PoGO-Net: Pose Graph Optimization with Graph Neural Networks

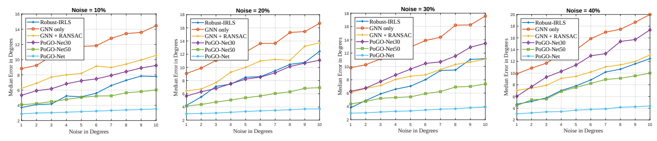
Supplementary Materials

6.1. Overview

In the supplementary materials, we will first provide more details of the ablation study on the de-noising layers of the PoGO-Net in §6.2, followed by the discussion on its capability of generalization by integrating the network with ORB-SLAM in §6.3. We finally report and analyze the full experimental result on the Photo Tourism Dataset [60] in §6.4.

6.2. Ablation Study on De-Noising

In our full network, we arrange the layers by setting a de-noise layer before every GNN layer, such that the network updates the connectivity in each iteration by message passing. To better understand the effects of the de-noise layers, we conduct the ablation study on the 7Scenes dataset [50] with PoGO-Net variants. In detail, we randomly remove the de-noise layers which consist of 50%, 70% and 100% of the amount of the de-noise layers in the original network, re-train the networks with the new settings and test them on the *Pumpkin* scene testing set with additional noise.



(a) Noise = 10% (b) Noise = 20% (c) Noise = 30% (d) Noise = 40%

Figure 5: Ablation study with different de-noise layers settings on the 7Scenes Dataset [50], with noise distributed to a) 10% b) 20% c) 30% d) 40% of the view-graph edges on the Pumpkin scene.

Specifically, to add the extra corruption into the scene, we bring in a uniformly increasing noise ranging from 1° to 10°, and distribute the noise randomly on the 10%, 20%, 30% and 40% of the edges in the initial view-graph. We repeat the experiment with different noise levels and record the performance, we also include results of the state-of-the-art conventional approach Robust-IRLS [10]. The results are given in Fig. 5.

For 'GNN-only' variation, there are only GNN layers in the network and it is very difficult to initialize the nodes in the view-graph as it has been witnessed that the errors on the edges have been severely propagated over the graph during the initialization. Therefore we first manually filter out the outlier edges in randomly selected cycles in the view-graph by enforcing the cycle identity. Similar with 'GNN only', we remove all the de-noise layers in 'GNN+RANSAC' but we pre-process the view-graph with the visual information. Particularly, we run RANSAC iterations to remove the erroneous feature matching between image frames and only keep the edges with more than 60% valid correspondences.

It can be observed that with the low percentage of edges corrupted, the PoGO-Net variations with fewer de-noise layers can still work but yields lower accuracy with increasing noise level, while those equipped with purely GNN layers perform poorly against the corruptions. When the noisy edge percentage is high, the performance of all the PoGO-Net variations deteriorates quickly with the rising noise. It is also noteworthy that, though the conventional approach Robust-IRLS [10] leverages robust loss functions, the performance starts to degrade vastly with more than 20% of the edges corrupted. PoGO-Net remains to perform well and achieves steadily high accuracy against developing noise levels and rising amounts of corrupted edges, further demonstrating the robustness of the network.

6.3. §5.4 Results on KITTI Odometry [21]

We test PoGO-Net on the KITTI Odometry [21] to further demonstrate its capability of generalization. Particularly, we train PoGO-Net with large scale outdoor datasets including the Cambridge dataset [30] and the Photo Tourism Dataset [60], which are similar with the KITTI Odometry scenes in size and scale, followed by the training of the network on eight sequences in the KITTI dataset. We test PoGO-Net on Seq.00, Seq.02 and Seq.08 as the most challenging sequences in the dataset.

Specifically, we first run modified ORB-SLAM [43] on the whole dataset and use the view-graphs without global BA (GBA) as the input to the PoGO-Net. In addition, we tune down the local mapping thread parameters such that the intensity of local BA is lower, resulting denser and noisier intermediate view-graphs. As shown in Table 5, the input to PoGO-Net are much less accurate compared with the

Table 5: Quantitative results on KITTI Odometry. We report the RMSE(m), mean angular errors(°), global BA iterations and runtime(s) (PoGO-Net runtime + BA runtime) on CPU, compared with the state-of-the-art conventional PGO approach utilized by ORB-SLAM.

