

PointNetLK Revisited

Supplementary Material

Xueqian Li¹ Jhony Kaesemodel Pontes¹ Simon Lucey^{2,3}
¹Argo AI ²The University of Adelaide ³Carnegie Mellon University

xueqianl@alumni.cmu.edu jpontes@argo.ai simon.lucey@adelaide.edu.au

1. Introduction

Here we provide additional information to supplement our main submission regarding the derivations of our voxelized analytical Jacobian, our network design strategies, different loss functions, and extra experimental results.

2. Voxelized Analytical Jacobian Derivation

We show a 2D illustration of our voxelization strategy in Fig. 1 (in our experiments, we voxelized the point cloud in the 3D space).

In our main submission, we used a conditioned warp Jacobian $\frac{\partial \xi_{\mathbf{v}_m}}{\partial \xi^T}$ to transform the local voxel Jacobians to the global coordinate frame as

$$\mathbf{J}_g = [\mathbf{J}_{\mathbf{v}_1}, \dots, \mathbf{J}_{\mathbf{v}_M}] \begin{bmatrix} \left(\frac{\partial \xi_{\mathbf{v}_1}}{\partial \xi^T} \right) \\ \vdots \\ \left(\frac{\partial \xi_{\mathbf{v}_M}}{\partial \xi^T} \right) \end{bmatrix}. \quad (1)$$

We explored two different ways to derive the conditioned warp Jacobian: an explicit and an implicit derivation.

2.1. Explicit derivation

The explicit derivation considers that the local voxel frame can be transformed to the global frame explicitly. Formally, let $\{v\}$ be the voxel coordinate frame, and $\{g\}$ be the global coordinate frame. The twist parameters w.r.t the local frame $\{v\}$ are $\xi_v = (\omega_v, v_v)$, and the twist parameters w.r.t the global frame $\{g\}$ are $\xi_g = (\omega_g, v_g)$, where ω and v are the rotations and translations along the xyz axis respectively. We define the transformation matrix, in homogeneous coordinates, from the local frame $\{v\}$ to the global frame $\{g\}$ as

$$T_{gv} = \begin{bmatrix} \mathbf{R} & \mathbf{p} \\ \mathbf{0} & 1 \end{bmatrix}, \quad (2)$$

and the transformation from the global frame $\{g\}$ to the local frame $\{v\}$ as T_{vg} . Then, we can formulate the relation-

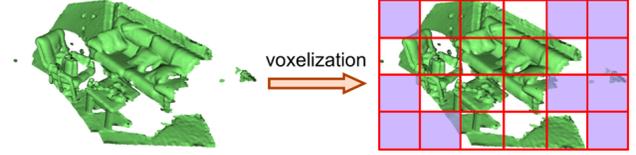


Figure 1. **How the voxelization is performed?** Given a complex scene, we voxelize it into regularly spaced voxels as illustrated by the grid in **red**. Each voxel containing enough points, defined by a threshold, is used to extract the PointNet feature and compute its local Jacobian. The voxels that are empty or with very few points are discarded, as shown by the **purple** shaded areas.

ship between these two transformation as

$$[\xi_g] = T_{gv} [\xi_v] T_{gv}^{-1}, \quad (3)$$

where

$$[\xi_g] = \begin{bmatrix} [\omega_g] & v_g \\ 0 & 0 \end{bmatrix}, \quad (4)$$

$$[\xi_v] = \begin{bmatrix} [\omega_v] & v_v \\ 0 & 0 \end{bmatrix}, \quad (5)$$

$[\omega_g]$ and $[\omega_v]$ represent the skew-symmetric matrix of ω_g , ω_v respectively. By extending Eq. (3) using Eq. (2), we achieve,

