Net and Predator are taken are from [20].

| Method      | ModelNet | ModelLoNet |
|-------------|----------|------------|
|             | RRE      | RTE  | CD  | RRE      | RTE  | CD  |
| PointNetLK  | 0.297    | 0.0235 | 48.567 | 0.507 | 0.0367 |
| OMNet [48]  | 0.032    | 0.00015 | 6.517  | 0.129 | 0.0074 |
| DCP-v2 [44] | 0.171    | 0.0117 | 16.501 | 0.300 | 0.0268 |
| RPM-Net [50]| 0.018    | 0.000085 | 7.342  | 0.124 | 0.0050 |
| Predator    | 0.019    | 0.000089 | 5.235  | 0.132 | 0.0083 |
| Ours        | 0.014    | 0.000078 | 3.930  | 0.087 | 0.0037 |

Table 3. Run time in milliseconds on the 3DMatch benchmark test set. Similar pre-processing is used for D3Feat, Predator and our algorithm, but our algorithm uses a faster GPU implementation.

ModelNet40. We also evaluate on the ModelNet40 [46] dataset comprising synthetic CAD models. We follow the data setting in [20, 50], where the point clouds are sampled randomly from mesh faces of the CAD models, cropped and subsampled. Following [20], we evaluate on two partial overlap settings: ModelNet which has 73.5% pairwise overlap on average, and ModelLoNet which contains a lower 53.6% average overlap. We train only on ModelNet, and perform direct generalization to ModelLoNet. We follow [20, 50] and measure the performance using Relative Rotation Error (RRE) and Relative Translation Error (RTE) on all point clouds, as well as the Chamfer distance (CD) between the registered scans.

The results are shown in Tab. 2, with example qualitative results in Figs. 8e and 8f. We compare against recent correspondence-based [20] and end-to-end registration methods [1, 44, 48, 50]. Predator [20] samples 450 points in this experiment. OMNet [48] was originally trained only on axis-asymmetrical categories, and we retrained it on all categories to obtain a slightly improved result. As noted in [20], many of the end-to-end registration methods are specifically tuned for ModelNet. RPM-Net [50] additionally uses surface normal information. Despite this, our REGTR substantially outperforms all baseline methods in all metrics under both normal overlap (ModelNet) and low overlap (ModelLoNet) regimes. Our learned attention mechanism is able to outperform the optimal transport (in RPM-Net) and RANSAC step (in Predator).

5.3. Analysis

We perform further analysis in this section to better understand our algorithm behavior. All experiments in this section are performed on the larger 3DMatch dataset.

Runtime. We compare the runtime of REGTR against several algorithms in Tab. 3. We conducted the test on a single Nvidia Titan RTX with Intel Core i7-6950X @ 3.0GHz and 64GB RAM. Our entire pipeline runs under 100ms, and is feasible for many real-time applications. The time consuming step for correspondence-based algorithms
Figure 4. Histogram and CDF plot of $\ell^2$ errors of predicted correspondences. 95.7% of predicted correspondences have errors below 0.112m, the median keypoint-to-keypoint distance (denoted by red dashed line). Errors are clipped at 0.2m for clarity, with only 3.0% of predicted correspondences exceeding this error.

is the pose estimation which includes feature matching and RANSAC. For example, excluding preprocessing required for the KPConv backbone, Predator [20] takes 234ms for the registration when sampling just 1,000 points. DGR [13] and PCAM [6] also require long times for pose estimation due to their robust refinement and RANSAC safeguard. Although the global feature-based OMNet runs faster than our algorithm, we note that it is unable to obtain good accuracy on the 3DMatch dataset.

