Neural Prior for Trajectory Estimation

Chaoyang Wang[1] Xueqian Li[2,3] Jhony Kaesemodel Pontes[3] Simon Lucey[12]
[1]Carnegie Mellon University [2]University of Adelaide [3]Argo AI
chaoyanw@cs.cmu.edu {xueqian.li, simon.lucey}@adelaide.edu.au jpontes@argo.ai

1. Additional discussions

1.1. Difference to neural scene flow prior [5]

NSP [5] only models pairwise scene flows, while this paper models long-term correspondences in the form of 3D trajectories. To model the long-term *temporal* prior, our paper assumes that all trajectories within an input sequence are compressible by a bottleneck decoder. The compressibility assumption is motivated by previous trajectory-based NRSfM methods using rank/subspace constraints (see Sec. 3.1), which is not applicable to NSP as scene flows are already low dimensional (=3).

The proposed trajectory field also significantly improves efficiency when enforcing long-term consistency to reduce drift for scene flow integration. Consider the evaluation of long-term cyclic consistency loss with T frame intervals (T is large). Using NSP would require T steps of forward integration and then T steps of backward integration, which results in evaluating the MLP for $2T$ times. This is computationally intractable during iterative optimization when T is large. In comparison, the proposed trajectory field only needs 2 MLP evaluations, as the MLP directly outputs all positions at different frames.

1.2. Difference to Neural NRSfM [7]

Neural NRSfM [7] is a shape-based approach, using a decoder to model shapes (represented as a set of 3D points) rather than trajectories. For dense NRSfM (# points per frame $>>$ # frames), such formulation is less efficient. Consider just the output layer of the decoder, [7] requires $3P \times K$ parameters, where P refers to number of points per frame, and K is the layer width (=32 by default). This means for the synthetic face experiment (Tab.2) with 99 frames and $P = 28,000$, the output layer alone for [7] requires 2.7m #params, while our whole model requires only around 0.4m #params, since the size of our model is only proportional to #frames. In practise, our method also optimizes much faster ($\sim$15 mins v.s. 3+ hrs on a single GPU).

1.3. Societal impact

Though our method is not data-driven, the data collected for use in our method might still contain privacy issues. Potential malicious or unintended uses of surveillance are also possible.

2. NRSfM implementation details

2.1. Hyperparameters

Cost function. Thoroughout our evaluations, we set $\lambda_1 = 0.01$ and $\lambda_2 = 0.01$.

Network architecture. (i) the trajectory weight decoder f_α is a MLP with ReLU as the non-linear activation layers. The number of hidden units for each layers are 4(for sparse NRSfM) / 3(for dense NRSfM), 128, 128, 256. These numbers were picked heuristically without tuning. (ii) f_φ is modeled as a 5 layer MLP with each layer having 128 hidden units. (iii) f_τ is a MLP with 3 hidden layers, each with 128 hidden units. (iii) f_z is also a 5 layer MLP with 128 hidden units per layer. In addition, the 2D inputs are embedded using positional encoding [6] with 5 frequencies.

2.2. Optimization details

Bilevel optimization. Due to the orthogonality constraint of the camera matrices $\mathbf{M}_t$'s, directly minimizing Eq (4) in Sec. 5.1 using gradient descent does not converge to good solutions. Instead, we follow the bilevel optimization approach proposed by Wang *et al*. [9], *i.e*.

$$\min_{\boldsymbol{\theta}_\tau, \boldsymbol{\Phi}, \mathbf{z}} \min_{\mathbf{M}} \mathcal{C}_{\text{2D recon.}} + \lambda_1 \mathcal{C}_{\text{smooth traj.}} + \lambda_2 \|\boldsymbol{\Phi}\|_2^2, \quad (1)$$

where the upper and lower optimization share the same cost function, but are optimized with respect to different variables. For the lower level problem, we minimize the cost with respect to the camera matrices $\mathbf{M}_t$'s. Due to the terms $\mathcal{C}_{\text{traj. smooth}}$ and $\|\boldsymbol{\Phi}\|_2^2$ do not dependent on $\mathbf{M}_t$'s, the lower level problem in fact is $\min_{\mathbf{M}} \mathcal{C}_{\text{2D recon.}}$, which is an Orthographic-N-point(OnP) problem. Various efficient geometric or algebraic OnP solvers were proposed in litera-

ture (see survey [8]). In this work, we derived a simple algebraic solution inspired from the initialization step of the algorithm of Green and Gower [3].