$$\begin{aligned} \begin{bmatrix} [\omega_g] & v_g \\ 0 & 0 \end{bmatrix} &= \begin{bmatrix} \mathbf{R} & \mathbf{p} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} [\omega_v] & v_v \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{R}^T & -\mathbf{R}^T \mathbf{p} \\ \mathbf{0} & 1 \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{R}[\omega_v]\mathbf{R}^T & -\mathbf{R}[\omega_v]\mathbf{R}^T \mathbf{p} + \mathbf{R}v_v \\ 0 & 0 \end{bmatrix} \\ &= \begin{bmatrix} [\mathbf{R}\omega_v] & [\mathbf{p}]\mathbf{R}\omega_v + \mathbf{R}v_v \\ 0 & 0 \end{bmatrix}, \end{aligned} \quad (6)$$

where $[\mathbf{p}]$ is the skew-symmetric representation of \mathbf{p} . From the above equation, we get

$$\begin{aligned} \omega_g &= \mathbf{R}\omega_v \\ v_g &= [\mathbf{p}]\mathbf{R}\omega_v + \mathbf{R}v_v, \end{aligned} \quad (7)$$

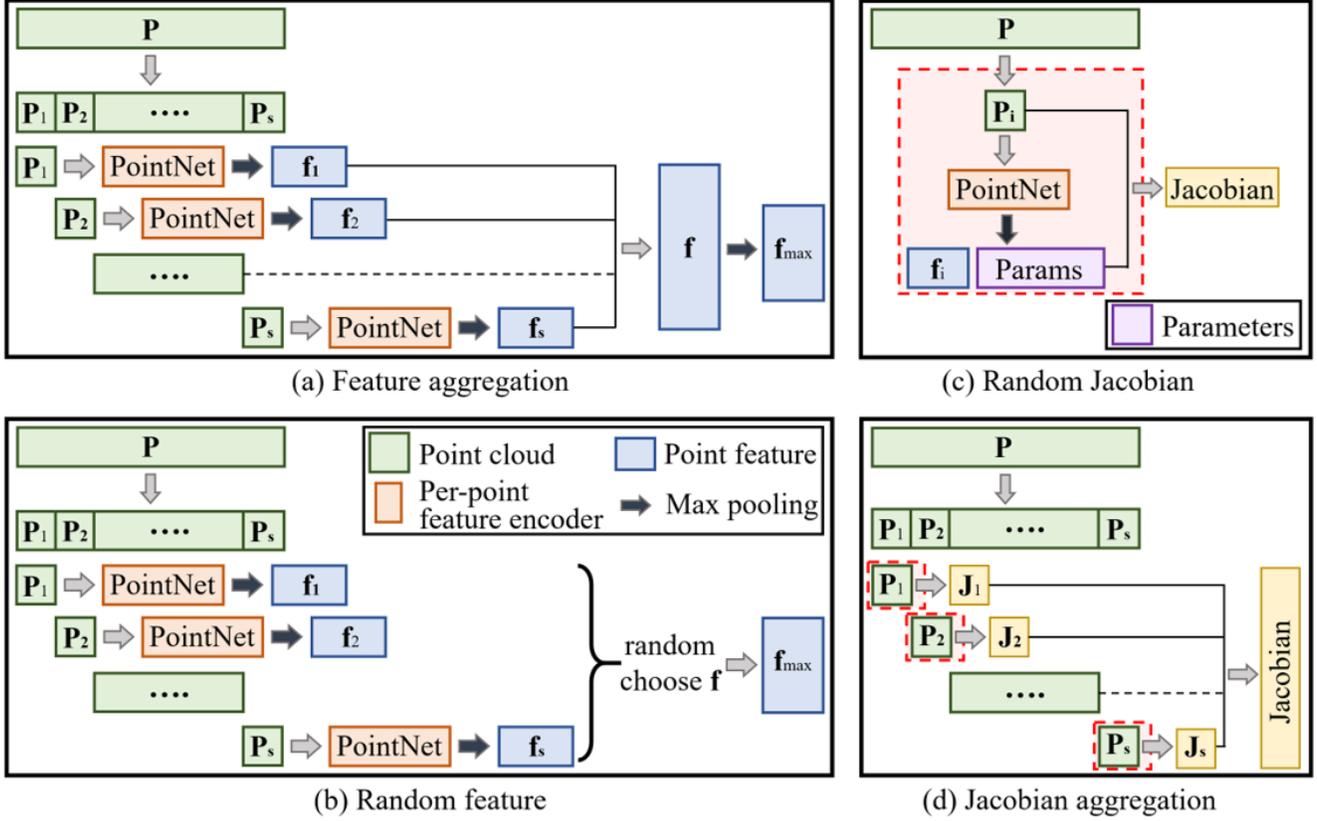


Figure 2. **Network design strategies.** Different combination of the network design strategies can be chosen to improve the accuracy and the efficiency of our method. For example, the combinations (a)(c), (a)(d), (b)(c), (b)(d) might be employed.

and in matrix format as

$$\begin{bmatrix} \omega_g \\ \mathbf{v}_g \end{bmatrix} = \begin{bmatrix} \mathbf{R} & 0 \\ [\mathbf{p}]\mathbf{R} & \mathbf{R} \end{bmatrix} \begin{bmatrix} \omega_v \\ \mathbf{v}_v \end{bmatrix}. \quad (8)$$

For simplicity, we set the rotation between the local voxel frame and the global frame to be an identity matrix, *i.e.* $\mathbf{R} = \mathbf{I}$. Therefore, Eq. (8) simplifies to

$$\begin{bmatrix} \omega_g \\ \mathbf{v}_g \end{bmatrix} = \begin{bmatrix} \mathbf{I} & 0 \\ [\mathbf{p}] & \mathbf{I} \end{bmatrix} \begin{bmatrix} \omega_v \\ \mathbf{v}_v \end{bmatrix}, \quad (9)$$