Accuracy of predicted correspondences. In Fig. 4, we plot the distribution of $\ell^2$-error of the predicted correspondences for keypoints within the overlap region (where $o_{x_i}, o_{y_i} > 0.5$) for the 3DMatch test set. The median error of our predicted correspondences is 0.028m, which is significantly smaller than the median distance between keypoints (0.112m). For comparison, an oracle matcher that matches every keypoint to the closest keypoint using the ground truth pose obtains a median error of 0.071m. Our direct prediction of correspondences is able to overcome the resolution issues from the downsampling, and thus explains the precise registration obtained by REGTR.

We also visualize the predicted correspondences of a point cloud pair in Fig. 5. The short green error lines in Fig. 5b indicate our predicted transformed locations are highly accurate within the overlap region, even in non-informative regions (e.g. floor). Interestingly, correspondence for points outside the overlap region are projected to near the overlap boundaries. These observations suggest that REGTR is able to make use of rigidity constraints to guide the positions of the predicted correspondences.

Visualization of attention. In Fig. 6, we visualize the attention for a point on the ground for the same point cloud pair from the previous section. Since the point lies in a non-informative region, the point attends to multiple similar looking regions in the other point cloud in the first transformer layer (Fig. 6a). At the sixth layer, the point is confident of its position and mostly focuses on its correct corresponding location (Fig. 6b). The self-attention in Fig. 6c shows that the point makes use of feature rich regions within the same point cloud to help localize its correct location.

5.4. Ablations

We further perform ablation studies on the 3DMatch dataset to understand the role of various components.

Number of cross-encoder layers. We evaluate how the performance varies with the number of cross-encoder layers in Fig. 7. Our network cannot function without any cross-encoder layers, and we show the registration recall for two to eight layers. Performance generally improves with more cross-encoder layers, but saturates around $L = 6$ encoder layers (which we use for all our experiments).

Comparison with RANSAC. We compare with a version of REGTR where we replace our output decoder with a two-layer MLP which outputs 256D feature descriptors and a parallel single-layer decoder that outputs the overlap score. The pose is subsequently estimated using RANSAC on nearest neighbor feature matches. The network is trained
Figure 8. Example qualitative registration results for (a, b) 3DMatch, (c, d) 3DLoMatch (e) ModelNet40, and (f) ModelLoNet.

| Method               | 3DMatch RR(%) | RRE(◦)  | RTE(m) | 3DLoMatch RR(%) | RRE(◦) | RTE(m) |
|----------------------|---------------|---------|--------|----------------|---------|--------|
| RANSAC baseline      | 87.7          | 2.296   | 0.072  | 52.7           | 4.038   | 0.112  |
| Weighted coor.       | 90.9          | 1.468   | 0.046  | 63.1           | 2.540   | 0.076  |
| No feature loss      | 90.4          | 1.638   | 0.049  | 61.9           | 2.898   | 0.083  |
| Circle loss          | 90.0          | 1.696   | 0.051  | 61.2           | 3.217   | 0.092  |
| Loss on all layers   | 83.9          | 1.781   | 0.053  | 46.8           | 3.448   | 0.101  |
| REGTR                | 92.0          | 1.357   | 0.049  | 64.8           | 2.827   | 0.077  |
| REGTR+RANSAC         | 91.2          | 1.607   | 0.049  | 63.3           | 2.753   | 0.079  |

Table 4. Ablation of components and losses using only $L_\infty$ and $L_F$ (using Circle Loss [39]). This results in a lower registration recall, and significantly higher rotation and translation errors as the downsampled keypoints do not provide enough resolution for accurate registration.

We also try applying RANSAC to the predicted correspondences from REGTR to see if the performance can further improve. Row 7 of Tab. 4 shows marginally worse registration recall. This indicates that RANSAC is no longer beneficial on the predicted correspondences that are already consistent with a rigid transformation.

Decoding scheme. We compare with decoding the coordinates as a weighted sum of coordinates (Eq. 4). Compared to our simpler approach of regressing the coordinates using a MLP, computing the coordinates as a weighted sum achieves a slightly better RTE and RRE, but lower registration recall. See rows 2 and 6 of Tab. 4.