Seq.			**RMSE/Ang. Err.** (before)		**RMSE/Ang. Err.** (after)		**# BA iter.**		**Runtime(s)**	
	# Frames	Scale(m x m)	ORB	Our Input	ORB+GBA	Ours+GBA	ORB	Ours	ORB+GBA	Ours+GBA
Seq.00	4541	564 x 496	6.68m / 15.63°	26.82m / 27.29°	5.33m / 12.77°	2.03m / 1.92°	20	3	24.83	2.03 + 3.56
Seq.02	4661	599 x 946	21.75m / 17.99°	74.65m / 29.87°	21.28m / 14.23°	7.66m / 1.08°	20	3	30.07	2.62 + 3.97
Seq.08	4071	808 x 391	46.58m / 21.49°	68.29m / 28.25°	46.68m / 19.68°	5.89m / 2.17°	20	4	25.60	2.39 + 1.88

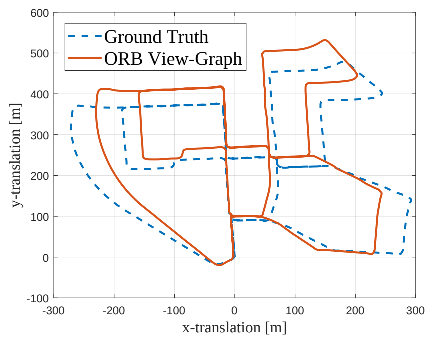
camera poses given by the regular ORB-SLAM before conducting GBA. Since we leverage MRA to process the PGO, only the camera orientations are optimized, we then handle the translation averaging by exploiting *g2o* [24] with the camera orientations treated as fixed.

As reported in Table 5, PoGO-Net with GBA has achieved significantly higher accuracy on all three testing sequences compared with the state-of-the-art conventional PGO approach. Moreover, as it requires so many iterations for the GBA to converge (it might also fail to converge in some cases), the ORB-SLAM system runs GBA for 20 iterations while PoGO-Net requires less than 5 iterations in all our experiments. We further provide the qualitative results of running PoGO-Net on the three testing sequences in Fig. 6, where we record the trajectory initialized by ORB-SLAM, the trajectory optimized by PoGO-Net and the final trajectory optimized for the camera translations for each sequence. It can be seen that the camera poses are already noticeably accurate after running PoGO-Net even without the final translation fine-tuning.

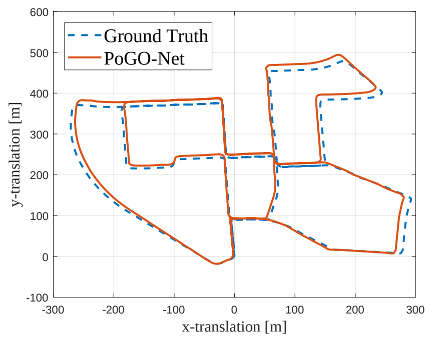
We also notice that PoGO-Net achieves up to 6x faster processing speed compared with the conventional approach, validating its potential to be extended to a full SfM/SLAM system, fulfilling the real-time requirements.

6.4. Full Results on the Tourism [60]

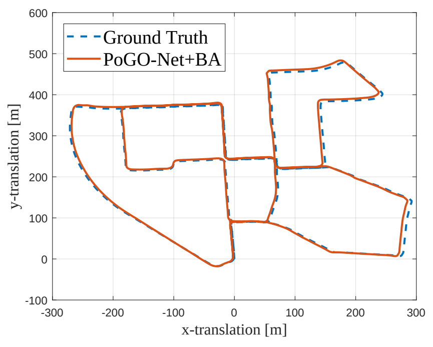
We report the angular errors and runtime on the Photo Tourism Dataset [60] in Table 6. It can be seen our proposed network has achieved the best performance on most of the dataset. Among the datasets Piccadilly (Picca.), Arts Quad (ArtsQ.) and Trafalgar (Trafag.) are the three largest datasets with more than 300K edges. While on these three challenging datasets PoGO-Net falls slightly short for the conventional approach Robust-IRLS [10] on accuracy, it has achieved up to 500x faster processing speed.