which equals to

$$\begin{bmatrix} \omega_v \\ \mathbf{v}_v \end{bmatrix} = \begin{bmatrix} \mathbf{I} & 0 \\ [\mathbf{p}] & \mathbf{I} \end{bmatrix}^{-1} \begin{bmatrix} \omega_g \\ \mathbf{v}_g \end{bmatrix}, \quad (10)$$

Then, the conditioned warp Jacobian can be expressed as

$$\frac{\partial \xi_{\mathbf{V}_m}}{\partial \xi^T} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & -\mathbf{p}_3 & \mathbf{p}_2 & 1 & 0 & 0 \\ \mathbf{p}_3 & 0 & -\mathbf{p}_1 & 0 & 1 & 0 \\ -\mathbf{p}_2 & \mathbf{p}_1 & 0 & 0 & 0 & 1 \end{bmatrix}^{-1}. \quad (11)$$

Note that the rank of $\frac{\partial \xi_{\mathbf{V}_m}}{\partial \xi^T}$ equals to 6, which means that the conditioned warp Jacobian is invertible. The intuition of Eq. (11) is that the local rotation of each voxel is identical to the global rotation, while the translation of each voxel is changed when converted to the global translation.

2.2. Implicit derivation

We can also use the implicit expression to compute the conditioned warp Jacobian as

$$\frac{\partial \xi_{\mathbf{V}_m}}{\partial \xi^T} = \left(\frac{\partial \text{vec}(\mathbf{P})}{\partial \xi_{\mathbf{V}_m}^T} \right)^{-1} \cdot \frac{\partial \text{vec}(\mathbf{P})}{\partial \xi^T}, \quad (12)$$

where \mathbf{P} represents a point cloud, $\text{vec}(\cdot)$ is the vectorization operator. Since $\partial \text{vec}(\mathbf{P})$ will be canceled out in Eq. (12), we can choose \mathbf{P} to be any point cloud. In practice, one can choose $\mathbf{P} = \mathbf{V}_m$, where \mathbf{V}_m is a point cloud contained in a local voxel.