Loss ablations. Rows 3-6 of Tab. 4 shows the registration performance with different loss configurations. Without the feature loss to guide the network outputs, the network obtained a 1.6% and 2.9% lower registration recall for 3DMatch and 3DLoMatch, respectively. Using circle loss from [20] also underperformed as the network cannot incorporate positional information into the feature as effectively. We also experimented with applying the losses on all $L = 6$ transformer layers (instead of just the final one) with the output decoder shared among all cross-encoder layers. This additional supervision led to a 8.1% (3DMatch) and 18.0% (3DLoMatch) lower registration recall. Consequently, we only apply the supervision for the output of the last cross-encoder layer.

6. Limitations

Our use of transformers layers with quadratic complexity prevents its use on large number of points, and we can only apply them on downsampled point clouds. Although our direct correspondence prediction alleviates the resolution issue, it is possible that a finer resolution can result in even higher performance. We have tried transformer layers with linear complexity [12, 23], but that obtained subpar performance. Alternate workarounds include using sparse attention [9], or performing a coarse-to-fine registration.

7. Conclusions

We propose the REGTR for rigid point cloud registration, which directly predicts clean point correspondences using multiple transformer layers. The rigid transformation can then be estimated from the correspondences without further nearest neighbor feature matching nor RANSAC steps. The direct prediction of correspondences overcomes the resolution issues from the use of downsampled features, and our method achieves state-of-the-art performance on both scene and object point cloud datasets.

Acknowledgement. This research/project is supported in part by the National Research Foundation, Singapore under its AI Singapore Program (AISG Award No: AISG2-RP-2020-016), and the Tier 2 grant MOE-T2EP2012-0011 from the Singapore Ministry of Education.
References

[1] Yasuhiro Aoki, Hunter Goforth, Rangaprasad Arun Srivatsan, and Simon Lucey. Pointnetlk: Robust & efficient point cloud registration using pointnet. In CVPR, pages 7163–7172, 2019. 2, 6

[2] Xuyang Bai, Zixin Luo, Lei Zhou, Hongbo Fu, Long Quan, and Chiew-Lan Tai. D3Feat: Joint learning of dense detection and description of 3d local features. In CVPR, pages 6359–6367, 2020. 1, 2, 3, 5, 6

[3] Simon Baker and Iain Matthews. Lucas-kanade 20 years on: A unifying framework. IEEE TPAMI, 14(2):239–256, 1992. 2

[4] Eric Brachmann, Alexander Krull, Sebastian NOWOZIN, and Tolga Birdal. Learning multiview 3d point cloud registration. In CVPR, pages 1759–1769, 2020. 2, 4

[5] Paul J. Besl and Neil D. McKay. A method for registration of 3-d shapes. IEEE TPAMI, 14(2):239–256, 1992. 2

[6] Anh-Quan Cao, Gilles Puy, Alexandre Boulch, and Renaud Marlet. PCAM: Product of cross-attention matrices for rigid registration of point clouds. In ICCV, pages 13229–13238, 2021. 2, 6, 7

[7] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In ECCV, pages 213–229, 2020. 2, 3

[8] Y. Chen and G. Medioni. Object modeling by registration of multiple range images. In ICRA, pages 2724–2729 vol.3, 1991. 2

[9] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. arXiv preprint arXiv:1904.10509, 2019. 8

[10] Jurgen Fischer, Martin Simon, Florian Olsner, Stefan Milz, Horst-Michael Groß, and Patrick Mader. Stickypillars: Robust and efficient feature matching on point clouds using graph neural networks. In CVPR, pages 313–323, 2021. 2

[11] Kai Fischer, Martin Simon, Florian Olsner, Stefan Milz, Horst-Michael Groß, and Patrick Mader. Stickypillars: Robust and efficient feature matching on point clouds using graph neural networks. In CVPR, pages 313–323, 2021. 2

