Stress-Testing Point Cloud Registration on Automotive LiDAR

Amnon Drory **Raja Giryes** **Shai Avidan**

Tel-Aviv University

Abstract

Rigid Point Cloud Registration (PCR) algorithms aim to estimate the 6-DOF relative motion between two point clouds, which is important in various fields, including autonomous driving. Recent years have seen a significant improvement in *global* PCR algorithms, *i.e*. algorithms that can handle a large relative motion. This has been demonstrated in various scenarios, including indoor scenes, but has only been minimally tested in the Automotive setting, where point clouds are produced by vehicle-mounted LiDAR sensors. In this work, we aim to answer questions that are important for automotive applications, including: which of the new algorithms is the most accurate, and which is fastest? How transferable are deep-learning approaches, *e.g*. what happens when you train a network with data from Boston, and run it in a vehicle in Singapore? How small can the overlap between point clouds be before the algorithms start to deteriorate? To what extent are the algorithms rotation invariant? Our results are at times surprising. When comparing robust parameter estimation methods for registration, we find that the fastest and most accurate is not one of the newest approaches. Instead, it is a modern variant of the well known RANSAC technique. We also suggest a new outlier filtering method, Grid-Prioritized Filtering (GPF), to further improve it. An additional contribution of this work is an algorithm for selecting challenging sets of frame-pairs from automotive LiDAR datasets. This enables meaningful benchmarking in the Automotive LiDAR setting, and can also improve training for learning algorithms.

We share our code and registration sets.[1]

1 Introduction

Rigid Point Cloud Registration (PCR) is an important task in many fields, including autonomous driving. Its goal is to estimate the relative motion between two point clouds, in 6 degrees of freedom (x,y,z translation; pitch, roll and yaw). Recent years have seen a significant improvement in *global* PCR algorithms, which can handle large relative motions. This is to a large extent thanks to deep-learned local features. A popular and successful approach to global PCR is to divide it into two main stages: *feature matching* and robust parameter estimation. In feature matching, a local descriptor (a.k.a feature) is calculated for each point in both clouds, and then each point in the source cloud is matched to the point in the target cloud that has the most similar descriptor to it. This results in a set of point pair matches, from which one could estimate a 6-DOF motion. However, the set of pairs contains many *outliers*, *i.e*. pairs whose relative motion does not follow the general motion between the two point clouds. This can be caused by independently moving objects in the scene, *e.g*. vehicles or pedestrians. It can also be caused by incorrectly matched pairs, due to various causes

[1] `https://github.com/AmnonDrory/LidarRegistration`

Machine Learning for Autonomous Driving Workshop at the 36th Conference on Neural Information Processing Systems (NeurIPS 2022), New Orleans, USA.

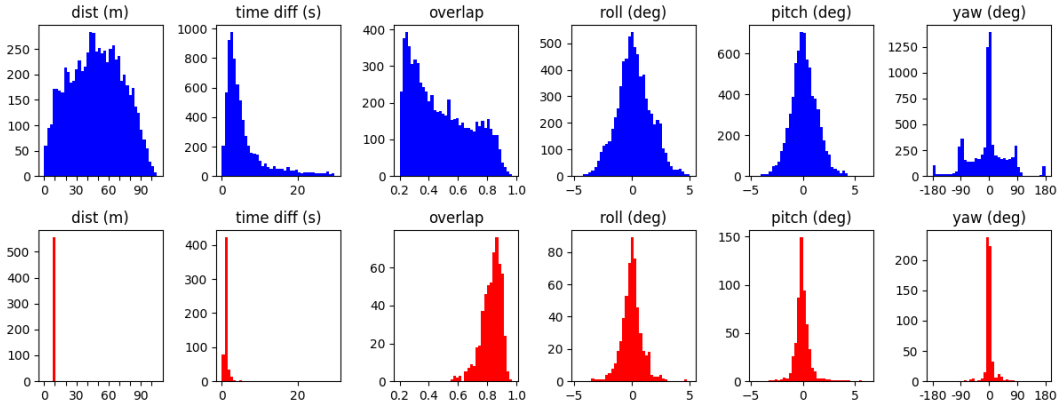

Figure 1: **Challenging registration sets.** The KITTI-10m registration set has been used extensively to evaluate PCR algorithms in the automotive LiDAR setting. However, it has recently become saturated: multiple algorithms achieve essentially perfect success. We suggest a new algorithm to select much more challenging registration sets. To compare the sets, we present here the distribution of test samples in *KITTI-10m* (bottom) and our suggested *Apollo-Southbay-Balanced* (top). We show the distribution of samples by (from left to right): distance between the pair of point clouds, time offset, overlap between scans, and rotation in three separate axes. *Apollo-Southbay-Balanced* includes a balanced representation of all the relative motions that are encountered in a real driving scenario. It is much more challenging, containing significantly larger rotations and smaller overlaps than *KITTI-10m*.

including occlusions, partial overlap between point clouds, limitations of the local descriptors, and more. To overcome outliers, it is necessary to use robust parameter estimation algorithms for the 6-DOF motion. These include explicit filtering of outliers, RANSAC, and many other approaches.

Recently, a dramatic improvement in the accuracy of PCR was achieved thanks to new, deep-learning based, features. Fully Convolutional Geometric Features (FCGF [10]) have achieved dramatically improved accuracy compared to classical hand-crafted features such as FPFH [26].

New robust parameter estimation algorithms for the 6-DOF motion have also been suggested recently. These include DGR [8] and PointDSC [2], which are based on deep learning, and also TEASER++ [31], which is not learning-based. These techniques have been validated mostly on indoor scenes, showing a considerable success. When combined with weaker features (*e.g.* FPFH), they show an ability to improve the results considerably. When combined with stronger features (*e.g.* FCGF) they achieve state-of-the-art results.

In the automotive field, point clouds are produced by vehicle mounted LiDAR sensors, and PCR is useful for estimating the ego-vehicular motion. There are several significant differences between automotive LiDAR PCR and other settings (such as single object scans and indoor): the scans are significantly larger, and often contain many distinct independent movers (vehicles, pedestrians, etc). In recent works, testing in the automotive LiDAR setting has been done almost entirely using the KITTI-10m registration set. This consists of pairs of frames from the KITTI Odometry dataset, that are separated by 10 meters. As a result, it appears that its test set consists almost entirely of high-overlap and small rotation situations. Thus, it fails to present a significant challenge to recent PCR algorithms, as many of them are able to achieve essentially perfect recall (up to a handful of failures at most, see Fig. 2).

In this work, we aim at thoroughly testing these new and promising PCR approaches in the automotive LiDAR setting. Our first goal is to achieve a meaningful ranking between these algorithms. We are interested not only in accuracy, but also in running time, which is critical for realistic applications. To get a meaningful comparison, we need a better benchmark than the ones previously used. We achieve this by using larger and more modern LiDAR datasets, and also by selecting a much more challenging set of frame-pairs (see Fig. 1). We developed an algorithm that selects a set of frame pairs that is balanced in the sense that it contains various relative motions (large and small rotations, short and large offsets, etc), a range of overlap ratios, and a fair sampling from each of the driving

sequences in a given dataset. Using these challenging sets, we get an insight into questions such as how low the overlap can get before registration results significantly deteriorate, to what extent the local descriptors are rotation invariant, and more.

We are also interested in understanding the importance and limitations of learning-based algorithms for PCR. Learned features have been shown to be superior to hand-crafted ones is various fields. However, learning brings with it the issue of transferability. When we train FCGF features on a dataset that was collected in a specific location, *e.g*. Boston, an important question is whether they will also be usable in other locations, *e.g*. Singapore? How much of the learning is in fact memorization of characteristics that are specific to a location?