In the main submission, we employed the explicit formulation due to its simplicity since we only need to construct one matrix instead of computing two different warp Jacobians.

3. Network Design Strategies

We noticed that the computational complexity of our analytical PointNetLK grows linearly with the number of points (*i.e.*, $\mathcal{O}(N)$), which makes the naive implementation problematic for training when dealing with large-scale point clouds. Here we propose simple design strategies to make our method computationally efficient.

Feature aggregation: A simple strategy is to randomly split the point cloud of size N into s segments, where each segment has N/s points. Each point cloud segment s can then be fed to the network to generate its feature vector \mathbf{f}_i . Then, max pooling can be employed to aggregate the collection of feature vectors \mathbf{f}_i into a global feature vector \mathbf{f} . Note that the feature aggregation strategy does not increase the time complexity. As shown in Fig. 2 (a), the long green box is the entire point cloud \mathbf{P} that contains N points. We split it into s segments where each segment \mathbf{P}_i is denoted as a small green box. Each segment is encoded through a per-point embedding PointNet (orange box) to get a feature \mathbf{f}_i which is depicted as a blue box. We then concatenate all the feature segments to get \mathbf{f} . We use max pooling to get our final feature vector \mathbf{f}_{\max} .

Random feature selection: We can also treat each segment of the point cloud as a small mini-batch. Without the feature aggregation, we consider each mini-batch as individual data that enables the network to learn better representations. Moreover, the network converges faster to a solution. (see Fig. 2 (b))

Random point selection for the Jacobian computation: Computing a large analytical Jacobian matrix for a large-scale point cloud is computationally expensive. A simple strategy is to compute the Jacobian for a set of randomly selected points. For example, we can randomly sample 10% of the points to compute the Jacobian and still have a high-fidelity registration as long as the important salient points are captured (see Fig. 2 (c)). The dimension of each matrix for the Jacobian computation shrinks intensively. Theoretically, our analytical PointNetLK can process the point cloud with a large number of points.

Point aggregation for the Jacobian computation: Randomly sampling points from a point cloud might result in loss of information when we have sparse point clouds and the selected points are not representative of the 3D shape. In this case, we can aggregate the Jacobian of each point cloud segment s to capture more important features in a point cloud. (shown in Fig. 2 (d))

4. Point Distance Loss

Inspired by [4], we can also use the Chamfer distance loss during training. Rather than directly computing the feature distance of the source and template point clouds, one can use a decoder to first reconstruct the point cloud from

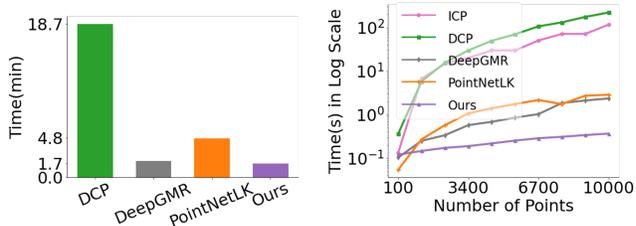


Figure 3. **Training and testing time.** Left figure shows the training time per epoch when training with the same GPU consumption. Our method takes about 2.5 minutes to train one epoch, while numerical PointNetLK takes 4.8 minutes and DCP takes 18.7 minutes. Right figure is the testing time of one point cloud on a single CPU. Purple line indicates that our method is fast during testing and is hardly affected by the number of points. As the number of points grows, the test time of correspondence-based methods grows quadratically.

the feature vector, then compute the Chamfer distance between the reconstructed source and target point clouds. The point distance loss is defined as

$$\mathcal{L}_P = \frac{1}{|\tilde{\mathbf{P}}_\tau|} \sum_{\mathbf{x} \in \tilde{\mathbf{P}}_\tau} \min_{\mathbf{y} \in \mathbf{P}_s} \|\mathbf{x} - \mathbf{y}\|_2^2 + \frac{1}{|\tilde{\mathbf{P}}_s|} \sum_{\mathbf{y} \in \tilde{\mathbf{P}}_s} \min_{\mathbf{x} \in \mathbf{P}_\tau} \|\mathbf{x} - \mathbf{y}\|_2^2, \quad (13)$$

where $\tilde{\mathbf{P}}$ is the reconstructed point cloud.