A simple OnP solver. First, for each frame t, we minimize $\mathcal{C}_{\text{2D recon.}} = \|\mathbf{W}_t - \mathbf{M}_t\mathbf{S}_t\|_F$ as a least square problem, dropping the orthogonal constraint of $\mathbf{M}_t$. Then, given the least square solution $\tilde{\mathbf{M}}_t = \mathbf{W}\mathbf{S}^\dagger$, we perform metric upgrade to project $\tilde{\mathbf{M}}_t$ to an orthogonal matrix, *i.e.* $\mathbf{M}_t^* = \mathbf{U}\mathbf{V}^T$, where $\mathbf{U}$, $\mathbf{V}$ are matrices from svd$(\tilde{\mathbf{M}}_t)$. We implemented the above procedure using modern autograd packages, thus our OnP solver is a differentiable operator, which takes input from $\mathbf{W}$, $\mathbf{S}$ and outputs $\mathbf{M}$. Finally, we note that this simple OnP solver does not work for cases where 3D points are coplanar. Luckily, coplanarity is rare in real dynamic scenes, and never occurred in our experiments.

Single level reduction. With the differentiable OnP solver, we reduce the bilevel optimization to a single level optimization problem, *i.e.*

$$\min_{\boldsymbol{\theta}_\tau, \boldsymbol{\Phi}, \mathbf{z}} \|\mathbf{W} - \mathbf{M}^*(\mathbf{W}, \mathbf{S})\mathbf{S}\|_F + \alpha_1 \mathcal{C}_{\text{smooth traj.}} + \alpha_2 \|\boldsymbol{\Phi}\|_2^2, \tag{2}$$

where $\mathbf{M}^*(\mathbf{W}, \mathbf{S})$ denotes the OnP solution conditioned on the inputs $\mathbf{W}$, $\mathbf{S}$. This single level objective can then be solved using gradient descent.

Warm up with low-rank prior. We find that directly performing gradient descent with the above cost is sensitive to random seeds and sometimes diverges to poor solutions. To reduce variance of solutions, we develop the following trick. In the first 100 gradient descent iterations, we augment the cost with an additional cost, *i.e.* $\|\mathbf{S}\|_*$, which enforces the estimated shape $\mathbf{S}$ to have lower rank. This low-rank cost is only used as an initialization step, and it is removed from the objective after 100 iterations. On the other hand, keeping $\|\mathbf{S}\|_*$ as a permanent cost term would significantly degrade accuracy, as it strictly enforces the final solution to be low rank.

Optimization parameters. We use Adam optimizer with initial learning rate set as 0.001. We then linearly decrease learning when optimization progresses. For sparse NRSfM sequences, the learning rate is decreased by a factor of 0.9 every 500 iterations, and train for a total of 10k iterations; for dense NRSfM sequences, the learning rate is decreased by 0.9 every 2k iterations, and runs 100k iterations.