Another question is with respect to the benefit of using deep-learning for robust parameter estimation. Some recent deep learning based algorithms, namely DGR and PointDSC, have shown a significant improvement for calculating the registration parameters when used with *weak* features. Yet, when used with strong features that are in themselves learned (*e.g*. FCGF), it is less clear whether an additional benefit is gained from these learning techniques over simpler approaches. For example, classical RANSAC has been shown to achieve comparable accuracy to the novel algorithms. Yet, prior works [8, 2, 31] claim that this requires significantly higher running time. Here, we suggest that this may not be a completely fair comparison: While basic RANSAC may be slow, many improvements have been suggested to it over the years for increasing its speed and accuracy. In this work, we evaluate combinations of several such improvements, and end up with a variant of RANSAC that is both more accurate *and* faster than the current state-of-the-art robust estimation algorithms. We show that the accuracy of our presented framework can be further improved with an outlier filtering method, which we propose and name Grid-Prioritized Filtering (GPF).

2 Related Work

Algorithms for rigid registration can roughly be divided into *local* and *global* ones. Local registration algorithms are based on the assumption that the motion is small. Global registration algorithms aim to handle any relative motion, but might be less accurate. Often their results are refined by running a local registration algorithm.

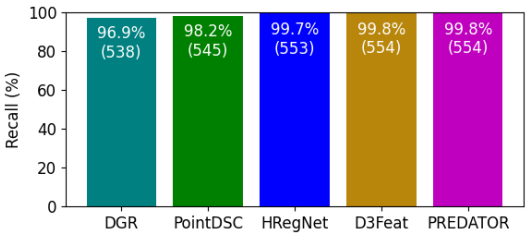

Figure 2: **Saturation of KITTI-10m**. KITTI-10m has been the standard benchmark for LiDAR registration for the last few years. It is essentially saturated: several recent algorithms have achieved almost perfect recall on it, failing only on a handful of the 555 point-cloud pairs in its test set. The values shown here were taken from the corresponding papers [8, 2, 20, 3, 18]. This saturation happens also in other existing datasets for LiDAR registration.

Local Registration: Iterative Closest Points (ICP) [5] is one of the earliest successful approaches to local point cloud registration, and it remains popular to this day. The ICP algorithm has been developed in various different directions [24]. Chen and Medioni [7] replaced the point-to-point loss function of ICP with a point-to-plane one, by using local normals. Segal *et al*. [28] presented the popular Generalized-ICP (G-ICP) [28] approach, which reformulated point-to-plane ICP in probabilistic terms and achieved improved accuracy. Rusinkiewicz recently suggested symmetric-ICP [25], which uses a surface-to-surface distance function that treats both point clouds symmetrically. It has been demonstrated to be superior to G-ICP in accuracy, and to have larger convergence basins. Drory *et al*. [13] presented Best-Buddies Registration, specifically BBR-F, which uses a set of mutual-nearest-neighbors in the registration to improve accuracy.

Global Registration: A successful strategy for global registration is to generate a set of point-matches based on local descriptors, and estimate a motion from these matches. A popular classical descriptor is FPFH [26] which uses histograms of gradients of neighboring points.

As in other fields, learned features have been shown to be superior to hand-crafted ones [1, 27, 33, 17, 34, 21, 29, 30]. Various such descriptor have been suggested, e.g. [20, 18, 10]. Fully Convolutional

Geometric Features (FCGF) [10] are based on sparse convolutions over a voxelized representation of the point cloud. The FCGF network is very fast, and produces dense features.

Robust optimization: The set of descriptor matches typically includes a significant fraction of *outliers*, which must be taken into consideration when estimating the relative motion. This can be done for example by using robust loss functions and algorithms, or by filtering the set of matches to remove outliers [32]. RANSAC [14] is a popular method, which works by repeatedly sampling a minimal set of point-matches, estimating a motion from the sample, and calculating its score by the fraction of matches that agree with this motion. This is repeated until a preset number of iterations is performed, or until early stopping occurs when the best-so-far motion has a fraction of inliers that is sufficient (relative to a confidence value supplied by the user [4]). This simple framework has been greatly enhanced over the years, improving RANSAC in both speed and accuracy.

PROSAC [11] performs a prioritized selection of candidate sets. It accepts the putative pairs sorted according to a quality measure, and orders the selection of sets so that sets with higher quality pairs are examined earlier. This simultaneously makes RANSAC faster and more accurate, by making it more likely that a good model is found early.

LO-RANSAC [12] adds a local-optimization step: when a best-so-far model is found, its inliers are used to find a better model, for example by performing RANSAC only on the inliers. Local optimization can be repeated several times, as long as the best-so-far model keeps improving significantly. Though the local-optimization step is expensive, it is only performed a few times over the run time of the RANSAC algorithm, and so its amortized time is small. The recently proposed GC-RANSAC [4] uses a Markov-Random Field formulation and solves it with Graph-Cuts to divide pairs into inliers and outliers.

Another important addition to RANSAC are early rejection methods, which can be applied quickly to reject a minimal set without going through the full scoring stage. We consider two such methods: Sequential Probability Ratio Test (SPRT) [23], a general domain method, and Edge-Length Consistency (ELC) [2], which is specific to PCR.

In addition to producing local descriptors, deep-learning has also been used for robust estimation. Deep Global Registration (DGR) [8] is based on training a second FCGF-like deep network for the task of recognizing outliers. PointDSC [2] too is based on a second network, but not to simply recognize outliers. Instead, it learns an embedding space where one can locate groups of mutually-consistent pairs, that can be used to generate candidate motions. PointDSC integrated ELC into the neural network, to encourage spatial consistency.

A novel approach that is not based on deep learning is TEASER, which is based on truncated least squares estimation and semi-definite relaxation. TEASER++ is a faster version that is based on Graduated Non-Convexity.

Dataset Generation. Fontana *et al*. [15] present a collection of datasets to be used as a benchmark for registration algorithms, and specify the method for the creation of these datasets. Unlike them, we focus specifically on LiDAR point cloud datasets, and registration sets that are challenging for global registration. We adopt their idea of achieving a balanced set of relative motions by random sampling. However, in their method a random motion is applied to an existing point cloud, thus creating a synthetic sample. Instead, we produce natural samples by selecting a pair of point clouds from a recorded sequence, so that their relative motion is as close to the randomly selected one as possible.

Huang *et al*. [18] present the 3DLoMatch set, that contains pairs of low-overlap scans from the 3DMatch [35] dataset. They define an overlap of between 10% and 30% as low. We set the minimum-overlap of our registration sets to 20%, which is in the same range. In line with their findings, our experiments show that low-overlap is a strong indicator for registration failure.

3 Balanced LiDAR Registration Sets

Popular registration benchmarks for the automotive LiDAR setting have become too easy for the newest registration algorithms (see Fig. 2). We believe the main cause for that are the simple heuristics used for selecting frame-pairs for registration: a constant offset in space or time, which is typically not very large (e.g. 10 meters, or 1 second).

How could we instead select a more interesting set of frame-pairs? A naive approach would be to enumerate all possible frame-pairs in each driving-sequence, and then select randomly from them. This approach has two problems: first, many frame-pairs have no overlap, making registration impossible. Second, and more importantly, for a large majority of frame-pairs, the relative motion between them is simple, e.g. "small offset, no rotation".

We suggest a different approach: sample uniformly from the space of motions. We think of the space of all relative motions as a 6-dimensional hyper-cube, whose axes are x-offset, y-offset, z-offset, roll, pitch and yaw. Different areas in this cube represent different *types* of motions: small-offset with large yaw, large-offset with small yaw but large pitch, etc. By sampling uniformly at random from this hyper-cube, we end up with a set of frame-pairs that is challenging and contains representatives of all the types of motion that appear in the LiDAR dataset.