Furthermore, we can combine the loss functions as following: $\mathcal{L}_1 = \mathcal{L}_G + \mathcal{L}_\phi$ for supervised learning, where \mathcal{L}_G is the transformation loss and \mathcal{L}_ϕ is the feature loss; $\mathcal{L}_2 = \mathcal{L}_G + \mathcal{L}_P$ for semi-supervised learning; and $\mathcal{L}_3 = \mathcal{L}_P$ for unsupervised learning. We employ \mathcal{L}_1 for most of our experiments. The \mathcal{L}_2 and \mathcal{L}_3 are used for ablation studies. Please refer to the main submission for details on the \mathcal{L}_G and \mathcal{L}_ϕ losses.

5. Results

5.1. Efficiency

Fig. 3 demonstrates that our method is more computationally efficient than other methods during training and testing. We trained each network using 1,000 points and a single GPU. During testing, we varied the number of points from 100 to 10,000. Using a simplified PointNet with 3 layers and only 100 points for the Jacobian computation, our method is faster than the original PointNetLK. It also requires less space and time than other methods, especially when the number of points is large. With the number of points increasing, our method still maintains high efficiency. This suggests that our approach has the potential to efficiently cope with large number of points.

#	Numerical Jacobian	Analytical Jacobian	Random Jacobian	Aggregated Jacobian	Aggregated feature	Random feature				Rot. Error (degrees)		Trans. Error	
							\mathcal{L}_G	\mathcal{L}_ϕ	\mathcal{L}_P	RMSE	Median	RMSE	Median
1	✓		✓				✓	✓		8.1825	3.63e-6	0.0743	5.96e-8
2	✓		✓					✓		5.2323	2.47e-6	0.0580	5.96e-8
3		✓	✓		✓		✓	✓		5.5578	2.83e-6	0.0493	5.96e-8
4		✓	✓			✓	✓			3.3502	2.17e-6	0.0307	4.47e-8
5		✓	✓					✓		3.3234	2.18e-6	0.0380	4.47e-8
6		✓	✓			✓		✓		3.6901	2.12e-6	0.0382	3.73e-8
7		✓		✓	✓		✓	✓		6.0874	2.77e-6	0.0665	5.96e-8
8		✓		✓		✓	✓			5.0043	2.13e-6	0.0546	4.47e-8
9		✓		✓	✓			✓		4.4247	1.71e-6	0.0481	2.98e-8
10		✓		✓	✓		✓	✓		4.2186	1.91e-6	0.0457	2.98e-8

Table 1. **Ablation study.** Results on different network design strategies and loss functions. The analytical PointNetLK achieved higher fidelity than the original PointNetLK with numerical Jacobian, which highlights the advantage of our analytical Jacobian. Using random features improved the alignment results. Random Jacobian computation did not lower the registration accuracy. Replacing the feature difference loss \mathcal{L}_ϕ with a point distance loss \mathcal{L}_P did not drastically improve the registration performance.

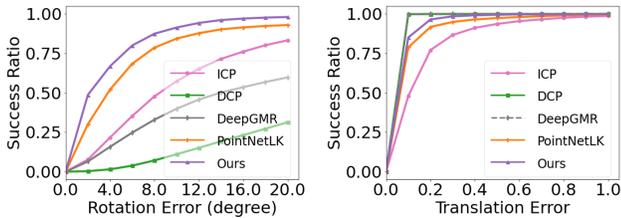


Figure 4. **Partial registration.** Left figure shows the success ratio of the rotation and the right figure is the translation success ratio. Our method is denoted as purple line, which has the highest success ratio for the rotation error metric, and relatively high success ratio for the translation error metric.

