Optimizing Odometry Accuracy through Temporal Information in Self-Supervised Deep Networks

Nazrul Ismail
School of Digital Science
Universiti Brunei Darussalam
Brunei Darussalam
20m8540@ubd.edu.bn

Ong Wee Hong
School of Digital Science
Universiti Brunei Darussalam
Brunei Darussalam
weehong.ong@ubd.edu.bn

Owais Ahmed Malik
School of Digital Science
Universiti Brunei Darussalam
Brunei Darussalam
owais.malik@ubd.edu.bn

***Abstract*— In this paper, we propose a self-supervised deep recurrent convolutional network with a self-attention pipeline for LiDAR-Inertial odometry estimation that leverages temporal information. In contrast to existing learning-based approaches, our network does not require expensive ground truth and it utilizes sliding window optimization for self-supervisory signal for training. The results indicate that the proposed network is able to achieve accuracy levels comparable to conventional model-based methods and outperform other learning-based approaches that do not incorporate temporal information. Our findings provide strong evidence for the potential of incorporating temporal information in deep networks for LiDAR-Inertial odometry and highlight the effectiveness of our proposed self-supervised approach.**

I. INTRODUCTION

Ego-motion estimation is crucial for safe navigation in robotics and autonomous driving. With the success of the deep neural network, recent progress has attracted many researchers in solving odometry estimation via learning from data. However, scalability issues and ground truth requirements pose challenges for real-world applications. To address this, we propose a self-supervised learning approach that does not require ground truth labels. We use LiDAR and IMU data to improve the accuracy and robustness of LiDAR odometry (LO) systems. Our contributions in this paper are highlighted as follows:

- We demonstrate that the use of temporal information in deep networks improves odometry estimation accuracy.
- We present a self-supervised convolutional network for odometry estimation with no ground truth poses required which can help to reduce the reliance on expensive ground truth data.
- Unlike previous learning-based approaches, our network uses a scene agnostic sliding window optimization with plane normal, 3D geometric and trajectory constraints to refine poses, eliminating the need for loop closure. This makes our approach more robust and adaptable to different environments.

Traditional methods: Conventional LiDAR odometry (LO) methods include Iterative Closest Point (ICP) [1] followed by feature-based approaches which have been the standard for real-time performance in the literature. Typically, these approaches iteratively find matching points via the nearest-neighbor searching method and optimize its rigid transformation i.e rotation and translation. However, due to the sparsity and irregularity of LiDAR point cloud data, establishing correspondences can be challenging. Over the decade, the performance of model-based LiDAR Odometry And Mapping (LOAM) [2] has dominated in several benchmark datasets and is the state-of-the-art in LiDAR odometry. Although LOAM achieves state-of-the-art results, it does not consider the dynamic objects present in the environment, which may cause performance degradation [3]. In order to increase the robustness and improve the performance, researchers exploit IMU measurements as supplemental information, hence extending LOAM to LeGO-LOAM [4].

Deep Learning based methods: The use of learning approaches can provide a robust estimation without discarding any information from data, as sampling points and tedious parameter tuning plagues the performance of the traditional approaches. In the works of [5], [6], the authors demonstrate the feasibility of using deep neural network for LiDAR odometry in autonomous driving application. However, these approaches are constrained to labeled data for training, making them impractical as annotating LiDAR data is time-consuming. As a result, [7] have incorporated self-supervised learning strategy using geometric losses. However, it did not outperform the existing learning methods. Recent approaches, such as Godard et al. [8], follow the same principle using image warping as part of reconstruction loss to create a self-supervision signal similar to our proposed network. With the existing approaches to learning-based odometry only capturing just-in-moment information, temporal information in the sequence of frames is, however unused. Motivated by the issues mentioned above, to capture spatio-temporal features, in this study we proposed a self-supervised recurrent convolutional odometry network that accepts LiDAR and IMU as input.

II. METHODOLOGY

The proposed approach learns a mapping function from unlabeled LiDAR scan sequences and raw IMU measurements to regress 6-DoF motion. This function predicts the pose as the system moves through the environment. The sliding window traverses to the next window when the number of inferred poses satisfies the window size parameter w. To ensure a fair comparison with mapping-based approaches, our method incorporates a backend optimization adapted from LOAM [2]. Fig. 1 provides an overview of our proposed method.