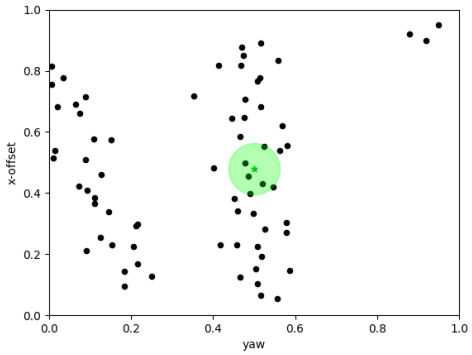

Figure 3: **Selection of Balanced Registration Set.** Toy example of our selection method, using a 2DOF motion model (instead of 6DOF). Each black point represents the relative motion between a *frame-pair*. The space of all motions is normalized into the unit square. Iteratively, we randomly sample a location (*green asterisk*), and select one of the frame-pairs that is close enough to this location (within *green circle*).

Generating a pool of candidates. In theory, every pair of point-clouds from the dataset could be considered as a candidate for the registration set. Yet, the total number of pairs is quadratic in the size of the dataset making this impractical. To generate a reasonably sized candidate pool, we take each kth frame in a sequence to be a source frame. For each source frame we find the set of frames whose overlap with it is above $min\_overlap$, and randomly choose the target frame from this set.

Random selection of samples. We wish to select uniformly at random from the space of all relative motions that appear in the candidate pool. We iteratively repeat the following procedure (demonstrated in Fig. 3): First, we normalize each axis of the 6D hyper-cube separately to the range [0,1], to overcome different ranges for different axes (x-offset, yaw, etc.). Then, we randomly generate a location in the unit hyper-cube. If our location is farther than a radius r from any candidate, we discard it and generate another. Otherwise, we consider the set of candidates within a radius r. They represent essentially the same type of motion, and we choose between them according to a second criterion: which driving sequence they come from. This allows us to encourage a fair representation for each driving sequence in the dataset, which is important since different sequences often include different challenges: highways vs. residential areas, daytime vs. nighttime etc.

We find it important to discard random locations that are farther than r from any candidate. Allowing such locations to select the candidate nearest to them would have distorted the distribution of samples that we select. For instance, candidates that lie next to a large empty region of the hyper-cube would have a much higher probability of being selected.

Balanced registration sets. Various Automotive LiDAR datasets are available, including KITTI-Odometry [16], NuScenes [6], Apollo-Southbay [22] and others. We use our algorithm to create three registration sets, that we use in our experiments. The sets are built over the Apollo-Southbay and NuScenes datasets. We divide NuScenes into two parts: Boston and Singapore. We name our registration sets *Apollo-Southbay-Balanced*, *NuScenes-Boston-Balanced* and *NuScenes-Singapore-Balanced*. We set $min\_overlap = 0.2$ and $r = 0.1$. Our sets are considerably larger than KITTI-10m (see Appendix A for size table). We believe this is beneficial in training, and also allows finer-grain comparison between algorithms in testing.

In Fig. 1 we compare the distribution of samples in *Apollo-Southbay-Balanced* to that in KITTI-10m. We show marginal distributions according to different parameters: time-offset, distance, overlap, roll, yaw and pitch. In all parameters, our set includes a wider range of values than KITTI-10m. This is especially evident for distance, which for KITTI-10m is by definition always approximately 10 meters,

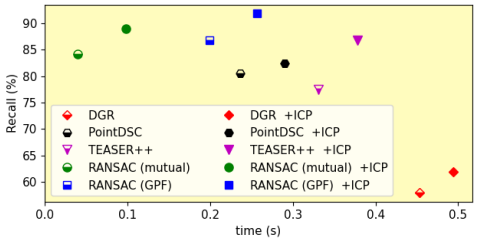

Figure 4: **Comparison of registration algorithms on a balanced LiDAR dataset**. We use *NuScenes-Boston-Balanced* to compare recent point-cloud registration algorithms. All algorithms use FCGF local-descriptors that were trained on this dataset. We show wall-time and recall, with and without ICP refinement. Advanced RANSAC is simultaneously *faster and more accurate* than all other algorithms. Its two versions differ in the pre-filtering method used; The faster one *(mutual)* uses mutual-nearest neighbors, and the more accurate one *(GPF)* uses our proposed Grid-Prioritized Filtering.

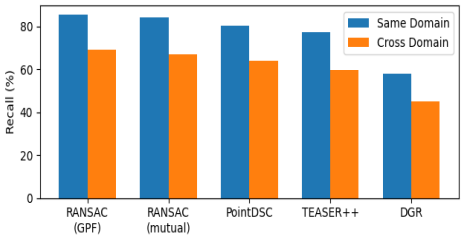

Figure 5: **The effects of cross-domain testing**. When FCGF features are trained using a training set which is substantially different from the test set, we see a drop in accuracy. Here, the test set is from *NuScenes-Boston-Balanced*, and the training set is either from *NuScenes-Boston-Balanced* (same-domain, blue), or instead from *Apollo-Southbay-Balanced* (cross-domain, orange). We see a drop in accuracy across all algorithms, of approximately 16 percentage points on average.

and in our set is a wide range, upto over 50 meters. KITTI-10m includes only high-overlap pairs, while our dataset contains a range, actually focusing on the harder, low-overlap cases. Regarding yaw, KITTI-10m includes only small rotations, while our dataset includes a wide range, up to 90 degree turns and even some complete U-turns. Our dataset also contains more samples with significant roll and pitch than KITTI-10m does.

4 Grid-Prioritized Filtering (GPF)

Pre-filtering the set of putative pair-matches, to reduce the fraction of outliers in it, is an important step in various PCR algorithms, including RANSAC and TEASER++. Various heuristic methods appear in the literature, including mutual-nearest neighbors (MNN, a.k.a reciprocity check), and ratio test [19], which can be based on distances in feature space or x-y-z space. In our work we consider registration with relatively low overlap, and we find that in this setting special care needs to be taken to make sure that after filtering, we are still left with points that are well spread spatially. To encourage this, we divide the point cloud into a grid (in the x-y dimension), and perform filtering separately in each grid cell. Special care is taken to balance the number of points remaining in each cell. We take sort the pairs in each cell according to two criteria: is/isn't MNN, and 1^{st} to 2^{nd} neighbor distance ratio. We look at distances in feature space, which makes more extensive usage of the deep-learned features than just ascertaining nearest-neighbor pairs. We call our filtering method Grid-Prioritized Filtering (GPF), and present its full details in Appendix B.

5 Experiments

In this section we present several experiments, comparing different registration algorithms on the proposed LiDAR registration sets. All methods use FCGF [10] deep-features trained on these sets, and differ in the robust estimation step. In some experiments the train-set and the test-set come from the same LiDAR dataset (same-domain), and in others from different datasets (cross-domain). This allows us to analyze the effect of cross-domain testing on deep feature accuracy. We compare the following algorithms: **Learned:** DGR [8], PointDSC [2], **algorithmic:** TEASER++ [31] and RANSAC. We tried various flavors of RANSAC (see Appendix D), and the best combination found includes:

1. Prioritized selection of candidates (PROSAC), using the same priority order used in GPF
2. Fast-rejection by edge-length consistency (ELC)

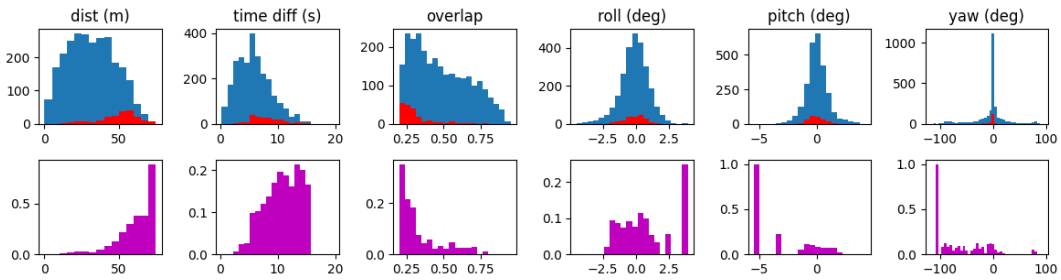

Figure 6: **Analysis of Failures.** We show the distribution of failed samples when running RANSAC (GPF) on the *NuScenes-Boston-Balanced* dataset, with ICP refinement (see Fig. 4). On the top row we show the distribution of successful registrations (blue) and failed ones (red), according to several parameters. On the bottom row, we show the ratio of failures for each bin in the corresponding top row histogram. Large distance and small overlap emerge as the most influential parameters for failure. Other parameters seem to have little influence, except in the most extreme cases.