2.3. PUAL + NTP details

In Sec. 5.2, we combined PUAL [9] with our approach and observed improved accuracy on the benchmark. Since PUAL is a shape-based approach assuming the aligned 3D shapes are compressible with a decoder, the combination of NTP with PAUL is enforcing the bottleneck neural prior on both 3D trajectories and shapes. The combined optimization cost is

$$\begin{aligned} &\|\mathbf{S}' - \mathbf{R}\mathbf{S}_{\text{NT}}\|_F + \|\mathbf{S}' - \mathbf{R}\mathbf{S}_{\text{NS}}\|_F \\ &+\alpha_1 \mathcal{C}_{\text{smooth traj.}} + \alpha_2(\|\boldsymbol{\Phi}_{\text{NT}}\|_2^2 + \|\boldsymbol{\Phi}_{\text{NS}}\|_2^2), \end{aligned} \tag{3}$$

where $\boldsymbol{\Phi}_{\text{NT}}$, $\boldsymbol{\Phi}_{\text{NS}}$ are concatenation of trajectory codes and shape codes, $\mathbf{S}_{\text{NT}}$ and $\mathbf{S}_{\text{NS}}$ denotes 3D shape matrices produced by the trajectory decoder and the shape decoder respectively. $\mathbf{R} = \text{blockdiag}(\mathbf{R}_1, \mathbf{R}_2, \ldots, \mathbf{R}_F) \in \mathbb{R}^{3F\times 3F}$ is a block diagonal matrix stacking rotation matrices $\mathbf{R}_t \in SO(3)$ for each frame. And $\mathbf{S}'$ denotes the 3D shapes at camera frame, formed by concatenating input 2D observations $\mathbf{W}$ and unknown depth values $\mathbf{Z} \in \mathbb{R}^{F\times P}$, *i.e.*

$$\mathbf{S}' = \begin{bmatrix} \mathbf{w}_1 \\ \mathbf{z}_1 \\ \vdots \\ \mathbf{w}_F \\ \mathbf{z}_F \end{bmatrix}_{3F\times P}, \quad \mathbf{w}_t \in \mathbb{R}^{2\times P}, \quad \mathbf{z}_t \in \mathbb{R}^{1\times P}, \tag{4}$$

where $\mathbf{w}_t$ are the 2D measurement from t-th frame, and $\mathbf{z}_t$ denotes the unknown depth values at t-th frame.

The purpose of minimizing $\|\mathbf{S}' - \mathbf{R}\mathbf{S}_{\text{NT}}\|_F + \|\mathbf{S}' - \mathbf{R}\mathbf{S}_{\text{NS}}\|_F$ is to enforce the estimated shapes from the trajectory decoder, *i.e.* $\mathbf{S}_{\text{NT}}$ and the ones from the shape decoder, *i.e.* $\mathbf{S}_{\text{NS}}$ not only agree with the input 2D measurement, but also are consistent with each other.

To optimize the cost function, we adapt the same bilevel optimization strategy as described previously – for lower level problem, $\mathbf{R}$, $\mathbf{Z}$ are solved in closed form, and the solver is treated as a differentiable operation with $\mathbf{W}$, $\mathbf{S}_{\text{NT}}$, $\mathbf{S}_{\text{NS}}$ as inputs. Finally, the bilevel problem is reduced to a single level one and optimized with gradient descent.

2.4. Baseline method details

The baseline methods mentioned in Table 1, *i.e.* smooth traj. and low rank, are described as follow.

Smooth traj. In this baseline, we dropped the neural trajectory prior and only use the trajectory smoothness constraint. Thus the difference between this baseline to the proposed method is, the entire shape matrix $\mathbf{S}$ in Eq (4) is treated as variables to optimize, instead of forming it using the output from the trajectory decoder.

Low rank. We replace the multi-layer trajectory decoder with a single layer decoder, thus the trajectories are now formed as a linear combination of trajectory bases. By further limiting the bottleneck dimension, this approach is explicitly limiting the rank of its 3D reconstruction. In our

experiment, we tried rank=2, 4, 8, 12, 16, 32, 64, 128, and reported results using the setting (*i.e.* rank=12) which gave the best overall accuracy.

3. Lidar scene flow integration details

3.1. Details of NTP

Network architecture. For a fair comparison to Li *et al.* [5], we adopt their network architecture to model the trajectory code field f_φ. More specifically, f_φ is a 7 layer MLP with ReLU activations, and the layer width is 128. The rest of the architecture is kept the same as in the NRSfM experiments.

