DeepPRO: Deep Partial Point Cloud Registration of Objects Supplementary Material

Donghoon Lee Onur C. Hamsici Steven Feng Prachee Sharma Thorsten Gernoth
Apple
{donghoonlee,ohamsici,shuo_feng,prachee_sharma,tgernoth}@apple.com

1. Implementation Details

Network Architecture. Figure 1 shows the network architecture of DeepPRO. For EdgeConv layers, we search 20 neighbor points to define edges. We refer [14] for more details of how EdgeConv works. All blocks except the last blocks of the conditional point cloud generation network and transform estimation network, we use Conv-Batch Norm-ReLU layers. For the last block of the conditional point cloud generation network, we apply the tangent hyperbolic activation after the convolutional layer. For the quaternion, we use Sigmoid as the activation function of the first element without the batch normalization layer. Other elements are normalized by ℓ_2-norm after the convolutional layer.

Linemod Dataset Experiments. Given eleven objects in the Linemod dataset, we conduct leave-one-object-out training and test on the unseen object during training. We use AdamW optimizer [10] with default parameters in PyTorch, i.e., initial learning rate of 0.001, weight decay of 0.01, betas as (0.9,0.999), and epsilon as 1e-8. The learning rate is decreased by half at 1,500, 2,000, 2,500, 3,000, and 3,500 epochs. The network is trained until 4,000 epochs with batch size of 64 per GPU. The input point cloud has $N = 512$ points throughout the paper. The point cloud is rotated at most 5° in the random rotation layer and embedded into $d = 512$ dimensional feature space by the encoder. We train our algorithm on Intel(R) Xeon(R) Gold 6148 CPU and NVIDIA V100 graphics card. By using eight GPUs, it takes approximately a single day to train our algorithm. Across GPUs, we use synchronized batch normalization. For the object mask, we use ground truth labels for main experiments and report additional results with mask predicted by [2] in the ablation study.

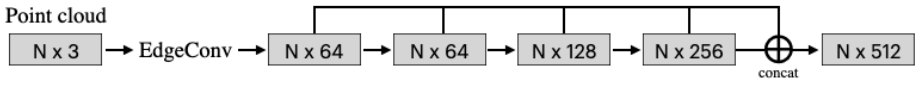
PRO1k Dataset Experiments. Among 1,000 objects in the dataset, there are objects that have challenging properties for point cloud estimation. For example, if the object has transparent or reflective part, it is difficult to obtain reliable point cloud since depth sensors like Kinect do not work well on those regions. Thin objects are also challenging as thin structures are often ignored by the depth sensor. We set these objects as our future work and focus on objects that does not have such challenging issues. To this end, we use 291 sequences for training and 7 representative sequences for evaluation.

We use distributed training with 240 GPUs to efficiently consume the dataset. To stabilize the training, we adopt state-of-the-art techniques such as warmup training, learning rate rescaling [8] and LAMB optimizer [18]. We have five warmup epochs and initial learning rate as 4e-6 which is linearly rescaled to about 0.001 based on the number of GPUs. The learning rate is decreased to half at 200, 400, 600, 800 epochs where 1,000 is the total number of epochs. Parameters for the LAMB optimizer are similar to the AdamW optimizer in the Linemod dataset experiment. Synchronized batch normalization is used. We detect object's 3D bounding box based on CGAL library (https://github.com/CGAL) and use points inside the box.

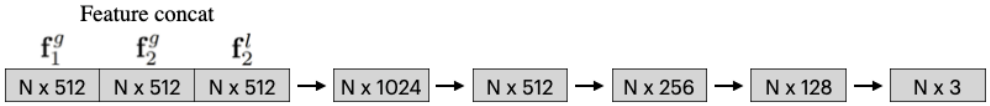
ModelNet40 Dataset Experiments. We follow the experimental settings and data preprocessings described in [13] to train DeepPRO on the ModelNet40 dataset. We first randomly sample 1024 points and simulate partial scan by placing a point in space and gather 768 nearest neighbors. Then, we add point-wise noise sampled from Gaussian $\mathcal{N}(0, 0.01)$ and clipped to $[-0.05, 0.05]$. Given the synthetic partial point cloud, we generate training pair by randomly rotating each axis $[0^\circ, 45^\circ]$ and translate $[-0.5, 0.5]$. Overall 12,311 CAD models are split to 9,843 train and 2,468 test data. For more details, we refer [13]. Other parameters are the same as the Linemod data experiments.

2. Comparison to Existing Methods

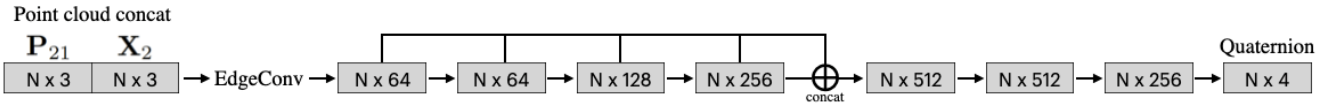
We provide webpage links to the code, describe how we run the code, and discuss their results. Before we evaluate different methods on the Linemod dataset, we first validate that the code works properly by comparing test outputs to the reported accuracy in each method's published paper on



(a) DGCNN encoder



(b) Conditional point cloud generation network



(c) Transformation estimation network

Figure 1: Network architectures of DeepPRO. It shows a path to predict $\mathbf{R}_{21}$ and $\mathbf{t}_{21}$. For the opposite path, the concatenated features in (b) are $\mathbf{f}_2^g$, $\mathbf{f}_1^g$ and $\mathbf{f}_1^l$ and the concatenated point clouds in (c) are $\mathbf{P}_{12}$ and $\mathbf{X}_1$.

their dataset. For learning-based approaches, we use the same leave-one-object out train and test scheme as discussed in the main paper.

DCP [12] https://github.com/WangYueFt/dcp. In our experiments, the training fails to converge on real objects. It is because DCP assumes that all points in one point cloud have a correspondence in the other point cloud. However, for real objects, this assumption does not hold due to partially observable point clouds. As the training is not converged, we use the pretrained model on the synthetic ModelNet40 dataset to report results. Due to the assumption that does not hold for real observations, the results are inaccurate.