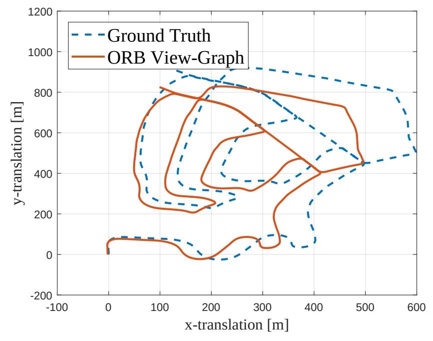
(a) seq.00 Init



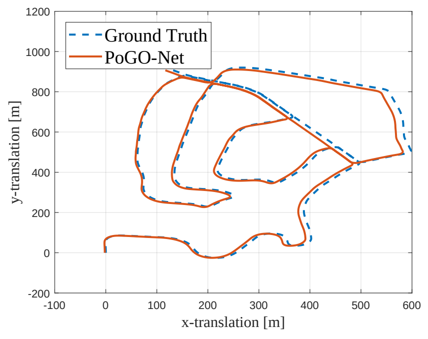
(b) seq.00 PoGO-Net



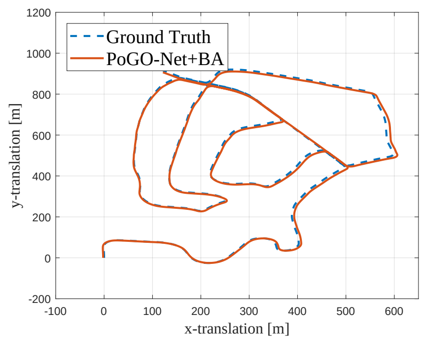
(c) seq.00 PoGO-Net with BA



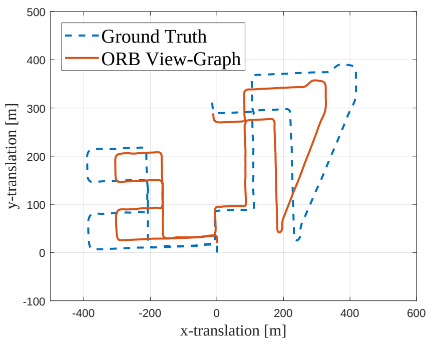
(d) seq.02 Init



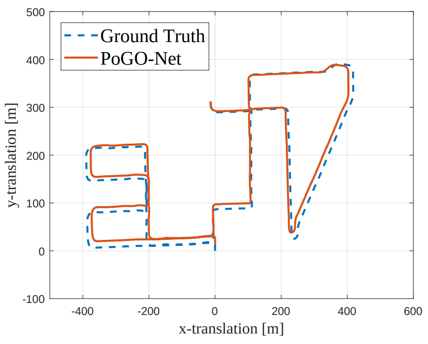
(e) seq.02 PoGO-Net



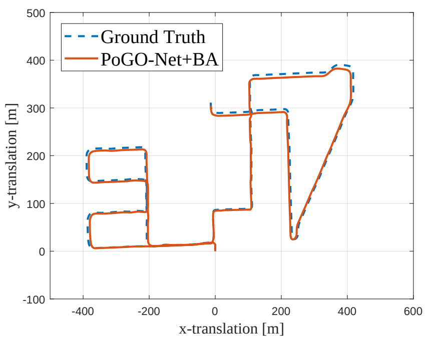
(f) seq.02 PoGO-Net with BA



(g) seq.08 Init



(h) seq.08 PoGO-Net



(i) seq.08 PoGO-Net with BA

Figure 6: Qualitative results on Seq.00, 02 and 08 from the KITTI Odometry [21] dataset. The leftmost column presents the initial trajectory given by ORB-SLAM [43] without running global BA, the middle column shows the trajectory after running PoGO-Net on the initial view-graph, the rightmost column displays the final optimized trajectory by running BA with camera orientations fixed.

Table 6: Experiment results on the Tourism Dataset [60]. We report the angular errors (°) and runtime (s) on CPU. The best results are highlighted