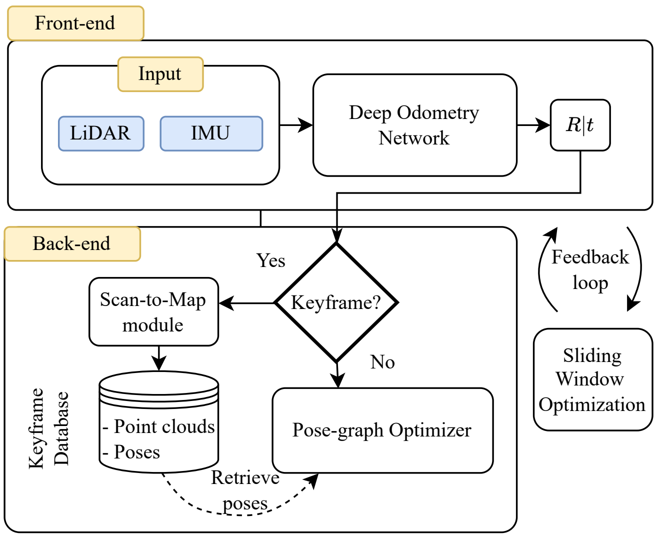

Fig. 1. System overview of the proposed approach

It is important to note that the mapping modules are only used in evaluating the system and were not included in the training of the network.

A. Data preprocessing

*a) **LiDAR Scan**:* This work proposes a pose estimator that infers 3D transformations without relying on ground truth data. It uses LiDAR point cloud scans from three time steps, S_{t-1}, S_t, S_{t+1}, for estimation. To reduce the sparsity of the point cloud data, a projection step is applied, projecting each point onto a cylindrical coordinate system to obtain a 2D image representation. This projection allows the utilization of Convolutional Neural Networks (CNNs) for extracting spatial and geometric features. The projection involves mapping from R^3 to $R^{H\times W}$ by discretizing azimuth and polar angles in spherical coordinates and keeping only the nearest points at each pixel's location. The projection function is defined using the equations:

$$\begin{aligned} \alpha &= \arctan(y/x)/\Delta\alpha \\ \beta &= \arcsin\left(z/\sqrt{x^2+y^2+z^2}\right)/\Delta\beta \end{aligned} \tag{1}$$

Here, α and β represent the indexes of the projected matrix, and $\Delta\alpha$ and $\Delta\beta$ denote the average angular resolution between consecutive beam emitters in the horizontal and vertical directions, respectively. The resulting matrix has dimensions $H \times W \times 4$, where H represents the height (number of vertical line scans) and W is the total number of points. The matrix includes the (x, y, z) coordinates of the points as well as their range r from the LiDAR sensor.

*b) **Inertial Measurement Unit**:* To process the IMU data for our deep neural network, we concatenate the measurement sequence into a single matrix. The raw IMU measurements are sampled and arranged as shown in Eqn. 2. Each row in the matrix represents the linear acceleration as a and angular velocity as v of the IMU at a given time stamp, and the number of samples is denoted by n. The resulting matrix, denoted by $\mathbf{M}$, has dimensions of $n \times 6$.

$$\mathbf{M} = \begin{bmatrix} a_0^0 & a_1^0 & a_2^0 & v_0^0 & v_1^0 & v_2^0 \\ a_0^1 & a_1^1 & a_2^1 & v_0^1 & v_1^1 & v_2^1 \\ \vdots & \dots & \vdots & \vdots & \dots & \vdots \\ a_0^n & \cdots & a_2^n & v_0^n & \cdots & v_2^n \end{bmatrix} \in \mathbb{R}^{n\times 6} \tag{2}$$

where a and v are linear acceleration and angular velocity, respectively. To further refine the preprocessing of the IMU data, we apply a low pass filter to remove high frequency noise and apply standardization to ensure that the data is centered around zero mean and has unit variance. By preprocessing the IMU data in this way, we can effectively use the temporal information to improve the accuracy of odometry estimation in our network.

B. Network architecture

We formulate our network as a sequential learning problem to leverage temporal dependencies, processing sequences of LiDAR scans and IMU data to estimate the 6 Degrees of Freedom (DoF) relative pose. The approach consists of three deep networks: Odometry network, IMU processing network, and Pose regression network. Importantly, our network operates solely on raw sensor data, eliminating the need for sensor calibration during inference.