D3Feat [3] https://github.com/XuyangBai/D3Feat. It is a recent keypoint based approach which shows state-of-the-art results for registering outdoor-scale [7] and indoor-scale [19] point clouds. One notable experiment in their paper is that they show strong generalization ability on the outdoor dataset [11] based on the model trained on the indoor dataset [19]. We focus on whether it can be generalized to the object-scale point cloud as well. To evaluate the algorithm, we follow author's suggestions to tune the network inference parameters such as the scale of the kernel points for objects. However, results show that the pretrained model on the indoor dataset [19] does not generalize well on small objects. It demonstrates that keypoint based point cloud registration approaches developed on the outdoor or indoor databases might suffer from less salient geometric features of the partial and noisy point cloud of objects.

PointNetLK [1] https://github.com/hmgoforth/PointNetLK. We fine-tune the algorithm on the Linemod dataset based on the pretrained weights on the ModelNet40 dataset. As the method relies on a single global feature vector to describe the point cloud, it is less accurate on the partial and noisy point clouds.

PRNet [13] https://github.com/WangYueFt/prnet. It is a keypoint-based approach for object-scale point clouds studied on the synthetic data. In our experiments on real objects, however, it frequently crashes when we train the network from scratch or finetune it using a pretrained network on ModelNet40. Therefore, we report results of PRNet trained on the ModelNet40 dataset and tested on the Linemod dataset. During training, PRNet mimics real observations by augmenting synthetic point. Results show that it is difficult to enclose the gap between real and synthetic point clouds due to complex self-occlusion and irregular noises in real data.

RPM-Net [17] https://github.com/yewzijian/RPMNet. We train RPM-Net on the Linemod dataset using the author's code. For each input point cloud, point normals are computed using Open3D library each time during training. We use the same settings for the model-related hyper parameters as the authors.

DGR [5] https://github.com/chrischoy/DeepGlobalRegistration. DGR is a keypoint-based method which shows state-of-the-art results on outdoor and indoor scenes. Using the author's code, we train DGR on the Linemod dataset with a smaller voxel size to match with the scale of the object point cloud while other parameters are kept the same. We try three different voxel sizes $(0.1\,\mathrm{cm}, 0.5\,\mathrm{cm}, 1\,\mathrm{cm})$ and report the best result with $0.5\,\mathrm{cm}$ case. DGR shows good

results among the existing learning-based methods while it is less accurate than Go-ICP on the Linemod dataset. It is interesting to note that DGR outperforms Go-ICP on the indoor dataset in the DGR paper. It again shows that keypoint based approach might not be optimal for object-scale point cloud.

GMM-based registration [9] https://github.com/bing-jian/gmmreg-python. It represents point set as a Gaussian mixture model and the alignment task is formulated as minimizing the statistical discrepancy between the Gaussian mixture models. For input point cloud, it uniformly samples 500 to 800 points depending on the object size and distance. We use default parameters provided the shared code.

ICP [4] http://www.open3d.org/docs/release/index.html. We use Open3D library for evaluating ICP algorithm. We try both point-to-point and point-to-plane ICP methods and report point-to-plane accuracy as it shows better results. To find parameters for ICP, we did grid search for two parameters. We consider $[0.1\,\mathrm{cm}, 1\,\mathrm{cm}, 10\,\mathrm{cm}]$ for the voxel size and $[1\,\mathrm{cm}, 5\,\mathrm{cm}, 10\,\mathrm{cm}]$ for the maximum correspondence distance threshold. To estimate the normal plane of the point cloud, the library finds adjacent points. We use the search radius as two times bigger than the voxel size and search for up to 30 neighbors. We observe that ICP has low error in general, however, it often predicts a transform with large errors due to the noisy point cloud.

FGR [20] http://www.open3d.org/docs/release/index.html. FGR is also implemented in the Open3D library. We use the same voxel size search grid as in the ICP experiments. FGR shows similar results as ICP on the Linemod dataset.

Go-ICP [16] https://github.com/aalavandhaann/go-icp_cython. As suggested by authors, we first normalize input point clouds into $[-1, 1]^3$ using the same scaling factor. Then, we test three different trimming ratio of 0%, 1%, and 10%. Other parameters such as the convergence threshold and the number of discrete nodes are kept as default setting, i.e., 0.001 and 300, respectively. Go-ICP can be considered as a meta algorithm since it runs ICP multiple times to find the better solution. Therefore it is much slower than ICP and difficult to optimize for real-time applications.

TEASER++ [15] https://github.com/MIT-SPARK/TEASER-plusplus. This method requires initial correspondence between keypoints. As suggested in the offical code, we use FPFH feature to obtain the initial correspondences. As the original setting of this code is optimized for indoor scenes, we get poor results on the object-scale point clouds. We further tweak the parameters to find the best setting. We tried with and without the voxel downsampling before extracting FPFH features, changed the number of neighboring points to calculate FPFH features, and perform grid search on TEASER++ solver parameters such as graduated non-convexity and noise bound.

JRMPC [6] https://team.inria.fr/perception/research/jrmpc/. We used the Matlab code provided by authors. The original code works for joint alignment of multiple point sets with respect to a reference point cloud. To utilize this code in our setup, we set the reference point cloud as one of the two input point clouds and obtain the view transformation between the reference point cloud and each of the point cloud inputs using JRMPC. The relative pose between input point clouds are then obtained by combining the estimated view transformations. We also tested to tune the parameters and the default values provided the best results.

3. Additional Experiments

Training Stability. We evaluate the stability of the training process by training the network from scratch three times with different random seeds. Then, we measure the standard deviation of results at the final epoch. Across different objects, the network shows stable results, i.e., $0.07°$, $0.08\,\mathrm{cm}$, and $0.01\,\mathrm{cm}$ standard deviations on average for the rotation error, translation error, and ADD, respectively.

3DMatch Dataset Experiments. We evaluate DeepPRO on the 3DMatch dataset to see if it can be directly extended to indoor/outdoor point clouds. Unfortunately, we observe that DeepPRO overfits to training data when using the same network architecture and hyper-parameters. It is because the current point cloud generation network architecture is optimized to render the geometry of compact objects. We believe that the core idea of our approach, i.e., generate corresponding points, can be applied to indoor/outdoor data with different network architectures.