5.2. Partial data

In the partial data experiment, we followed similar settings as in [1]. The simulation process was to set a camera at the origin facing at direction (θ, ϕ) in the spherical coordinate, where ϕ is the polar angle, and θ is the azimuth angle. We sampled ϕ_T from a normal distribution $(0^\circ, 5^\circ)$ and θ_T from a normal distribution $(45^\circ, 5^\circ)$ for template, and ϕ_S from a normal distribution $(15^\circ, 5^\circ)$ and θ_S from a normal distribution $(30^\circ, 5^\circ)$ for source. Then, we moved the template/source point cloud along the vector $[r, \theta, \phi]^T$, where the distance r was set as 2. Next, we determined which points were visible to the camera. These visible points were the partial template/source point cloud used in our experiments.

As shown in Fig. 4, our method achieved higher accuracy than the other methods in rotation error metric. However, in translation, all the methods showed relatively high success ratio, while DCP and DeepGMR achieved better results.

5.3. Ablation study

Table. 1 shows quantitative results for different Jacobian computation strategies, different feature extraction strategies, and different loss functions. The first two rows are the

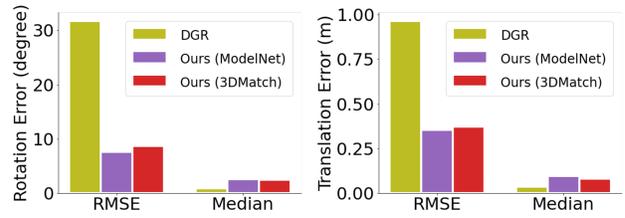


Figure 5. **Comparison with DGR [2].** The purple bars are the analytical PointNetLK trained on ModelNet40, and the red bars are the model trained on 3DMatch. Our method is robust to the initial misalignment with smaller root mean squared error (RMSE), while being competitive to DGR in median errors. This shows that our method has higher robustness than DGR when looking at all registration cases. Note that our model trained on synthetic ModelNet40 dataset still achieved high accuracy, which highlights the superior generalizability of our analytical PointNetLK.

canonical PointNetLK. Replacing feature difference loss \mathcal{L}_ϕ and transformation error loss \mathcal{L}_G with single point distance loss \mathcal{L}_P in row 2 improved the accuracy. The row 3-10 shows results of our analytical PointNetLK. It did not lose accuracy when aggregating feature for the entire point cloud (shown in rows 3, 7, 9, and 10). When using random feature rather than aggregated feature (rows 4, 5, 6, and 8), we improved the fidelity. Aggregating points for Jacobian computation (rows 7, 8, 9, and 10) slightly increased the fidelity but did not increase the accuracy, while computing Jacobian using random points (rows 3, 4, 5, and 6) performed slightly better in RMSE metric. Adopting point distance loss rather than feature loss (rows 5, 6, 9, and 10) would not likely improve the performance drastically. Note that with random feature or computing Jacobian using random points, our method is faster.

5.4. Comparison with DGR on 3DMatch

In our main submission, we showed how our method compares to the Deep Global Registration (DGR)

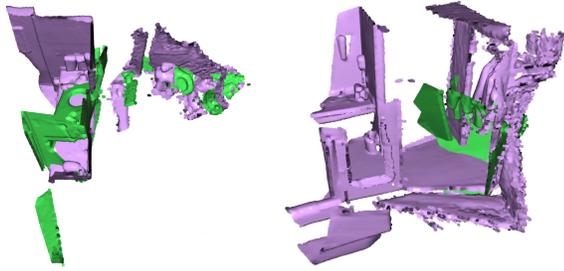


Figure 6. **Failure cases.** In the cases where there is a large space with a few features (left figure) or there is a small overlapped area (right figure), our method failed to achieve accurate registrations.