*a) **Odometry network**:* Similar to other tasks involving deep network architecture designs, our architecture is built upon an existing backbone network. We have adopted ResNet32 [9] as the foundation for feature extraction, specifically to extract geometric features from LiDAR scans. The network generates a feature mapping of size (N, $(512)*t$, $\frac{H}{2}$, $\frac{W}{32}$), where t represents the consecutive scans between time steps. To obtain a single value for each channel, adaptive average pooling is applied along the height and width of the feature map. The resulting feature map is then combined with a feature map generated by the Inertial Measuring Unit (IMU) processing network in a sensor fusion module. Circular padding is applied throughout all convolutional layers to emulate the behavior of a true (imaginary) 360° circular image.

*b) **Inertial Measuring Unit processing network**:* Building on the success of using Convolutional Neural Networks (CNNs) for extracting features from concatenated IMU data, as demonstrated in [10], our IMU processing network incorporates a module with 5 Convolutional blocks and 2 skip connections inspired by the work of He et al. [9]. This module is applied to both the angular velocity and linear acceleration measurements from the IMU to extract features and model the data streams. Our implementation of the skip connections are added every two blocks of CNN, effectively merges low-level and high-level information, leading to improved network robustness and accuracy. The network parameters are summarized in Table I.

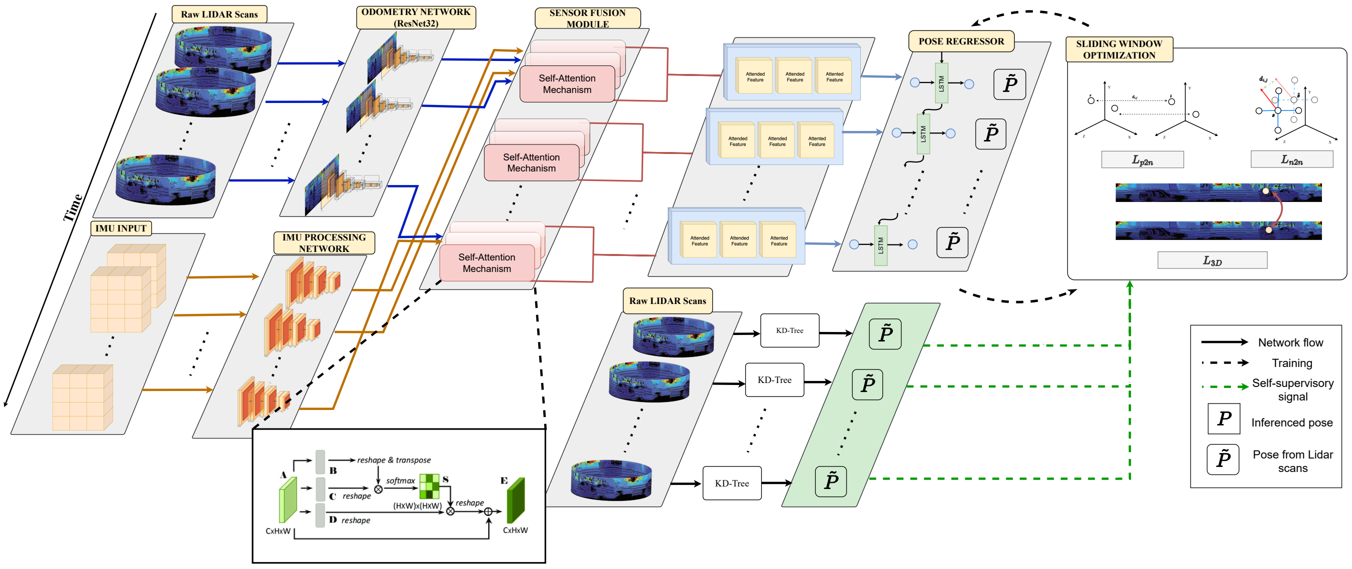

Fig. 2. Architecture overview: The proposed network pipeline takes 2D projected LiDAR scans and IMU measurements as inputs. There are two parallel feature extractor networks; the odometry network (ResNet32) and the inertial processing network, a sensor fusion module for intermediate feature fusion. Finally, feature outputs are propagated to a 2-layer LSTM for pose regression. A sliding window optimization step is applied during training for pose refinements.