Encoding temporal input. We found that directly input time t to f_φ is insufficient and the optimization is slow to converge. Instead, we embedded t using cosine encoding, *i.e.* $\{\cos(\pi t), \cos(2\pi t), \ldots, \cos(2^N \pi t)\}$, where the number of frequencies $N = \log_2 F$. We also experimented with a learnable embedding strategy, which optimizes a 8-dim code vector to encode each time step t. We found that this strategy produced similar result to the cosine encoding approach. For simplicity, we chose to use the cosine encoding.

Optimization details. We use Adam optimizer with initial learning rate = 0.001. For each iteration, we randomly sample 4 out of 25 frames to evaluate the loss and perform backpropagation. The learning rate is reduced by a factor of 0.5 every 500 iterations, and the total number of iterations is 2000 per sequence (with 25 frames).

3.2. Baseline details

Euler integration. Given the estimated pairwise scene flow fields $f_{t\to t+1}, \forall t \in [1, F-1]$ from NSFP [5], the trajectory of a point $\mathbf{p} = (x, y, z)$ at the first frame can be traced through forward integration of the scene flows. Specifically, the points τ_t on the trajectory is calculated as:

1. $\tau_1 = (x, y, z)$.
2. $\tau_{t+1} = f_{t\to t+1}(\tau_t) + \tau_t, \forall t \in [1, F-1]$.

As shown above, estimating trajectories by integrating scene flow fields requires $F-1$ times of forward propagation of the neural networks $f_{t\to t+1}$.

KNN integration. Due to the scene flow estimation from learning-based methods such as FlowStep3D [4] is sparse, we need to first interpolate the flows so as to perform integration. Denote the output flow vectors from FlowStep3D at frame t as $\{\mathbf{v}_{t,1}, \mathbf{v}_{t,2}, \ldots, \mathbf{v}_{t,P}\}$, where P denotes the number of points at frame t, and $\mathbf{v}_{t,i}$ is the flow for the i-th point $\mathbf{p}_i$. We use nearest neighbor interpolation to interpolate flows, *i.e.* $f_{t\to t+1}(\mathbf{p}) = \mathbf{v}_{t,i^*}$, $i^* = \arg\min_i \|\mathbf{p} - \mathbf{p}_{t,i}\|_2$.

Table 1. **Ablation of number of frequencies for positional encoding.** We vary the number of frequencies for the positional encoding of f_φ in the dense NRSfM experiment. Numbers are reported as the normalized mean 3D error. We find that increasing the number of frequencies marginally increases reconstruction error for the traj. B sequence. Overall our method is not sensitive to different settings of positional encoding.

# freqs	0 (w/o pe)	2	3	4	5	6
traj. A	0.0324	0.0322	0.0332	0.0342	0.0324	0.0326
traj. B	0.0350	0.0357	0.0361	0.0364	0.0407	0.0386

Given the interpolated flows $f_{t\to t+1}$, we then perform Euler integration as described above.

4. Additional evaluations

4.1. Ablations of positional encoding for dense NRSfM

We evaluated different number of frequencies for the positional encoding for f_φ. As shown in Tab. 1, we found that increasing the number of frequencies increases 3D error for the traj. B sequence, though the difference is marginal. The accuracy for the traj. B sequence is not affected. Overall our method is not sensitive to the number of frequencies for positional encoding, and it performs best without positional encoding.

4.2. Additional lidar scene flow integration results

Ego vs. Ego motion free The motion of points in a lidar sequence comes from a combination of object motions (*e.g.*, cars, pedestrians, *etc.*) and ego motions of the autonomous vehicle (AV). Removing the ego motions may help simplify the problem and potentially improve the performance. Therefore, we investigated the effect of removing ego motions from the input data for our baselines. As shown in Tab. 2, our method consistently outperformed baselines regardless of their input data containing ego motion.