3. Local-optimization step (LO-RANSAC), without graph-cuts

We compare two kinds of pre-filtering for RANSAC: mutual-nearest neighbors (MNN), and the novel GPF, with a 10×10 grid. TEASER++ also requires filtering, as it tends to get stuck indefinitely when receiving too many putative pair-matches as input. We use MNN for TEASER++ in all experiments, and add a second filtering with GPF when testing on *Apollo-Southbay-Balanced* (see ahead).

For each registration task we measure the rotation error (RE) and translation error (TE), defined as

$$\text{RE}(\hat{\mathbf{R}}) = \arccos \frac{\text{Tr}(\hat{\mathbf{R}}^{\mathbf{T}}\mathbf{R}^*) - 1}{2}, \tag{1}$$

$$\text{TE}(\hat{\mathbf{t}}) = \left\|\hat{\mathbf{t}} - \mathbf{t}^*\right\|_2. \tag{2}$$

where R^*, t^* is the ground-truth transformation. We follow [2] in defining a successful registration as one with RE$<$5 degrees and TE$<$0.6 meters (*i.e.*, twice the voxel-grid spacing, see ahead). As a measure for the accuracy of an algorithm we use *Recall*, which is the percentage of test samples for which registration succeeded. We report results before and after refinement, which is done with ICP (see Appendix C for experiments with some other refinement algorithms). Further implementation details can be found in Appendix A .

5.1 Stress-Testing LiDAR registration

In Fig. 4[2] we present the results of using the *NuScenes-Boston-Balanced* dataset to compare between DGR, PointDSC, TEASER++, and RANSAC with two pre-filtering algorithms: MNN (with max-iterations=1M, confidence=0.9995), and GPF(3.0) (with max-iterations=1M, and confidence=0.999). The fastest results are achieved by RANSAC with MNN filtering. The highest accuracy is achieved by RANSAC with GPF.

By analyzing the failures of a PCR algorithm we can achieve an insight into its limitations. In Fig. 6 we show the distribution of failures in the previous experiemnt (with RANSAC+GPF). Distribution is according to several measures: distance between the point clouds, overlap, time offset, and three axes of rotation. One interesting conclusion is that the algorithms are extremely rotation invariant - the extent of rotation in the initial motion does not seem to be correlated with failure rates, except perhaps in a very few, very extreme cases. What emerge as the most influential parameters for failure are large distance and small overlap. Below an overlap of 0.35 we start to see a measurable increase in the failure rate. However, even with an overlap rate as low as 0.2, most of the registration attempts still succeed.

In Tab. 1 we look at the setting of cross-domain testing. Here, all networks (FCGF, DGR and PointDSC) are trained on the *Apollo-Southbay-Balanced* dataset, but the testing is on *NuScenes-Boston-Balanced*. The ordering between algorithms remains the same as in the previous experiment, except here TEASER++ is faster than PointDSC. However, all recall values suffer a significant drop

[2]also see table in Appendix A

Table 1: Cross-Domain Evaluation

	Algo. only		with ICP	
	Recall	Time(s)	Recall	Time(s)
DGR	44.95%	0.418	48.07%	0.462
PointDSC	63.97%	0.234	66.78%	0.293
TEASER++	59.88%	0.146	71.99%	0.213
RANSAC (mutual)	66.94%	**0.107**	74.31%	**0.171**
RANSAC (GPF)	**69.14%**	0.113	**77.70%**	0.177

Table 2: All Registration Sets Cross-Domain

Test	Train	RANSAC (GPF)	RANSAC (mutual)	PointDSC	TEASER++
Apollo	Apollo	**98.97**	96.97	94.02	96.65
Apollo	Boston	**93.84**	93.25	88.53	92.62
Apollo	Singapore	**97.52**	94.86	93.54	95.16
Boston	Apollo	**77.70**	75.31	66.40	72.11
Boston	Boston	**91.13**	89.39	82.37	86.88
Boston	Singapore	**85.61**	80.79	75.39	79.63
Singapore	Apollo	**88.43**	87.92	79.01	86.69
Singapore	Boston	**91.47**	90.59	82.02	89.16
Singapore	Singapore	**94.60**	93.75	90.59	93.29

in the cross-domain case. Figure 5 visualizes this drop in accuracy for the case with ICP, showing a mean drop in recall of 16 percentage points. Cross-domain accuracies are significantly lower than the same-domain accuracies that we have seen in Fig. 4. We believe this shows that though FCGF features are quite transferable, some of their learning is location specific. In this experiment, we use GPF(2.0) with max-iterations=50K, and confidence=0.999. To allow clearer comparison, we use the same parameters also for the same-domain experiment in Fig. 5.

Table 2 presents a thorough test of our new datasets *Apollo-Southbay-Balanced* (Apollo), *NuScenes-Boston-Balanced* (Boston) and *NuScenes-Singapore-Balanced* (Singapore). In each of the 9 experiments, one dataset is used for training, and another for testing. We test four algorithms: PointDSC, TEASER++, RANSAC with MNN filtering, and RANSAC with GPF. ICP is used for refinement in all cases. The highest result in each *row* is in bold, the second underlined. Point clouds from the Apollo-Southbay dataset are approximately twice as large as those from NuScenes, and this ratio is maintained even after mutual-nearest neighbor filtering. As a result, TEASER++ tends to get stuck often ($\sim$15% of cases) when working on Apollo-Southbay point clouds. To overcome this, we use two mechanisms. First, a stricter filtering than usual for TEASER++: we first filter with MNN, and then filter with GPF, keeping a maximum of 2000 pairs. Second, we use a time-out of 10 seconds, after which registration is marked as failed. This happens very rarely (less than 0.1% of cases). The larger point-clouds in Apollo-Southbay also affect our settings for RANSAC+GPF. We use GPF(1.0) when testing on Apollo-Southbay, and GPF(2.0) when testing on NuScenes. All other settings for RANSAC are as in the previous experiment.

In all cases, we see that accuracy drops when testing cross-domain. In addition, we can see that *Apollo-Southbay-Balanced* is in a sense the simplest: it achieves the highest same-domain and cross-domain test results, but when networks are trained on it, they achieve the lowest cross-domain

accuracy. Training on the *NuScenes-Singapore-Balanced* dataset, on the other hand, leads to the highest cross domain accuracies. As far as algorithm comparison, RANSAC (GPF) is the most accurate, and RANSAC (mutual) the second except in one setting where TEASER++ is the second. Both RANSAC variants are also faster than the other algorithms in all cases (see Appendix A for running times).

6 Conclusion

We thoroughly test novel PCR algorithms on challenging benchmarks in the automotive LiDAR setting. We find that deep-learned features such as FCGF[3] suffer from partial reduction in accuracy when tested in a locale different from the one where they were trained.

We compare various robust parameter estimation algorithms, and find that the recent deep-learning based ones in fact do not achieve the best results. Instead, we find a variant of RANSAC which is both faster and more accurate than all other competitors. We also suggest an outlier filtering method, GPF, that further improves its accuracy. To improve current benchmarks, we have also introduced an algorithm for the selection of challenging frame-pairs from automotive LiDAR datasets. We believe it will be useful for future research, both for benchmarking PCR algorithms, and for training learned ones.

References

[1] Yasuhiro Aoki, Hunter Goforth, Rangaprasad Arun Srivatsan, and Simon Lucey. Pointnetlk: Robust & efficient point cloud registration using pointnet. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[2] Xuyang Bai, Zixin Luo, Lei Zhou, Hongkai Chen, Lei Li, Zeyu Hu, Hongbo Fu, and Chiew-Lan Tai. Pointdsc: Robust point cloud registration using deep spatial consistency. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15859–15869, June 2021.