[12] Zan Gojcic, Kaifa Zhou, Jan D Wenger, Leonidas J Guibas, and Tolga Birdal. Learning multiview 3d point cloud registration. In CVPR, pages 1759–1769, 2020. 2, 4

[13] Zan Gojcic, Kaifa Zhou, Jan D Wenger, and Andreas Wieser. The perfect match: 3d point cloud matching with smoothed densities. In CVPR, pages 5545–5554, 2019. 5, 6

[14] Zan Gojcic, Kaifa Zhou, Jan D Wenger, and Andreas Wieser. The perfect match: 3d point cloud matching with smoothed densities. In CVPR, pages 5545–5554, 2019. 5, 6

[15] Shengyu Huang, Zan Gojcic, Mikhail Usvyatsov, Andreas Wieser, and Konrad Schindler. Predator: Registration of 3d point clouds with low overlap. In CVPR, pages 4267–4276, 2021. 1, 2, 3, 4, 5, 6, 7, 8

[16] Xiaoshui Huang, Guofeng Mei, and Jian Zhang. Feature-metric registration: A fast semi-supervised approach for robust point cloud registration without correspondences. In CVPR, pages 11366–11374, 2020. 2

[17] 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

[18] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In ICML, pages 5156–5165, 2020. 8

[19] Marc Khoury, Qian-Yi Zhou, and Vladlen Koltun. Learning compact geometric features. In Proceedings of the IEEE international conference on computer vision, pages 153–161, 2017. 2

[20] Junha Lee, Seungwook Kim, Minsu Cho, and Jaesik Park. Deep hough voting for robust global registration. In ICCV, pages 15994–16003, 2021. 2

[21] Jiahao Li, Changhao Zhang, Ziyao Xu, Hangning Zhou, and Jiahao Li, Changhao Zhang, Ziyao Xu, Hangning Zhou, and Chi Zhang. Iterative distance-aware similarity matrix convolution with mutual-supervised point elimination for efficient point cloud registration. In ECCV, pages 378–394, 2020. 2

[22] Xueqian Li, Jhony Kaesemodel Pontes, and Simon Lucey. Pointnetlk revisited. In CVPR, pages 12763–12772, 2021. 2

[23] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In ICLR, 2019. 5

[24] Jiafeng Liu, Yang Liu, and Armand Joulin. Dsan: Deep self-supervised monocular depth estimation. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 6148–6157, 2021. 2

[25] Junha Lee, Seungwook Kim, Minsu Cho, and Jaesik Park. Deep hough voting for robust global registration. In ICCV, pages 15994–16003, 2021. 2

[26] Xiaohui Huang, Guofeng Mei, and Jian Zhang. Feature-metric registration: A fast semi-supervised approach for robust point cloud registration without correspondences. In CVPR, pages 11366–11374, 2020. 2

[27] Marc Khoury, Qian-Yi Zhou, and Vladlen Koltun. Learning compact geometric features. In Proceedings of the IEEE international conference on computer vision, pages 153–161, 2017. 2

[28] Xiaohui Huang, Guofeng Mei, and Jian Zhang. Feature-metric registration: A fast semi-supervised approach for robust point cloud registration without correspondences. In CVPR, pages 11366–11374, 2020. 2

[29] Marc Khoury, Qian-Yi Zhou, and Vladlen Koltun. Learning compact geometric features. In Proceedings of the IEEE international conference on computer vision, pages 153–161, 2017. 2

[30] Ishan Misra, Rohit Girdhar, and Armand Joulin. An end-to-end transformer model for 3d object detection. In ICCV, pages 2906–2917, 2021. 1, 3

[31] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. arXiv preprint arXiv:1807.03748, 2018. 5

[32] G Dias Pais, Srikumar Ramalingam, Venu Madhav Govindu, Jacinto C Nascimento, Rama Chellappa, and Pedro Miraldo. 3DRegNet: A deep neural network for 3D point registration. In CVPR, pages 7193–7203, 2020. 2

[33] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In CVPR, 2017. 2