Evaluation on extremely sparse point clouds. In addition, we evaluated the methods using the nuScenes dataset. We extracted the first 25 frames from each of the 150 validation sequences. We notice the point clouds from nuScenes are extremely sparse—the number of points ranges from 2k to 7k. As a result, the results for all compared methods are noisy. We provide the quantitative results in Tab. 2.

Visual results. We provide more visual results for trajectory estimation, point cloud densification in **supp.html**.

References

[1] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Gi-

Table 2. **Trajectory accuracy for data with or without ego motions on Argoverse and nuScenes.** We report Chamfer distance results for all three methods with different integration strategies. ✓ is the original dataset with ego motions of the AV, and ✗ means that we removed the ego motion from the dataset. ↑ means the higher the better, and ↓ means the lower the better. Our method achieved relatively low errors despite the ego motions included in the scene.

	Argoverse [2]							nuScenes [1]						
	ego motions	Acc. 0.5 (%)↑	Acc. 1 (%)↑	Out. (%)↓	cd-1 (m)↓	cd-10 (m)↓	cd-24 (m)↓	ego motions	Acc. 0.5 (%)↑	Acc. 1 (%)↑	Out. (%)↓	cd-1 (m)↓	cd-10 (m)↓	cd-24 (m)↓
FlowStep3D [4]	✓	5.25	6.81	87.20	0.23	4.27	12.68	✓	19.12	24.90	66.48	0.38	3.16	**11.34**
(KNN Int.)	✗	5.18	6.78	87.22	0.20	3.97	13.94	✗	18.57	23.70	67.24	0.37	3.16	12.60
NSFP [5]	✓	45.18	59.86	28.90	0.11	5.35	21.30	✓	35.10	51.88	35.48	**0.26**	4.24	15.35
(KNN Int.)	✗	43.40	53.54	37.91	**0.08**	5.06	19.23	✗	34.84	52.18	33.74	**0.23**	3.41	**12.15**
NSFP [5]	✓	45.28	59.52	29.33	0.10	4.43	14.09	✓	35.09	51.83	35.49	0.46	4.03	16.03
(Euler Int.)	✗	43.15	53.14	38.08	0.09	4.17	11.74	✗	34.86	52.17	33.73	0.44	3.38	13.36
NTP (Ours)	✓	**52.28**	**69.88**	**20.95**	**0.08**	**2.41**	**9.77**	✓	34.92	**53.21**	**32.90**	0.40	**2.70**	13.81

ancarlo Baldan, and Oscar Beijbom. nuScenes: A multimodal dataset for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11621–11631, 2020. 4

[2] Ming-Fang Chang, John Lambert, Patsorn Sangkloy, Jagjeet Singh, Slawomir Bak, Andrew Hartnett, De Wang, Peter Carr, Simon Lucey, Deva Ramanan, et al. Argoverse: 3D tracking and forecasting with rich maps. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8748–8757, 2019. 4

[3] John C Gower, Garmt B Dijksterhuis, et al. *Procrustes problems*, volume 30. Oxford University Press on Demand, 2004. 2

[4] Yair Kittenplon, Yonina C Eldar, and Dan Raviv. FlowStep3D: Model unrolling for self-supervised scene flow estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4114–4123, 2021. 3, 4

[5] Xueqian Li, Jhony Kaesemodel Pontes, and Simon Lucey. Neural scene flow prior. In *Neural Information Processing Systems (NeurIPS)*, 2021. 1, 3, 4

[6] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing scenes as neural radiance fields for view synthesis. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 405–421. Springer, 2020. 1

[7] Vikramjit Sidhu, Edgar Tretschk, Vladislav Golyanik, Antonio Agudo, and Christian Theobalt. Neural dense non-rigid structure from motion with latent space constraints. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020. 1

[8] Carsten Steger. Algorithms for the orthographic-n-point problem. *Journal of Mathematical Imaging and Vision*, 60(2):246–266, 2018. 2

[9] Chaoyang Wang and Simon Lucey. PAUL: Procrustean autoencoder for unsupervised lifting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 434–443, 2021. 1, 2