Failure case. We collect all failure cases on the Linemod dataset and sort them based on ADD. As shown in Table 1, we provide 15 worst cases and analyze why they have large errors. There are two main failure categories. First, object mask sometimes cover different objects or backgrounds. For example, Table 2 shows that the Phone object is slightly occluded by the Can object in view 1 and occluded by the Cup object in view 2. These Can and Cup objects are included in the ground truth mask of the Phone object. Therefore, the input point cloud contains points from different objects. As another example, in Table 7, the input point cloud has a point from very far away background. Second, the ground truth label is incorrect for some cases. Table 8 shows an example that the ground truth transform does not align the

Table 1: Summary of worst 15 failure cases on the Linemod dataset. Sorted by ADD.

Object	ADD (cm)	Visualization	Failure category
Phone	3.00	Table 2	Wrong object mask (including different object)
Phone	2.62	Table 3	Wrong object mask (including different object)
Hole puncher	2.03	Table 4	Wrong object mask (including different object)
Hole puncher	1.83	Table 5	Wrong object mask (including different object)
Iron	1.75	Table 6	Deformed object
Driller	1.73	Table 7	Wrong object mask (including very far away background)
Cam	1.71	Table 8	Wrong ground truth transform
Iron	1.70	Table 9	Algorithm failure
Cam	1.62	Table 10	Wrong ground truth transform
Iron	1.59	Table 11	Wrong object mask (including different object)
Duck	1.58	Table 12	Wrong object mask (including different object)
Cam	1.57	Table 13	Wrong ground truth transform
Iron	1.52	Table 14	Algorithm failure
Can	1.49	Table 15	Wrong ground truth transform
Cam	1.39	Table 16	Wrong ground truth transform

point clouds well. For all examples in this category (Table 8,10,13,15,16), DeepPRO registers point clouds better than the ground truth transform.

References

[1] Yasuhiro Aoki, Hunter Goforth, Rangaprasad Arun Srivatsan, and Simon Lucey. PointNetLK: Robust & efficient point cloud registration using PointNet. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2019. 2

[2] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. SegNet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(12):2481–2495, 2017. 1

[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 *IEEE Conference on Computer Vision and Pattern Recognition*, 2020. 2

[4] Paul J. Besl and Neil D. MacKay. A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2), 1992. 3

[5] Christopher Choy, Wei Dong, and Vladlen Koltun. Deep global registration. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2020. 2

[6] Georgios Dimitrios Evangelidis and Radu Horaud. Joint alignment of multiple point sets with batch and incremental expectation-maximization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(6):1397–1410, 2017. 3

[7] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? The KITTI vision benchmark suite. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012. 2

[8] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: Training Imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017. 1

[9] Bing Jian and Baba C. Vemuri. Robust point set registration using Gaussian mixture models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(8):1633–1645, 2011. 3

[10] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. 1

[11] François Pomerleau, Ming Liu, Francis Colas, and Roland Siegwart. Challenging data sets for point cloud registration algorithms. *The International Journal of Robotics Research*, 31(14), 2012. 2

[12] Yue Wang and Justin M Solomon. Deep closest point: Learning representations for point cloud registration. In *IEEE International Conference on Computer Vision*, 2019. 2

[13] Yue Wang and Justin M Solomon. PRNet: Self-supervised learning for partial-to-partial registration. In *Advances in Neural Information Processing Systems*, 2019. 1, 2

[14] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph CNN for learning on point clouds. *ACM Transactions on Graphics*, 2019. 1

[15] Heng Yang, Jingnan Shi, and Luca Carlone. TEASER: Fast and certifiable point cloud registration. *IEEE Transactions on Robotics*, 37(2):314–333, 2020. 3

[16] Jiaolong Yang, Hongdong Li, and Yunde Jia. Go-ICP: Solving 3D registration efficiently and globally optimally. In *IEEE International Conference on Computer Vision*, 2013. 3

[17] Zi Jian Yew and Gim Hee Lee. RPM-Net: Robust point matching using learned features. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2020. 2

[18] Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel,

Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training BERT in 76 minutes. In *International Conference on Learning Representations*, 2020. 1

[19] Andy Zeng, Shuran Song, Matthias Nießner, Matthew Fisher, Jianxiong Xiao, and Thomas Funkhouser. 3DMatch: Learning local geometric descriptors from RGB-D reconstructions. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017. 2

[20] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Fast global registration. In *European Conference on Computer Vision*, 2016. 3

Table 2: Results on the Linemod Phone object using frame 293 and 303.

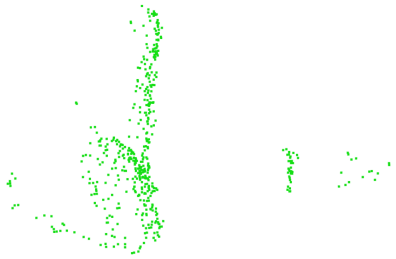
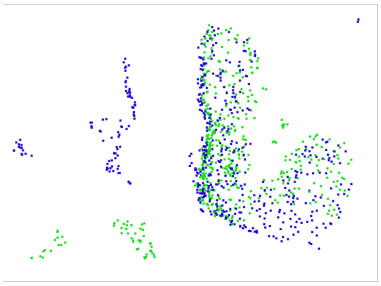
RGB (left: view 1, right: view 2)		
Point cloud from view 1 (left: observed viewpoint, right: different viewpoint)		
Point cloud from view 2 (left: observed viewpoint, right: different viewpoint)		
Ground truth alignment (left: observed viewpoint, right: different viewpoint)		
DeepPRO (left: observed viewpoint, right: different viewpoint)		

Table 3: Results on the Linemod Phone object using frame 293 and 302.