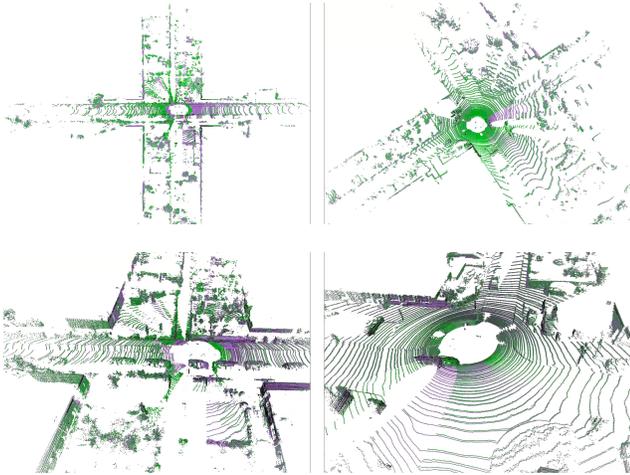


Figure 7. **Visual results of our method evaluated on the KITTI dataset.** The upper part shows two testing sequence results, and the lower part is the zoom-in figure of these two results.

method [2] in terms of the success ratio for the rotation and translation error metrics evaluated in the 3DMatch dataset. Here, we further show the RMSE and Median error metric results. In Fig. 5, our method, despite trained on the synthetic ModelNet40 dataset or the real-world 3DMatch dataset, achieved robust registration results for both rotation and translation. Our method is robust to the initial misalignment with smaller root mean squared error (RMSE), while being competitive to DGR in median errors. This shows that our method is more robust (smaller RMSE) than DGR, and will not fail with extreme large transformations. Note that the error metrics were calculated from all registration cases which clearly indicates the generalizability and robustness of our method.

5.5. Failure cases on 3DMatch

In Fig. 6, we show two typical failure cases of our method on the 3DMatch dataset. If the large-scale scenes have sparse features, our voxelized analytical PointNetLK will likely not capture robust features across the entire 3D space, thus leading to bad registrations. Another typical

Algorithm	Rot. Error (degrees)		Trans. Error (m)	
	RMSE ↓	Median ↓	RMSE ↓	Median ↓
Ours (no voxelization)	48.741	2.490	11.008	0.893
Ours (27 voxels, 37 points)	34.798	1.150	2.692	0.177
Ours (8 voxels, 125 points)	34.351	0.885	2.994	0.131
Ours (27 voxels, 148 points)	34.831	0.917	2.721	0.131
Ours (8 voxels, 500 points)	34.216	0.831	2.599	0.118
Ours (8 voxels, 1,000 points)	34.433	0.813	2.613	0.109

Table 2. **Performance on the KITTI dataset.** All of our methods were trained on the synthetic ModelNet40 dataset and tested on the KITTI dataset to further validate the generalizability of the analytical PointNetLK on a completely different dataset. Although the mean errors were large, which indicates several complete failure cases exist in the testing, the overall median errors were small, showing the high fidelity result of our methods. Note that our method achieved much better accuracy with the voxelization strategy, especially for translation. Moreover, more voxels with more points included in its interior improved the performance.

failure case is when the source point cloud and the template point cloud have a small overlapped area, and the point clouds contain a lot of outliers.

5.6. Results on the outdoor KITTI dataset

In order to show further generalizability of our method on more realistic settings, we demonstrate results on the KITTI odometry testing dataset [3] in Fig. 7 and Table 2.

References

- [1] Yasuhiro Aoki, Hunter Goforth, Rangaprasad Arun Srivatsan, and Simon Lucey. PointNetLK: Robust & efficient point cloud registration using PointNet. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7163–7172, 2019. 4
- [2] Christopher Choy, Wei Dong, and Vladlen Koltun. Deep global registration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2514–2523, 2020. 4, 5
- [3] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361. IEEE, 2012. 5
- [4] Xiaoshui Huang, Guofeng Mei, and Jian Zhang. Feature-metric registration: A fast semi-supervised approach for robust point cloud registration without correspondences. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11366–11374, 2020. 3