[3] 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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[4] Daniel Barath and Jiří Matas. Graph-cut ransac. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[5] Paul J. Besl and Neil D. McKay. A method for registration of 3-d shapes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14(2):239–256, Feb. 1992.

[6] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. *arXiv preprint arXiv:1903.11027*, 2019.

[7] Yang Chen and Gérard Medioni. Object modelling by registration of multiple range images. *Image Vision Comput.*, 10(3):145–155, Apr. 1992.

[8] Christopher Choy, Wei Dong, and Vladlen Koltun. Deep global registration. In *CVPR*, 2020.

[9] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3075–3084, 2019.

[10] Christopher Choy, Jaesik Park, and Vladlen Koltun. Fully convolutional geometric features. In *ICCV*, 2019.

[11] O. Chum and J. Matas. Matching with prosac - progressive sample consensus. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 220–226, jun 2005.

[12] Ondrej Chum, Jiri Matas, and Josef Kittler. Locally optimized ransac. In *DAGM-Symposium*, volume 2781 of *Lecture Notes in Computer Science*, pages 236–243. Springer, 2003.

[13] Amnon Drory, Tal Shomer, Shai Avidan, and Raja Giryes. Best buddies registration for point clouds. In *Proceedings of the Asian Conference on Computer Vision (ACCV)*, November 2020.

[14] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of The ACM*, 1981.

[15] Simone Fontana, Daniele Cattaneo, Augusto L. Ballardini, Matteo Vaghi, and Domenico G. Sorrenti. A benchmark for point clouds registration algorithms. *Robotics and Autonomous Systems*, 140:103734, 2021.

[16] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[3] Though our work focuses specifically on FCGF, our methods are also applicable to other types of learned features [3, 18]

[17] A. Hertz, R. Hanocka, R. Giryes, and D. Cohen-Or. Pointgmm: A neural gmm network for point clouds. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12051–12060, 2020.

[18] Shengyu Huang, Zan Gojcic, Mikhail Usvyatsov, Andreas Wieser, and Konrad Schindler. Predator: Registration of 3d point clouds with low overlap. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4267–4276, June 2021.

[19] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, Nov. 2004.

[20] Fan Lu, Guang Chen, Yinlong Liu, Lijun Zhang, Sanqing Qu, Shu Liu, and Rongqi Gu. Hregnet: A hierarchical network for large-scale outdoor lidar point cloud registration. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 16014–16023, October 2021.

[21] 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. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct 2019.

[22] Weixin Lu, Yao Zhou, Guowei Wan, Shenhua Hou, and Shiyu Song. L3-net: Towards learning based lidar localization for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6389–6398, 2019.

[23] Jiri Matas and Ondrej Chum. Randomized RANSAC with sequential probability ratio test. In *10th IEEE International Conference on Computer Vision (ICCV 2005), 17-20 October 2005, Beijing, China*, pages 1727–1732. IEEE Computer Society, 2005.

[24] François Pomerleau, Francis Colas, and Roland Siegwart. A review of point cloud registration algorithms for mobile robotics. *now*, 2015.

[25] Szymon Rusinkiewicz. A symmetric objective function for ICP. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 38(4), July 2019.

[26] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast point feature histograms (fpfh) for 3d registration. In *ICRA*, 2009.

[27] Vinit Sarode, Xueqian Li, Hunter Goforth, Yasuhiro Aoki, Rangaprasad Arun Srivatsan, Simon Lucey, and Howie Choset. Pcrnet: Point cloud registration network using pointnet encoding. *ArXiv*, abs/1908.07906, 2019.

[28] Aleksandr Segal, Dirk Hähnel, and Sebastian Thrun. Generalized-icp. In Jeff Trinkle, Yoky Matsuoka, and José A. Castellanos, editors, *Robotics: Science and Systems*. The MIT Press, 2009.

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

[30] Yue Wang and Justin M. Solomon. Prnet: Self-supervised learning for partial-to-partial registration. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 8812–8824, 2019.

[31] Heng Yang, Jingnan Shi, and Luca Carlone. Teaser: Fast and certifiable point cloud registration. *IEEE Transactions on Robotics*, 37(2):314–333, 2021.

[32] Jiaqi Yang, Ke Xian, Yang Xiao, and Zhiguo Cao. Performance evaluation of 3d correspondence grouping algorithms. In *2017 International Conference on 3D Vision (3DV)*, pages 467–476, 2017.

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

[34] Wentao Yuan, Benjamin Eckart, Kihwan Kim, Varun Jampani, Dieter Fox, and Jan Kautz. Deepgmr: Learning latent gaussian mixture models for registration. In *ECCV*, 2020.

[35] Andy Zeng, Shuran Song, Matthias Nießner, Matthew Fisher, Jianxiong Xiao, and Thomas Funkhouser. 3dmatch: Learning local geometric descriptors from rgb-d reconstructions. In *CVPR*, 2017.

[36] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018.

Table A.1: Balanced registration set sizes

	Train	Validation	Test
KITTI-10m	1338	200	555
NuScenes-Boston-Balanced	4032	384	2592
NuScenes-Singapore-Balanced	4032	384	2592
Apollo-Southbay-Balanced	4032	288	7008

Table A.2: Evaluation of Registration Algorithms

	Algo. only		with ICP	
	Recall	Time(s)	Recall	Time(s)
DGR	57.91%	0.453	61.81%	0.494
PointDSC	80.56%	0.236	82.48%	0.290
TEASER++	77.43%	0.331	86.88%	0.378
RANSAC (mutual)	<u>84.14%</u>	**0.040**	<u>89.01%</u>	**0.099**
RANSAC (GPF)	**86.88%**	<u>0.199</u>	**91.90%**	<u>0.257</u>

Appendix A Additional Experiment Details

The number of samples in each set is shown in Table A.1.

Table A.2 presents the results of the same-domain experiment in tabular form (corresponds to Fig. 4 in main article). The right side of the table shows results with ICP refinement, the left side without. The best result in each column is in bold and the second best is underlined.

In Tab. A.3 we show the running times (in seconds) for the All-Set Cross Domain experiment (corresponds to Tab. 2 in main paper). Fastest in each row (always RANSAC mutual) is in bold, second fastest (always RANSAC+GPF) is underlined.

We mention in the paper that *Apollo-Southbay-Balanced* has larger point clouds than the other datasets we use, and that this is also true after mutual-nearest neighbor filtering. In Tab. A.4 we show the

Table A.3: Running Times for All Registration Sets Cross-Domain Experiment

Test	Train	RANSAC (GPF)	RANSAC (mutual)	PointDSC	TEASER++
Apollo	Apollo	<u>0.326</u>	**0.292**	0.691	0.781
Apollo	Boston	<u>0.336</u>	**0.330**	0.725	0.354
Apollo	Singapore	<u>0.346</u>	**0.317**	0.702	0.449
Boston	Apollo	<u>0.177</u>	**0.171**	0.451	0.277
Boston	Boston	<u>0.157</u>	**0.098**	0.432	0.486
Boston	Singapore	<u>0.177</u>	**0.124**	0.437	0.477
Singapore	Apollo	<u>0.228</u>	**0.202**	0.616	0.250
Singapore	Boston	<u>0.224</u>	**0.147**	0.608	0.258
Singapore	Singapore	<u>0.237</u>	**0.119**	0.589	0.846

Table A.4: Pair-Match Set Sizes

Test	Train	Initial	MNN-filtered
Apollo	Apollo	23520	2123
Apollo	Boston	23520	1717
Apollo	Singapore	23520	1830
Boston	Apollo	8091	766
Boston	Boston	8091	837
Boston	Singapore	8091	841
Singapore	Apollo	10335	1106
Singapore	Boston	10335	1104
Singapore	Singapore	10335	1198

number of putative pair-matches for different experiments, before and after mutual-nearest neighbor (MNN) filtering. The datasets are *Apollo-Southbay-Balanced* (Apollo), *NuScenes-Boston-Balanced* (Boston) and *NuScenes-Singapore-Balanced* (Singapore). The values shown are averaged over all samples in each dataset.