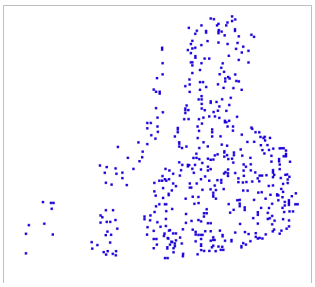
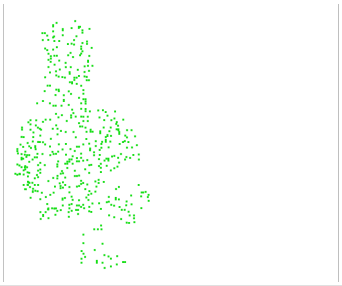
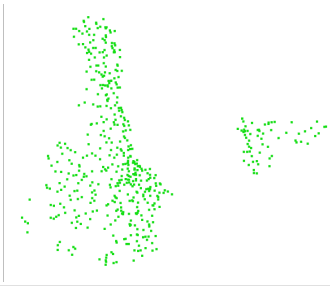
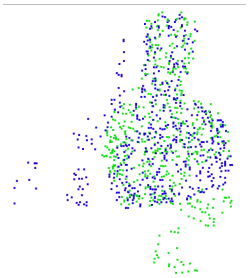
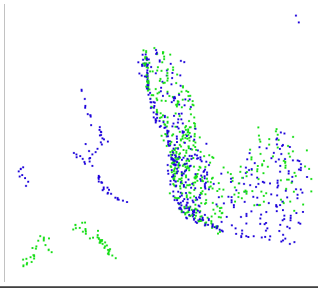
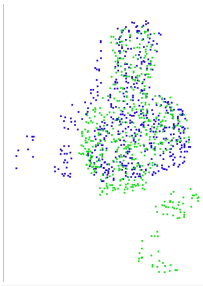
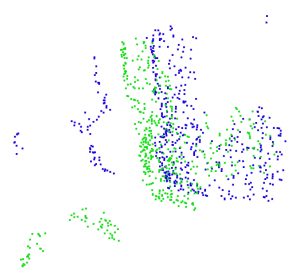
RGB (left: view 1, right: view 2)		
Point cloud from view 1 (left: observed viewpoint, right: different viewpoint)		
Point cloud from view 2 (left: observed viewpoint, right: different viewpoint)		
Ground truth alignment (left: observed viewpoint, right: different viewpoint)		
DeepPRO (left: observed viewpoint, right: different viewpoint)		

Table 4: Results on the Linemod Hole puncher object using frame 613 and 700.

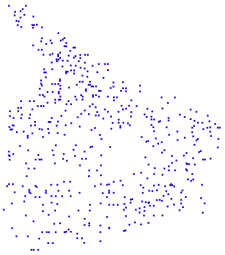
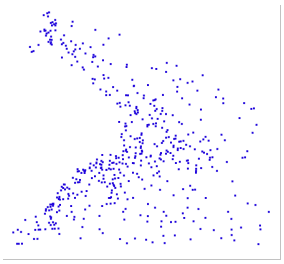
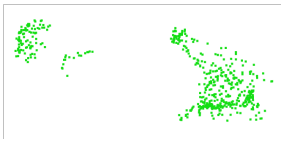
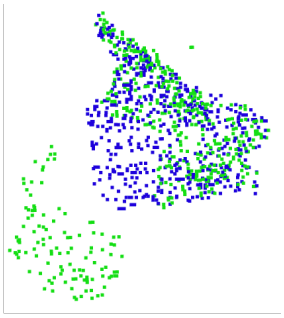
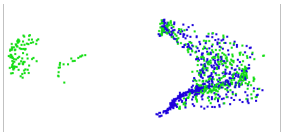
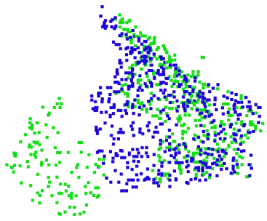
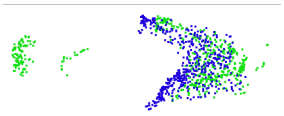
RGB (left: view 1, right: view 2)		
Point cloud from view 1 (left: observed viewpoint, right: different viewpoint)		
Point cloud from view 2 (left: observed viewpoint, right: different viewpoint)		
Ground truth alignment (left: observed viewpoint, right: different viewpoint)		
DeepPRO (left: observed viewpoint, right: different viewpoint)		

Table 5: Results on the Linemod Hole puncher object using frame 613 and 699.

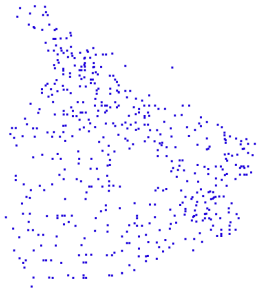
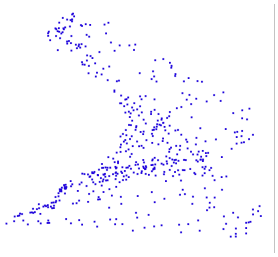
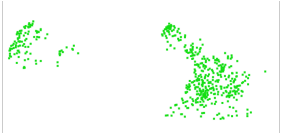
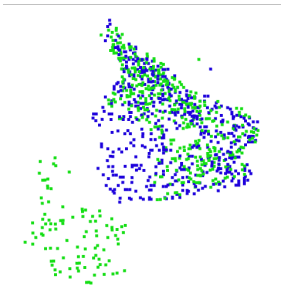
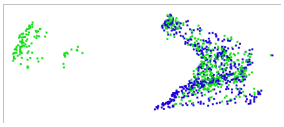
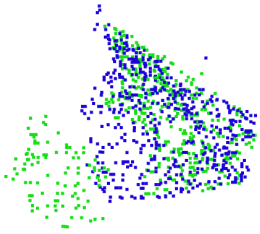
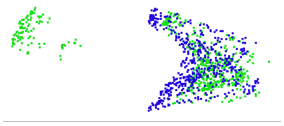
RGB (left: view 1, right: view 2)		
Point cloud from view 1 (left: observed viewpoint, right: different viewpoint)		
Point cloud from view 2 (left: observed viewpoint, right: different viewpoint)		
Ground truth alignment (left: observed viewpoint, right: different viewpoint)		
DeepPRO (left: observed viewpoint, right: different viewpoint)		

Table 6: Results on the Linemod Iron object using frame 205 and 964.