Scene	# Nodes	# Edges		**IRLS** [9]	**Robust-IRLS** [10]	**Weiszfeld** [25]	**Arrigoni** [4]	**Wang** [58]	**DISCO** [12]	**NeuRoRA** [45]	**PoGO-Net**
			mean	3.04	2.71	4.4	3.9	3.6	-	2.6	**1.91**
Ell. Isld.	247	20297	median	1.06	0.93	1.0	1.2	1.1	5.54	0.6	**0.43**
			runtime	3.2s	2.8s	8.9s	0.2s	2.6s	470s	**0.4s**	**0.43s**
			mean	3.64	3.67	4.9	6.2	5.3	-	4.9	**2.96**
Alamo	627	97206	median	1.30	1.32	1.4	1.2	1.1	7.86	1.2	**0.85**
			runtime	14.2s	15.1s	84.0s	2.7s	20.6s	3917s	2.2s	**1.74s**
			mean	1.25	1.22	2.1	4.8	2.0	-	1.2	**0.82**
M.N.D	474	52424	median	0.58	0.57	0.7	0.9	0.8	6.81	0.6	**0.37**
			runtime	8.5s	7.3s	41.5s	2.9s	10.1s	1608s	1.0s	**0.53s**
			mean	2.63	2.26	4.7	3.9	3.5	-	1.6	**1.17**
N.Dame	715	64678	median	0.78	0.71	0.8	1.0	0.9	7.48	0.6	**0.35**
			runtime	17.2s	22.5s	80.8s	4.2s	19.5s	4070s	2.0s	**1.24s**
			mean	5.12	5.19	26.4	22.0	10.1	36.0	**4.7**	4.93
Picca.	2508	319257	median	2.02	2.34	7.5	9.7	3.9	-	1.9	**1.75**
			runtime	353.5s	370.2s	1342.6s	43.7s	118.1s	15604s	5.9s	**3.19s**
			mean	2.71	2.66	3.8	3.9	2.9	-	1.9	**1.52**
NYC L.	376	20680	median	1.37	1.30	2.1	1.5	1.4	9.12	1.1	**0.88**
			runtime	3.5s	4.6s	14.4s	1.4s	3.2s	446s	**0.2s**	**0.24s**
			mean	4.1	3.99	4.8	10.8	6.2	-	3.0	**2.26**
P.D.P.	354	24710	median	2.07	2.09	1.3	1.2	1.1	12.11	**0.7**	0.81
			runtime	3.8s	4.1s	16.7s	0.6s	3.6s	583s	0.4s	**0.28s**
			mean	2.66	2.69	4.8	13.2	4.6	-	2.3	**1.55**
R.Frm.	1134	70187	median	1.58	1.57	1.8	8.2	3.5	35.36	1.3	**0.69**
			runtime	18.6	21.4	115.0s	16.8s	19.6s	1559s	**1.3s**	**1.26s**
			mean	3.42	3.41	4.7	4.6	2.9	-	2.6	**1.77**
T.o.L.	508	24863	median	2.52	2.50	2.9	1.8	1.5	10.38	1.4	**0.43**
			runtime	2.6s	2.4s	17.4s	3.9s	3.6s	479s	**0.3s**	0.38s
			mean	6.77	6.77	40.9	9.2	6.8	-	5.9	**3.3**
U.Sq.	930	25561	median	3.66	3.85	10.3	4.4	3.2	26.27	2.0	**1.25**
			runtime	9.0s	8.6s	42.8s	12.1s	4.1s	466s	0.6s	**0.29s**
			mean	2.6	2.45	5.7	4.5	3.5	-	2.5	**2.03**
Yorkm.	458	27729	median	1.59	1.53	2.0	1.6	1.3	26.17	0.9	**0.72**
			runtime	3.4s	4.3s	32.0s	2.5s	4.9s	641s	0.4s	**0.12s**
			mean	4.3	**3.6**	18.8	66.8	89.2	-	17.6	6.82
San.F.	7866	101512	median	3.9	3.4	16.4	43.9	75.5	54.38	12.6	**3.16**
			runtime	18.9s	15.2s	1462.7s	354.7s	27.2s	1413s	2.6s	**1.54s**
			mean	7.2	6.9	7.5	6.0	5.0	-	2.5	**2.33**
Mad.M.	394	23784	median	1.4	1.2	2.7	1.7	1.4	12.12	1.1	**0.96**
			runtime	3.5s	3.2s	14.5s	0.9s	3.6s	560s	0.2s	**0.12s**
			mean	9.1	8.2	11.7	19.3	10.1	-	**3.9**	4.26
Vien.C.	918	103550	median	3.9	1.2	1.9	2.39	1.8	22.35	1.5	**1.44**
			runtime	56.9s	48.1s	158.3s	6.0s	25.7s	4085s	2.1s	**1.53s**
			mean	5.1	**4.8**	34.4	35.2	6.0	-	27.5	9.88
ArtsQ.	5530	222044	median	4.3	3.5	23.1	15.8	**3.2**	87.12	7.3	4.53
			runtime	110.2s	116.1s	1980s	189.4s	73.9s	5227s	5.0s	**3.86s**
			mean	3.7	**3.5**	15.6	48.6	17.2	-	5.3	**3.5**
Trafag.	5433	680012	median	2.4	2.0	11.3	13.2	16.0	91.02	2.2	**1.7**
			runtime	844.3s	858.4s	5600s	167.4s	319.2s	43616s	**15.5s**	17.2s