A.1 Implementation Details

Calculating FCGF features requires all points to lie on a grid. Thus, we start all registration algorithms by down-sampling with an 0.3 meter voxel-grid filter (following [10, 8, 2]). We continue by calculating FCGF features and finding nearest-neighbors in the feature space. When reporting running time we omit the time taken by this pre-processing.

Code[4]**:** for RANSAC we use the GC-RANSAC [4] code base, which is efficiently implemented and offers multiple options (PROSAC, local-optimization, etc.). We added an ELC implementation based on the one in open3d (version 0.13) [36]. We've also tried the open3d implementation of RANSAC (see appendix), which offers fewer options, and is somewhat slower, though still quite fast. We run the GC-RANSAC code with *distance_ratio=0.6* and *spatial_coherence_weight=0*, which effectively makes it LO-RANSAC and not GC-RANSAC. We also enable PROSAC and ELC. We set outlier filtering parameters for each experiment separately, to demonstrate RANSAC's ability to achieve both the fastest and most accurate results. For ICP we use open3D, with *threshold=0.6*.

For DGR, PointDSC and TEASER++ we use the official implementations, with slight modifications. We use our own implementation for training FCGF features.

The number of training epochs was selected according to preliminary tests (not shown), to a level where further improvement is very slow. The values are: 400 epochs for FCGF, 50 for PointDSC and 40 for DGR (whose training is considerably slower). We also changed the rotation augmentation scheme to make more sense in the automotive LiDAR setting: instead of general rotations in all axes, we augment with nearly-planar rotations, where yaw is in the range ± 180 degrees, but pitch and roll are only up to ± 5 degrees.

We use two machines for our experiments:

A GPU: 4x Titan X, CPU: 20-core 2.20GHz Xeon E5-2630
B GPU: GTX 980 Ti, CPU: 8-core 4.00GHz i7-6700K

Most of our tests are performed on machine A, using a single GPU. TEASER++ code is run on machine B, due to its code failing to work on machine A. To compare running time, we extrapolate TEASER++'s presumptive running time on machine A. To do so, we calculate a normalizing ratio by

[4]See Appendix F for links and license information.

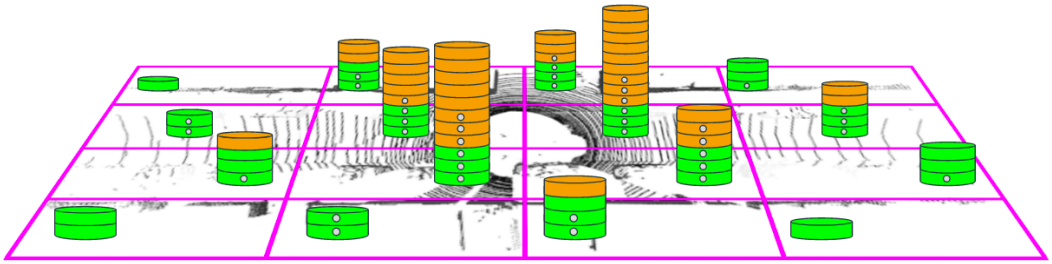

Figure B.1: **Grid-Prioritized Filtering (GPF).** GPF is a filtering algorithm used to select a subset of putative point-matches that are both high-quality and maximally spatially spread. This is achieved by dividing the source point-cloud into a grid of cells on the x-y plane, and selecting approximately the same number of matches from each cell. In the diagram, each match is represented by a disk, with those on the bottom, colored green, representing the ones that were selected by GPF. Within each cell, matches are ordered bottom-to-top by their estimated quality, based on analysis of the feature-space distance between the pair. Mutual nearest-neighbors (disks marked with small white circles) are selected first. Then, non-mutual. The secondary criterion for prioritizing is Eq. (3) (ratio between distance to 1st and 2nd nearest neighbor).

running RANSAC on both machines. In the appendix we analyze the differences in CPU and GPU running times across machines.

Appendix B Detailed Description of Grid-Prioritized Filtering (GPF)

We propose the Grid-Prioritized Filtering (GPF) method to explicitly ensure spatial spread in the selected pairs. As illustrated in Fig. B.1, GPF works by dividing the source point cloud into an $M \times M$ grid in the x-y plane. Then, ℓ matches are selected from each grid cell (or all matches if there are fewer than ℓ in the cell). The priority of pairs to select follows two criteria: First, matches that are MNNs are preferred. The secondary ordering criterion is the ratio S:

$$S(p) = \frac{d(p, q_2)}{d(p, q_1)}, \tag{3}$$

where P, Q are point clouds, $p \in P$, $q_1, q_2 \in Q$, q_1 is the nearest neighbor to p in Q , q_2 is the second-nearest, and $d()$ is the L_2 distance.

The number of pairs per cell, ℓ is determined by the total requested number, R. The simple calculation $\ell = R/M^2$ is only valid when all cells contain at-least ℓ pairs. Instead, we perform a quick binary search to find the value of ℓ that brings the overall selected number closest to R. R can be specified explicitly, but we believe that matching it to the properties of each point-cloud is preferable. To do so, we define it by:

$$R = \phi \cdot |\mathcal{N}|, \tag{4}$$

where $\mathcal{N}$ is the set of *mutual* nearest neighbors for each cloud, and ϕ is the user supplied *GPF factor*. We use notation like GPF(2.0) to refer to running GPF with $\phi = 2.0$.

Appendix C Local Registration (refinement)

Balanced datasets can also be used to compare local registration algorithms, such as ICP. Such algorithms take an initial coarse motion estimation, and refine it to achieve a high accuracy alignment. To use them with our balanced registration sets, we supply a standard set of *initial motions*, produced by performing RANSAC registration with FCGF features. These initial motions are generally close enough to the ground truth motion to allow local registration algorithms to succees. In Tab. C.1 we show the results of using the *Apollo-Southbay-Balanced* dataset, and comparing three local registration algorithms: ICP [5], symmetric-ICP [25], and BBR-F [13]. We use the official implementations of symmetric-ICP and BBR-F, and the open3d implementation of ICP. The point clouds are downsampled with a voxel-grid filter with a voxel size of 0.3 meters, and we set ICP's threshold to 0.6 meters (as we do in all experiments, following [8]). We report Recall, as well as

Table C.1: Refinement Experiment

Algorithm	Recall	TE (cm)			RE (deg)		
		mean	50%	95%	mean	50%	95%
ICP	98.99%	80.65	11.76	30.29	0.37	0.13	0.33
BBR-F	96.33%	86.98	15.10	52.84	0.47	0.19	0.66
sym-ICP	67.74%	548.85	17.68	3544.49	2.31	0.22	10.66