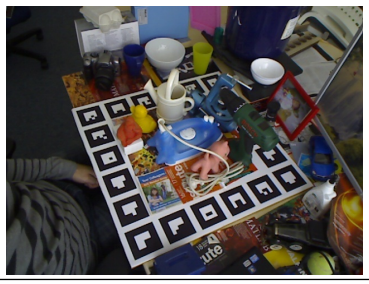
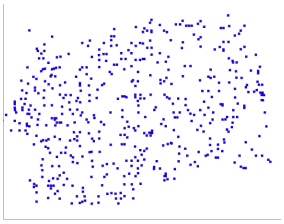
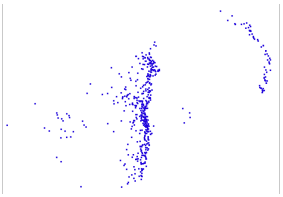
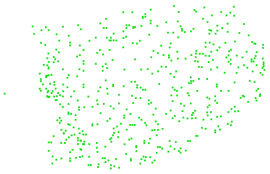
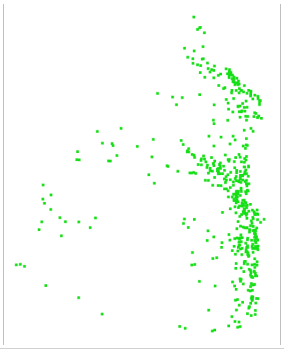
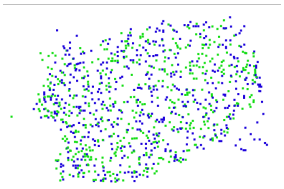
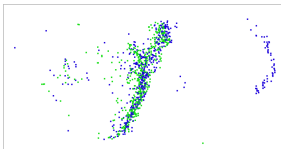
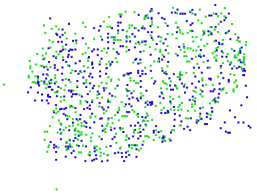
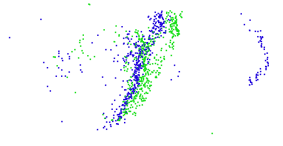
RGB (left: view 1, right: view 2)		
Point cloud from view 1 (left: observed viewpoint, right: different viewpoint)		
Point cloud from view 2 (left: observed viewpoint, right: different viewpoint)		
Ground truth alignment (left: observed viewpoint, right: different viewpoint)		
DeepPRO (left: observed viewpoint, right: different viewpoint)		

Table 7: Results on the Linemod Drill object using frame 956 and 957.

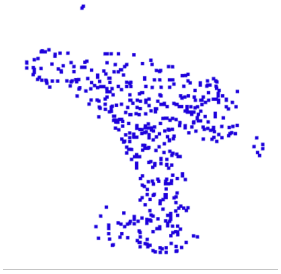
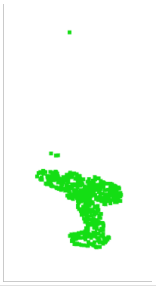
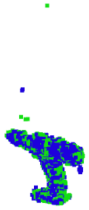
RGB (left: view 1, right: view 2)		
Point cloud from view 1 (left: observed viewpoint, right: different viewpoint)		
Point cloud from view 2 (left: observed viewpoint, right: different viewpoint)		
Ground truth alignment (left: observed viewpoint, right: different viewpoint)		
DeepPRO (left: observed viewpoint, right: different viewpoint)		

Table 8: Results on the Linemod Cam object using frame 544 and 545.

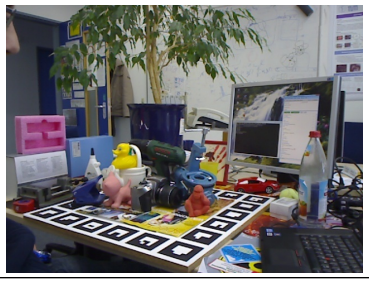
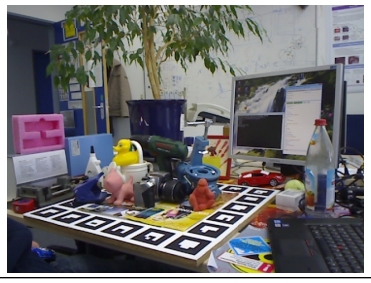
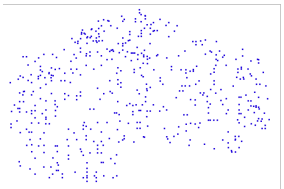
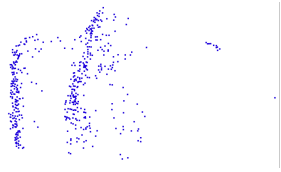
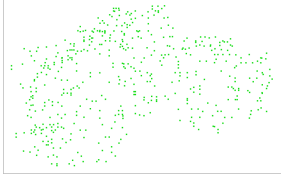
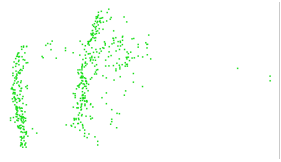
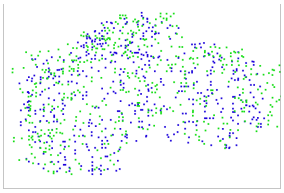
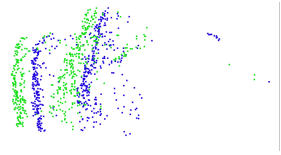
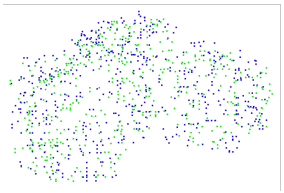
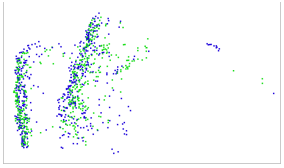
RGB (left: view 1, right: view 2)		
Point cloud from view 1 (left: observed viewpoint, right: different viewpoint)		
Point cloud from view 2 (left: observed viewpoint, right: different viewpoint)		
Ground truth alignment (left: observed viewpoint, right: different viewpoint)		
DeepPRO (left: observed viewpoint, right: different viewpoint)		

Table 9: Results on the Linemod Iron object using frame 864 and 894.