TABLE I
NETWORK PARAMETERS FOR IMU PROCESSING NETWORK

Layer	Kernel size	Stride	Filter size	Skip-connection
(Conv-1) x 2	3 x 5	3	64	Conv-2
Conv-2	3 x 5	3	128	Conv-3
Conv-3	3 x 5	3	256	Conv-4
Conv-4	3 x 5	3	256	-

c) ***Sensor Fusion module:*** The use of traditional techniques like direct feature fusion, which combines intermediate features from multiple sensors, can be ineffective due to the presence of non-discriminative or indistinct features, resulting in diminishing returns [11]. This is particularly true in real-world settings, where difficulties in synchronizing and poor calibration between modalities are common. To overcome these challenges, we have proposed a module utilizing a self-attention mechanism [12] to selectively attend to relevant features and learn the most pertinent ones for feature mapping fusion. This approach allows the network to be robust to the intrinsic white noise distribution and bias of each sensor, and effectively deal with potential perturbations in performing downstream tasks. This method of feature mapping fusion is seen to yield better performance and robustness in our network over direct feature fusion.

In our system, we have implemented a time step of 3, i.e. $(t-1, t, t+1)$, to capture temporal dependencies in the feature matrices obtained from the Convolutional layers of the odometry and Inertial Measurement Unit processing network. These feature matrices are then passed through a sensor fusion module, which combines the information from multiple sensors and feeds it into a 2-layer Long Short Term Memory (LSTM) network. The LSTM layer is specifically designed to capture long-term dependencies and is used in our system to perform regression on both the translation and rotation estimates. By using the LSTM layer in this way, we are able to significantly improve the accuracy of our pose estimates and better handle the complex dynamic interactions between different sensors.

C. Training

1) Hyperparameter: The training set had 24,801 point cloud scans, with 2,791 scans reserved for testing. To optimize our network, we used batch normalization, data augmentation, and Stochastic Gradient Descent. We trained the network with three consecutive 2D projected point cloud scans and IMU matrix M, using a batch size of 32 for 90,000 iterations. The learning rate was set to $1e^{-3}$ with a momentum of 0.9. Progress was monitored by evaluating the validation error every 1,000 iterations.

To train the network, we follow a two-step process: pre-training and full network training.

2) Pre-training: During the pre-training phase, we adopt the training protocol outlined in [13] for the Odometry Network branch. This involves utilizing fully connected layers, with one dedicated to estimating rotation and another for translation. For a comprehensive understanding of the pre-training protocol, we refer readers to the original paper.

3) Full Network Training: After the pre-training of the Odometry network branch, the entire network enters the training phase. At this point, the Odometry network is frozen, and only the IMU network, Sensor fusion module and Pose regressor network continues to train. This approach ensures that the learned features from the pre-training phase are effectively utilized during the subsequent training process. The full network is trained using the sliding window optimization, which will be explained in the following subsections.

4) Sliding window optimization: Our approach incorporates a sliding window optimization technique, which draws inspiration from the cost functions employed in [10], [14]. For a detailed understanding of these techniques, we encourage readers to refer to the original papers. In the subsequent section, we will specify the modifications we have made in our work.

Intra-window optimization: The LiDAR-Inertial stream $\langle S_0, M_0\rangle, \ldots, \langle S_{n-1}, M_{n-1}\rangle$ of each sliding window is utilized for geometric inference generation and intra-window optimization. The middle frame of the window is taken as the target view, while others are source views. While the normals n for each scan are precomputed offline, the $L_{normals}$ loss includes two components, L_{p2n} and L_{n2n}.

a) Point-to-Plane Loss: For each point s in the transformed source scan S_t, its distance to the corresponding point $\hat{s}$ in the target scan S_{t+1} is computed and projected onto the surface at that position. The point-to-plane loss function, denoted as L_{p2n}, measures the accuracy of the transformation between consecutive point cloud scans. It calculates the difference between each point s in S_t and its corresponding point $\hat{s}$ in S_{t+1} by projecting this distance onto the surface at that position. The point-to-plane loss function is defined as:

$$L_{\mathrm{p2n}} = \frac{1}{N}\sum_{i=1}^{N} |(\hat{s}_i - s_i)\cdot \hat{n}i|\, 2^2 \tag{3}$$