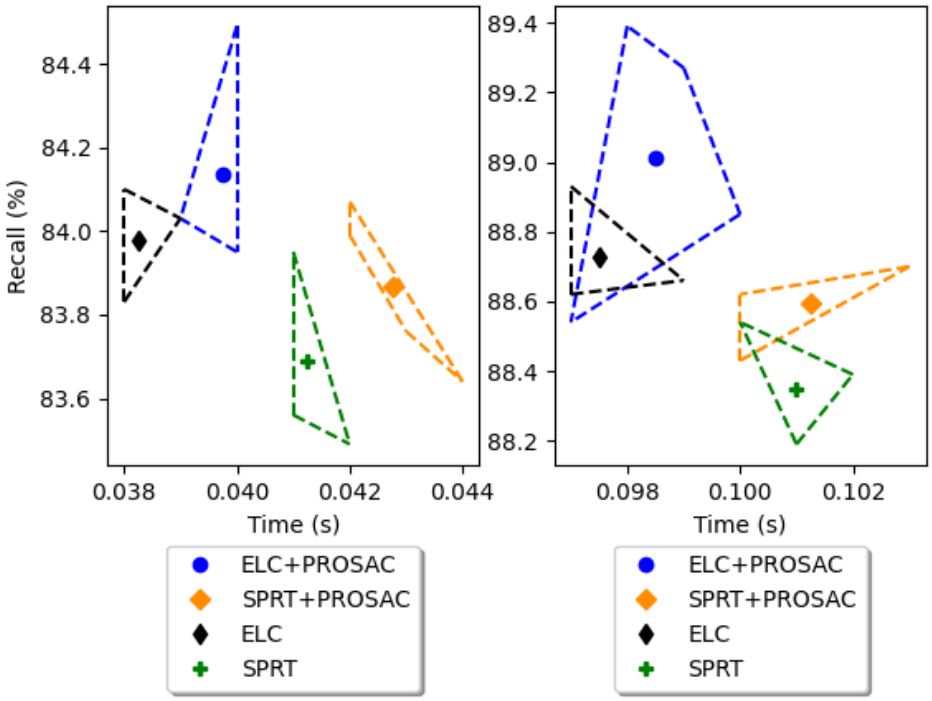

Figure D.1: **RANSAC Ablation: PROSAC and ELC/SPRT**. We show the accuracy and running time of different variants of RANSAC, with ICP (right) and without (left). For each setting, we repeat the run 4 times and show the spread of results by a polygon (the convex hull). We also show their mean. The best results are when we use both PROSAC and ELC.

translation error (TE) and rotation error (RE). We report mean, median and 95th percentile of TE and RE, and these statistics are taken over *all* test samples. The results show that ICP is more accurate than BBR-F, and both are considerably more accurate than symmetric-ICP. This differs from previous experiments in [13] that used a subset of KITTI. We believe the central factor is overlap between point-clouds: small overlap is common in our sets but not in KITTI. ICP explicitly filters point pairs whose distance is above a threshold, and BBR-F uses spatial mutual-nearest neighbors. These elements apparently gives them an edge over Symmteric-ICP in this setting.

Appendix D Ablation Studies

D.1 RANSAC

The version of RANSAC that we use in our experiments includes several improvements over classical RANSAC:

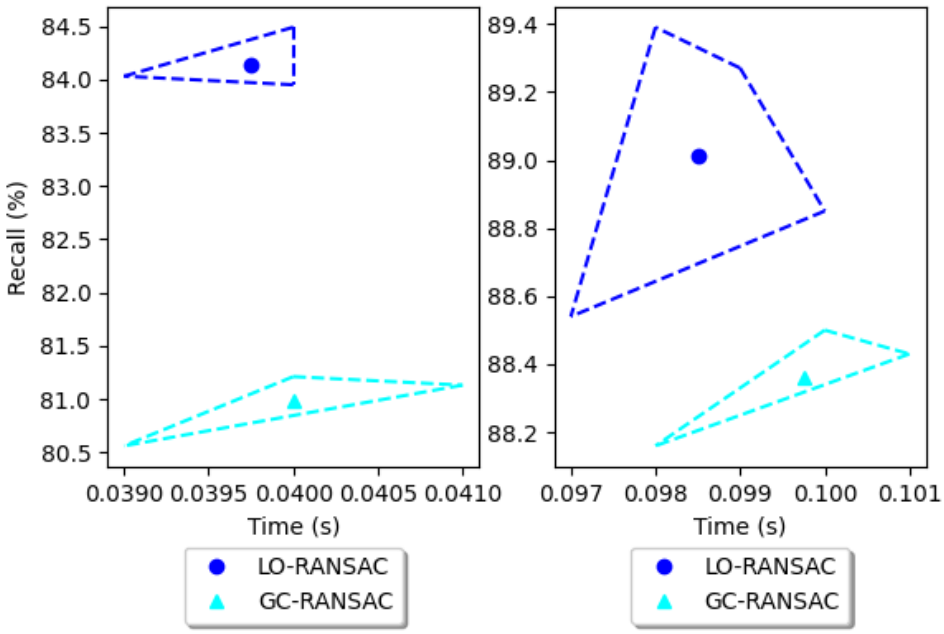

Figure D.2: **RANSAC Ablation: Local-Optimization**. Using the same visualization as Fig. D.1, we show LO-RANSAC is superior to GC-RANSAC in our setting.

1. Prioritized selection of candidate sets (PROSAC).
2. Quick rejection of candidate sets (with ELC).
3. Local-Optimization step (LO-RANSAC).

We perform ablation studies to show the importance of each element. We both train and test on *NuScenes-Boston-Balanced*, and use the same settings as in the experiment shown in Tab. 2 of the main paper, for the nearest-neighbor filtering case. All variants of RANSAC tested in this section are both faster and more accurate than the other algorithms we consider in our paper: TEASER++, PointDSC and DGR. The results of our first experiment are shown in Fig. D.1. We compare PROSAC to random selection of candidate sets, and in the quick rejection step, we compare ELC to SPRT. To show variance, we repeat each experiment 4 times, and plot both the mean and the convex hull of the 4 results. The results demonstrate that adding PROSAC improve accuracy but also adds to running time, and that replacing SPRT with ELC improves both accuracy and running time.

In Fig. D.2 we show a comparison of LO-RANSAC to GC-RANSAC. In both cases we use PROSAC and ELC, and the only difference is the parameter *spatial_coherence_weight*. To run LO-RANSAC we set it to 0. To run GC-RANSAC, we set it to its default value, 0.975. LO-RANSAC achieves higher recall than GC-RANSAC in our setting. We also tested other values of the parameter (not shown), and the best accuracy was achieved with 0 (i.e. LO-RANSAC).

In Fig. D.3 we compare the open3d implementation of RANSAC to the GC-RANSAC implementation which we use for most experiments (we refer to it as *GC-code*). The open3d implementation includes ELC, but does not include local-optimization and PROSAC. For the fairest comparison, we run the GC-code in a "compatible" setting, also using ELC but no local-optimization and no PROSAC. For reference, we also run the GC-code with our default setting (ELC, PROSAC and LO-RANSAC). Open3D is considerably slower than either GC-code setting. It is less accurate than our default setting of GC-code, but interestingly more accurate than the "compatible" setting. Possibly, this is due to differences in the implementation of early stopping. Open3d RANSAC is both faster and more accurate than all other algorithms we tested, (compare Fig. D.3 here to Fig. 6 in main paper).

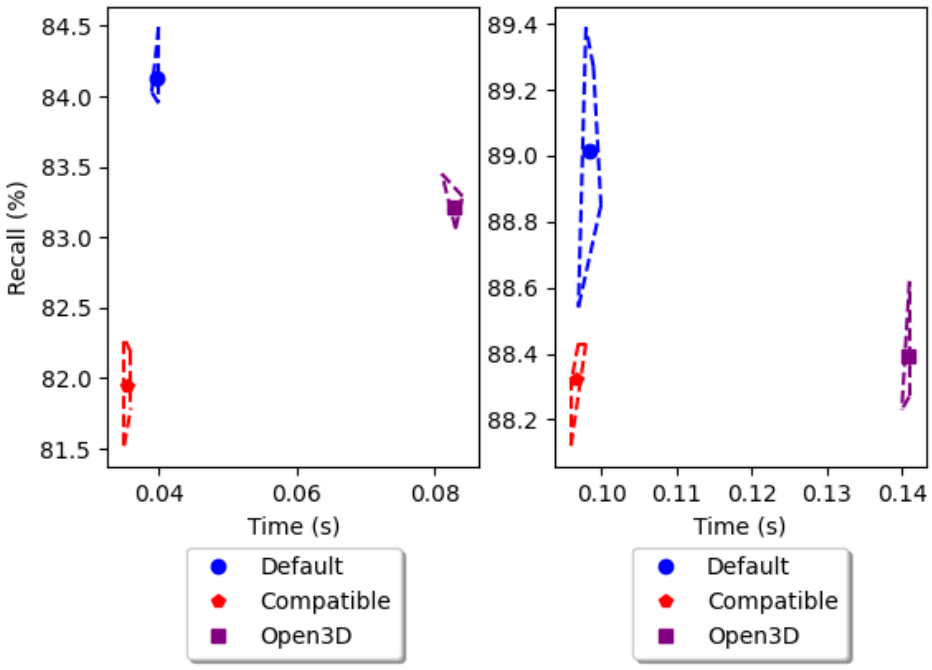

Figure D.3: **RANSAC code bases: Open-3D vs. GC-RANSAC**. We compare the open3d implementation of RANSAC (with ELC), to the GC-RANSAC implementation in two settings: "compatible" which is as similar as possible to open3d, and "default" which is what we use in most of our experiments. The open3d implementation is slower than the GC-RANSAC one. It also does not offer PROSAC and LO-RANSAC, causing it to be less accurate. However, it is still faster and more accurate than all other algorithm we tested (TEASER++, PointDSC, DGR).