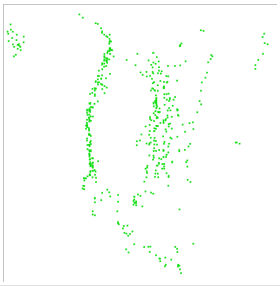
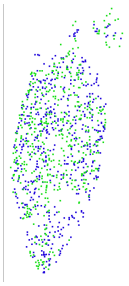
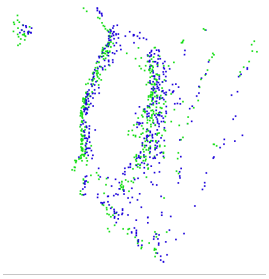
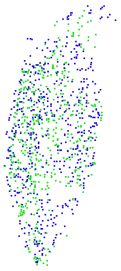
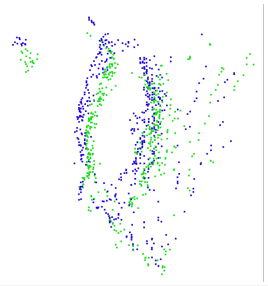
RGB (left: view 1, right: view 2)		
Point cloud from view 1 (left: observed viewpoint, right: different viewpoint)		
Point cloud from view 2 (left: observed viewpoint, right: different viewpoint)		
Ground truth alignment (left: observed viewpoint, right: different viewpoint)		
DeepPRO (left: observed viewpoint, right: different viewpoint)		

Table 10: Results on the Linemod Cam object using frame 106 and 107.

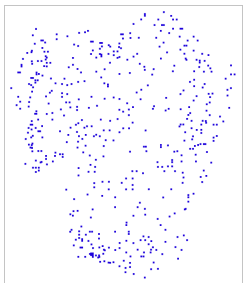
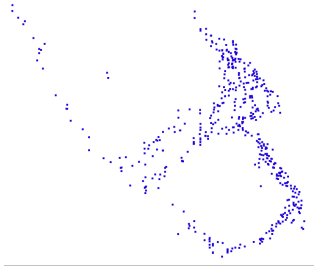
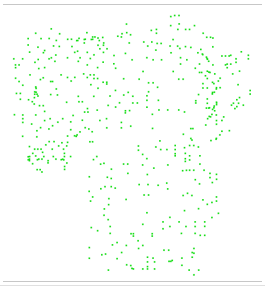
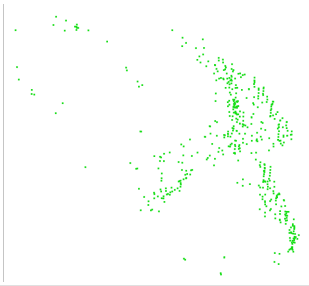
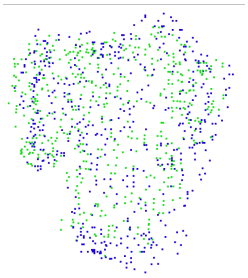
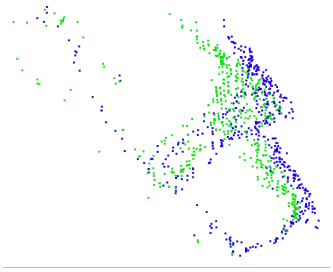
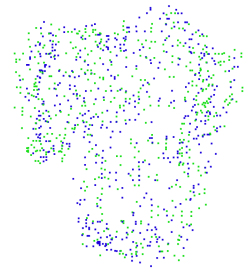
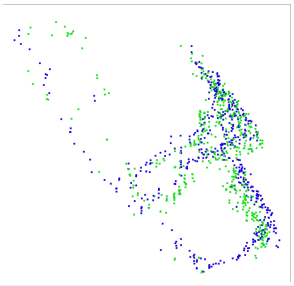
RGB (left: view 1, right: view 2)		
Point cloud from view 1 (left: observed viewpoint, right: different viewpoint)		
Point cloud from view 2 (left: observed viewpoint, right: different viewpoint)		
Ground truth alignment (left: observed viewpoint, right: different viewpoint)		
DeepPRO (left: observed viewpoint, right: different viewpoint)		

Table 11: Results on the Linemod Iron object using frame 199 and 248.

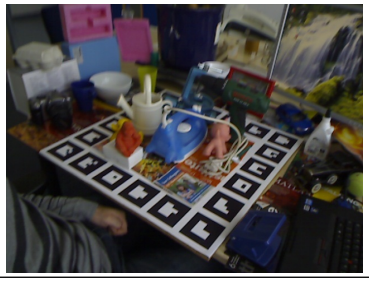
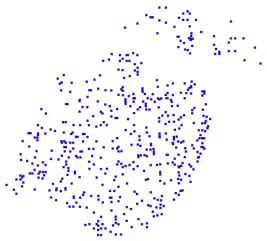
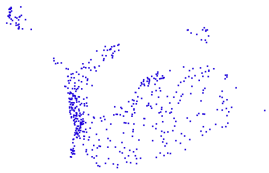
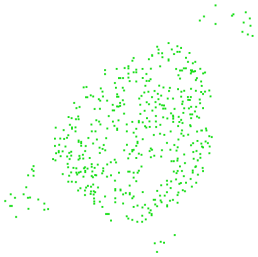
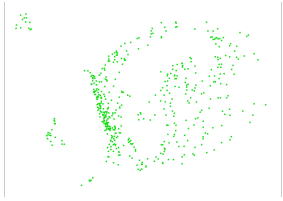
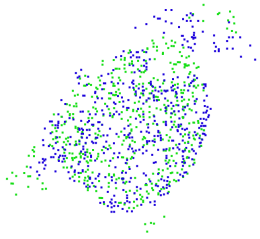
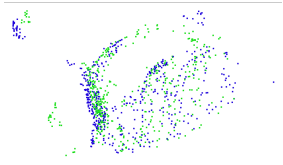
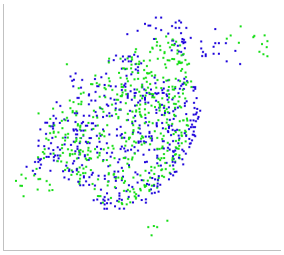
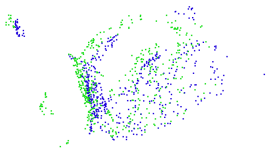
RGB (left: view 1, right: view 2)		
Point cloud from view 1 (left: observed viewpoint, right: different viewpoint)		
Point cloud from view 2 (left: observed viewpoint, right: different viewpoint)		
Ground truth alignment (left: observed viewpoint, right: different viewpoint)		
DeepPRO (left: observed viewpoint, right: different viewpoint)		

Table 12: Results on the Linemod Duck object using frame 81 and 94.