Here, $\hat{n}_i$ represents the target normal vector. If either the source or target point la

b) Plane-to-Plane loss : The second loss compares the surface orientation around two points by calculating the difference between the normal vectors at the transformed source location and the normal vectors at the target location. The normal vector at the transformed source location is represented by n_i and the normal vector at the target location is represented by $\hat{n_i}$.

$$L_{n2n} = \frac{1}{N}\sum_{i=1}^{N} |\hat{n_i} - n_i|_2^2 \tag{4}$$

The total plane normal loss is then calculated by adding the point-to-plane loss, L_{p2n}, and the normal vector loss, L_{n2n}. This total loss, represented by L_{normal}, is used to optimize the performance of the network.

$$L_{normal} = L_{p2n} + L_{n2n} \tag{5}$$

Inter window optimization: As it is prone to fall into a local optimum by only relying on the optimization within windowed frames, due to the lack of sequential constraint that may cause the universal scale ambiguity and the accumulated error problems in monocular odometry. We consider the inter-window optimization, including trajectory consistency check and 3D geometric consistency check. The point cloud scan of the target view and estimated transformation matrix $\langle T_i^{i+1}, \ldots, T_{i+w-1}^{i+w-2}\rangle$ between adjacent frames within the window, are checked by following loss consistency:

c) 3D geometric consistency loss: To define the 3D geometric consistency loss, L_{3D}, we use a measure of the accuracy of the transformation matrix, T_t^{t+1}, between consecutive scans. This loss is optimized during training to improve the accuracy of the transformation matrix and the overall performance of the network. The 3D geometric consistency loss is expressed as:

$$L_{3D} = \sum_{i=1}^{N} \frac{|S_{i+1} - T_i^{i+1} S_i|}{S_{i+1} + T_i^{i+1} S_i} \tag{6}$$

where N is the number of samples found from the correspondences search.

As relying on inter-window optimization are prone to local minima [10], posses is exploited to estimate the prior trajectory for the entire sequence. The estimated poses that aggregated from the windowed estimation and the corresponding p that estimated from the integrated sequential information are checked for the trajectory consistency. The estimated poses p that aggregated from the windowed estimation and the corresponding $\tilde{p}$ that estimated from the integrated sequential information are checked for the trajectory consistency by:

$$L_{\text{traj}} = \sum_{i=1}^{N-1} |\hat{p}_i^{i+1} - p_i^{i+1}| \tag{7}$$

To summarize, the loss functions are computed as.

$$L = L_{normal} + \lambda L_{3D} + L_{traj} \tag{8}$$

To balance the convergence of the point-to-plane loss, L_{p2n}, and the normal vector loss, L_{n2n}, we use the hyperparameter λ to assign more weight to the 3D geometric consistency loss, L_{3D}, as it significantly affects the performance [11] of the network. In our experiments, we set λ to 1.2.

III. Experiment results

A. KITTI odometry dataset

In order to evaluate the performance of our proposed LiDAR-Inertial sensor configuration odometry network, we conducted experiments on the widely-used KITTI odometry dataset. The training set consisted of Sequences $00-08$, while Sequences $09-10$ were used as the test set. This split is consistent with previous works such as LOAM [2], LeGO-LOAM [4], SUMA [15], DeLORA [5], LO-Net [14], DeepLO [16], and UnDeepLIO [7].

a) Evaluation metrics: We evaluated our method using the KITTI odometry benchmark [17], which measures the accuracy of the estimated trajectory with respect to the ground truth. The benchmark defines the relative translation error (t_{rel}) and relative rotation error (r_{rel}) as the root mean square error (RMSE) of the differences between the estimated and ground truth trajectories, normalized by the length of the ground truth trajectory. We report the t_{rel} and r_{rel} errors for our method on the KITTI dataset, and compare them with state-of-the-art methods in the literature. We evaluated the benchmark methods with only LiDAR