D.2 GPF

In Fig. D.4 we demonstrate the effect of the number of iterations and of the GPF parameter when running RANSAC+GPF. We can see that when adding iterations, running time always increases, but accuracy reaches saturation and plateaus at some point. Increasing the GPF parameter ϕ, which corresponds to keeping a larger set of point-pairs, leads to an increase in both running time and in accuracy. However, the increase in accuracy does become considerably slower as we advance the parameter above 3.0. In our main experiments we used the parameter values of 1.0, 2.0 and 3.0.

Appendix E GPU and CPU Running Time on Different Machines.

Some of the registration algorithms that we compare rely mostly on GPU for processing (PointDSC, DGR), while others mostly use the CPU (TEASER++, RANSAC). Therefore, a comparison of running times between these algorithms depends on the specific machine being used. We demonstrate this in Tab. E.1, by running the same experiment on two machines. The machines that we use are:

A GPU: 4x Titan X, CPU: 20-core 2.20GHz Xeon E5-2630
B GPU: GTX 980 Ti, CPU: 8-core 4.00GHz i7-6700K

On either machine, we use only one GPU for testing. The experiment consists of testing PointDSC and RANSAC on the *NuScenes-Boston-Balanced* dataset (training was also performed on the same dataset). We report the running times on both machines. The ratio between the running times of PointDSC and RANSAC is different between the machines, reflecting the different mixes of CPU

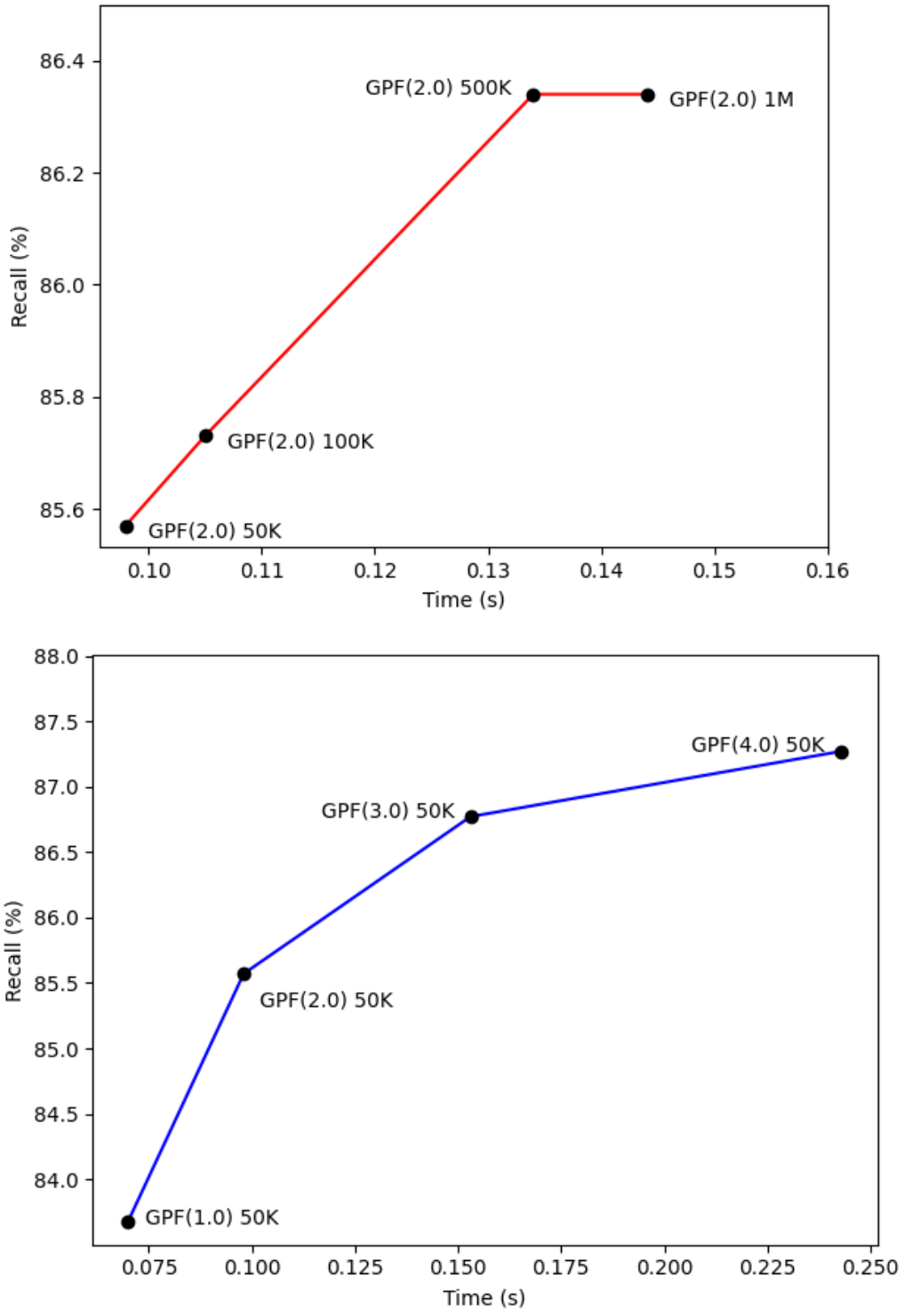

Figure D.4: **GPF Ablation**. We show the effects of different values of *max-iteration* (top) and of ϕ (bottom). Increasing max-iterations improves accuracy only up to a point, after which accuracy plateaus while running time increases. Increasing ϕ improves accuracy and increases running time, and the plateau phenomenon is much less pronounced.

Table E.1: Running Times Comparison on Two Machines

Algorithm	Main Resource	Machine A Time (s)	Machine B Time (s)
PointDSC	GPU	0.236	0.330
RANSAC	CPU	0.109	0.135
Ratio PointDSC/RANSAC		2.44	2.17

and GPU capabilities in each machine. For this experiment, we used the open3D implementation of RANSAC.

Appendix F Code Bases

In our work we make use the following code bases: **FCGF [10]**: `https://github.com/chrischoy/FCGF` (MIT License)

DGR [8]: `https://github.com/chrischoy/DeepGlobalRegistration` (MIT License)

Minkowski Engine [9]: `https://github.com/NVIDIA/MinkowskiEngine` (MIT License)

PointDSC [2]: `https://github.com/XuyangBai/PointDSC`

TEASER++ [31]: `https://github.com/MIT-SPARK/TEASER-plusplus` (MIT License)

Open3d [36]: `https://github.com/isl-org/Open3D` (MIT License)

GC-RANSAC [4]: `https://github.com/danini/graph-cut-ransac` (new BSD License)

Symmetric-ICP [25]: `https://gfx.cs.princeton.edu/proj/trimesh2/` (GPL Version 2 License)

Best-Buddies Registration [13]: `https://github.com/AmnonDrory/BestBuddiesRegistration`