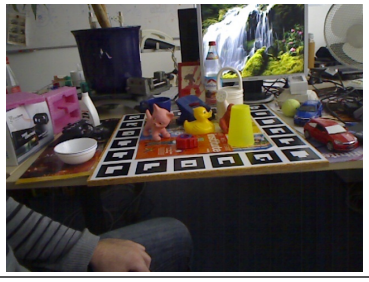
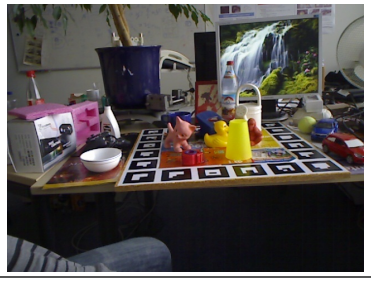
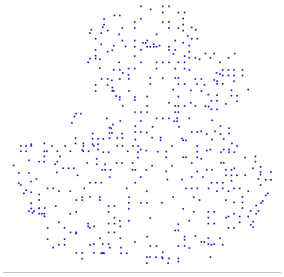
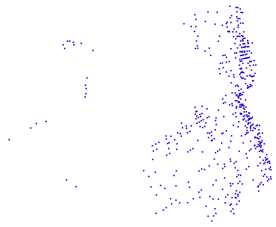
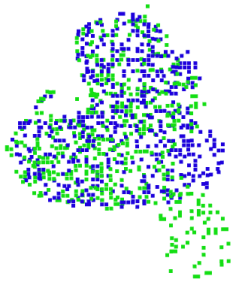
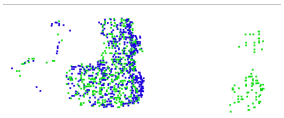
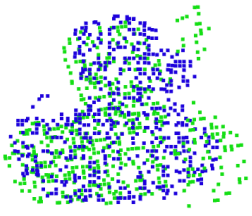
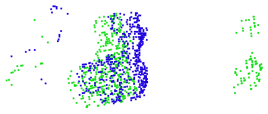
RGB (left: view 1, right: view 2)		
Point cloud from view 1 (left: observed viewpoint, right: different viewpoint)		
Point cloud from view 2 (left: observed viewpoint, right: different viewpoint)		
Ground truth alignment (left: observed viewpoint, right: different viewpoint)		
DeepPRO (left: observed viewpoint, right: different viewpoint)		

Table 13: Results on the Linemod Cam object using frame 1101 and 1038.

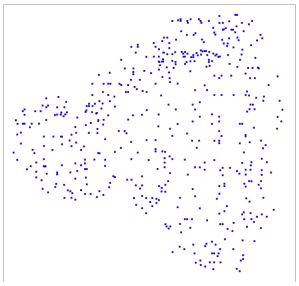
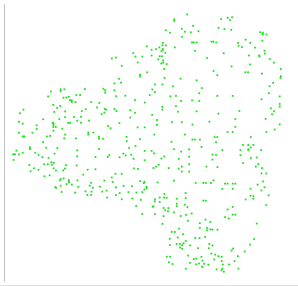
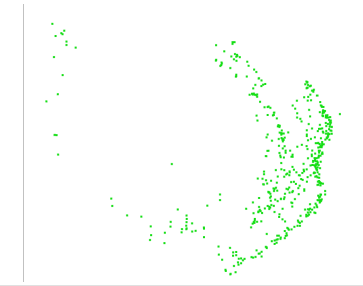
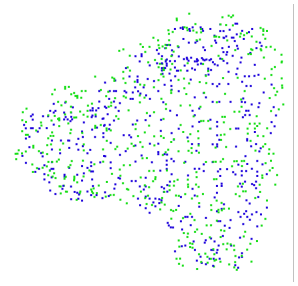
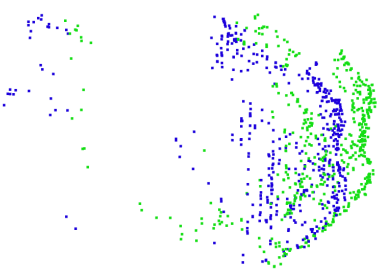
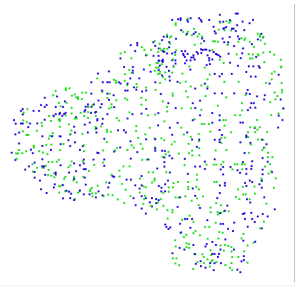
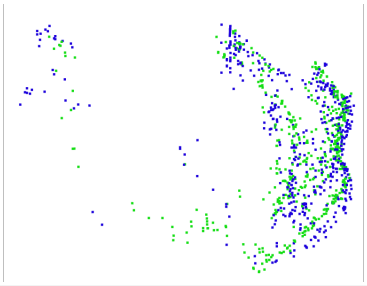
RGB (left: view 1, right: view 2)		
Point cloud from view 1 (left: observed viewpoint, right: different viewpoint)		
Point cloud from view 2 (left: observed viewpoint, right: different viewpoint)		
Ground truth alignment (left: observed viewpoint, right: different viewpoint)		
DeepPRO (left: observed viewpoint, right: different viewpoint)		

Table 14: Results on the Linemod Iron object using frame 140 and 141.