TABLE II

PERFORMANCE COMPARISON WITH AND WITHOUT INERTIAL MEASUREMENT UNIT (IMU) ON KITTI BENCHMARK DATASET WITH NO MAPPING

	00		01		02		03		04		05		06		07		08		09		10	
	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}
LeGO-LOAM [4]	1.44	0.65	21.12	2.17	2.69	0.99	1.73	0.99	1.70	0.69	0.98	0.47	0.87	0.45	0.77	0.51	1.35	0.58	1.46	0.64	1.84	0.74
LeGO-LOAM + IMU	7.24	2.44	20.07	**0.61**	2.56	0.91	x	x	1.68	**0.59**	**0.82**	**0.38**	**0.86**	**0.43**	**0.67**	**0.38**	1.29	**0.53**	1.49	0.58	1.75	**0.63**
UnDeepLIO [7]	1.33	0.91	3.40	1.09	1.53	1.19	1.43	1.42	1.26	0.61	1.22	0.78	1.19	0.64	0.97	4.56	1.92	2.86	3.87	2.34	2.69	2.89
UnDeepLIO + IMU	1.38	**0.62**	3.46	0.98	1.42	0.67	x	x	0.98	0.67	1.26	0.64	0.94	0.56	3.45	2.17	4.05	1.63	2.77	1.25	2.16	1.11
Ours (LiDAR)	0.97	0.78	1.71	0.78	1.39	0.81	0.93	0.82	0.81	0.87	1.06	0.72	1.71	0.72	1.53	0.75	0.95	0.72	1.44	0.68	1.49	0.85
Ours (LiDAR+IMU)	**0.81**	0.69	**1.33**	0.81	**1.02**	**0.62**	**0.45**	**0.77**	**0.76**	0.82	0.93	0.59	0.95	0.70	1.23	0.62	**0.72**	0.55	**1.20**	**0.50**	**1.24**	0.80

Note: The table includes only published results, except for the entry marked as 'x'. It is worth noting that LeGO-LOAM is the only traditional approach presented in the table, and UnDeepLIO does not incorporate temporal features. The best results are highlighted in bold.

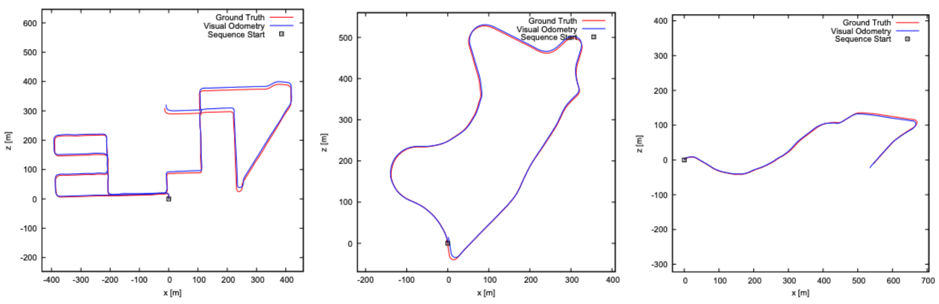

Fig. 3. Sample trajectory results on KITTI dataset for the proposed network(bottom) on Seq. 08-10. Both has similar performances and achieves very good precision and the differences from ground truth (red) are barely visible.

point cloud scans to show the impact when the inertial measurement unit is disabled (see Table II).

Table II presents a comparison of our method's performance with other state-of-the-art approaches on the KITTI benchmark dataset, in terms of translation (t_{rel}) and rotation (r_{rel}) errors. Our method outperforms LeGO-LOAM and UnDeepLIO in all tested sequences, particularly in Sequences 01, 02, and 04. When the IMU is included in LeGO-LOAM and UnDeepLIO, a decrease in performance can be observed, especially in Sequence 03, likely due to the noise and bias present in the IMU data [17]. Our method, on the other hand, effectively utilizes the IMU data to improve overall performance, resulting in lower translation and rotation errors. These results demonstrate the effectiveness of our proposed LiDAR-Inertial based odometry estimation system. It is worth noting that our deep learning approach compares favorably to the traditional method of LeGO-LOAM and the learning-based method of UnDeepLIO. While LeGO-LOAM relies on hand-crafted features and geometric constraints, and UnDeepLIO utilizes a deep neural network for feature extraction, our method directly learns the mapping between the IMU and LiDAR data through end-to-end training, allowing for more flexible and accurate modeling. This superior performance of our deep learning approach highlights the benefits of using this method for IMU-based odometry estimation.