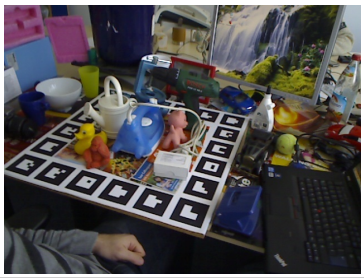
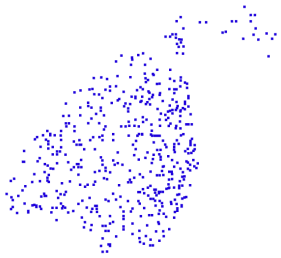
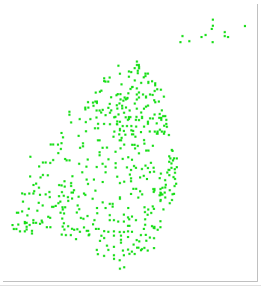
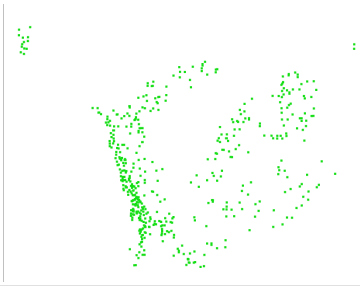
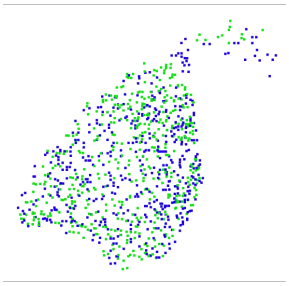
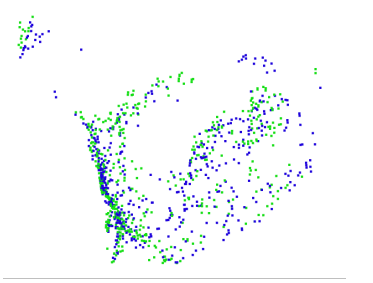
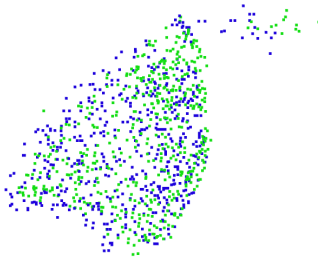
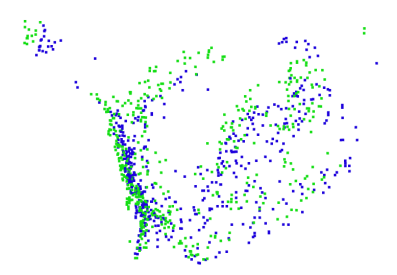
RGB (left: view 1, right: view 2)		
Point cloud from view 1 (left: observed viewpoint, right: different viewpoint)		
Point cloud from view 2 (left: observed viewpoint, right: different viewpoint)		
Ground truth alignment (left: observed viewpoint, right: different viewpoint)		
DeepPRO (left: observed viewpoint, right: different viewpoint)		

Table 15: Results on the Linemod Can object using frame 838 and 839.

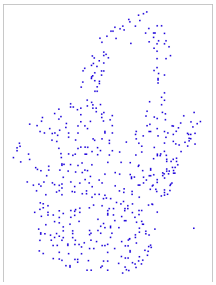
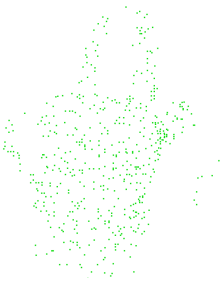
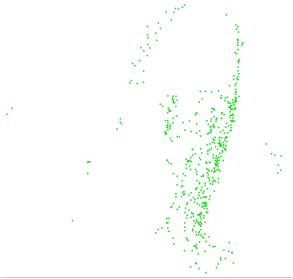
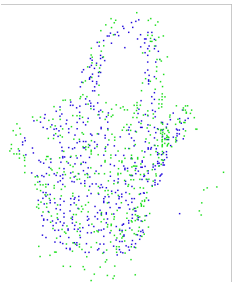
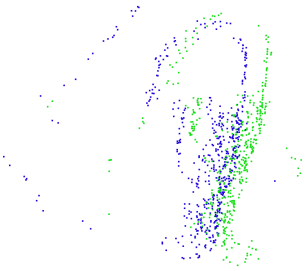
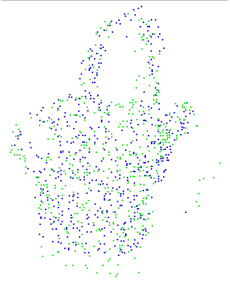
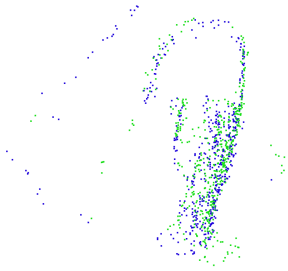
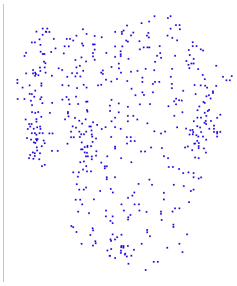
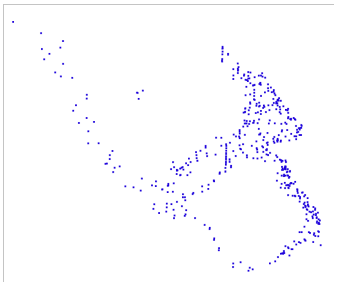
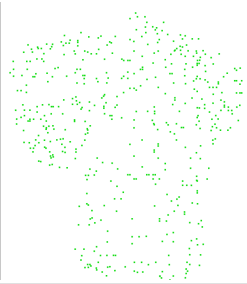
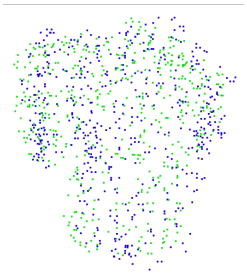
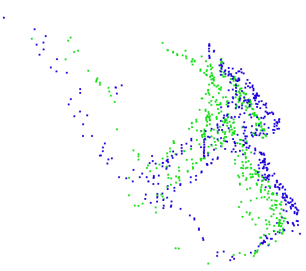
RGB (left: view 1, right: view 2)		
Point cloud from view 1 (left: observed viewpoint, right: different viewpoint)		
Point cloud from view 2 (left: observed viewpoint, right: different viewpoint)		
Ground truth alignment (left: observed viewpoint, right: different viewpoint)		
DeepPRO (left: observed viewpoint, right: different viewpoint)		

Table 16: Results on the Linemod Cam object using frame 106 and 108.

RGB (left: view 1, right: view 2)		
Point cloud from view 1 (left: observed viewpoint, right: different viewpoint)		
Point cloud from view 2 (left: observed viewpoint, right: different viewpoint)		
Ground truth alignment (left: observed viewpoint, right: different viewpoint)		
DeepPRO (left: observed viewpoint, right: different viewpoint)