Table III provides a comprehensive assessment of LiDAR-Inertial odometry systems, categorized by the presence or absence of mapping functionalities. Among the learning-based algorithms listed, all except ours do not utilize temporal features, while LOAM stands as the only classical method in the table. The approaches are evaluated on all possible sequences of lengths ranging from 100 to 800 meters. The translational error was measured in percentage, and the rotational error was measured in degrees per 100 meters. Among the approaches that do not employ mapping, our approach performs relatively well, with a translational error of 1.50% and a rotational error of 1.23 degrees per 100 meters on the training Sequences 00-08. On Sequence 09, our approach performed even better, with a translational error of 0.80% and a rotational error of 0.58 degrees per 100 meters. On Sequence 10, our approach achieved a translational error of 1.20% and a rotational error of 0.93 degrees per 100 meters. The higher errors in Sequence 10 can be attributed to the sharp motions present in the trajectory, as depicted in the most right column of Fig. 3. These motions present a challenge for neural network-based approaches in accurately predicting the odometry.

Our approach demonstrates strong performance across different scenarios, including those involving mapping. In the training Sequences 00-08, we achieved a translational error of 0.64% and a rotational error of 0.58 degrees per 100 meters. These results were consistently maintained in Sequence 10, where our approach exhibited a similar level of

TABLE III

COMPARISON OF TRANSLATIONAL (%) AND ROTATIONAL ($\frac{deg}{100m}$] ERRORS ON ALL POSSIBLE SEQUENCES OF LENGTHS OF $(100, 200, ..., 800)$ METERS FOR THE KITTI ODOMETRY BENCHMARK ON LIDAR-INERTIAL ODOMETRY.

		Training Seq. 00-08		Sequence 09		Sequence 10	
Mapping		t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}
No	DeepLO [16]	3.68	0.87	4.87	1.95	5.02	1.83
	UnDeepLIO [7]	2.42	1.13	2.77	1.25	2.16	1.11
	LO-Net [14]	**1.27**	**0.67**	1.37	**0.58**	1.80	**0.93**
	DeLORA [5]	3.00	1.38	6.05	2.15	6.44	3.00
	Ours	1.50	1.23	**0.80**	**0.58**	**1.20**	**0.93**
Yes	LOAM [2]	1.26	0.50	1.20	0.48	1.51	0.57
	SUMA [15]	3.06	0.89	1.90	0.80	1.80	1.00
	LO-Net [14]	0.81	**0.44**	0.77	**0.38**	0.92	**0.41**
	DeLORA [5]	1.78	0.73	1.54	0.68	1.78	0.69
	Ours	**0.64**	0.58	**0.68**	0.42	**0.72**	0.55

Note: Table includes LOAM as the only classical method and several learning-based methods, with our method being the only one incorporating temporal features. Best results are highlighted in bold.

accuracy, with a translational error of 0.72% and a rotational error of 0.55 degrees per 100 meters. Notably, our approach outperformed other methods in the test sequences, except for the rotational mapping, where LOAM showed better performance. Overall, our approach performed competitively with other methods on the KITTI odometry benchmark for LiDAR-Inertial odometry, delivering relatively low translational and rotational errors. This indicates its effectiveness, especially in sequences without mapping.

With the mapping modules disabled, our method performed comparably to Li et al. [14] and Nubert et al. [5]. Figure 3 shows qualitative results of our predicted odometry trajectories. Despite being trained only on sequence $00-08$, our method achieved similar performance to the conventional model-based approach. It produced accurate odometry estimates with minimal drift, even on challenging sequences with dynamic objects (Sequence 09 and 10). Incorporating mapping refinement modules further improved performance. Quantitatively, our proposed network matched the performance of a comparable learning-based LiDAR odometry approach [14], and surpassed it when used without mapping.

IV. Conclusions

We evaluate a deep learning approach for LiDAR-Inertial odometry that incorporates temporal information and outperforms existing learning-based methods while achieving comparable accuracy to traditional model-based methods. Our self-supervised training strategy employs sliding window optimization to ensure 3D geometric and trajectory consistency. Our approach has the potential to serve as a complementary method to traditional localization estimation techniques and effectively interpret the environment as detected by a multi-modal sensor configuration. While memory and resource requirements are currently a limitation, we plan to investigate lightweight network architecture designs for low-resource constraint devices in future work.

References

[1] P. Besl and N. D. McKay, "A method for registration of 3-d shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239–256, 1992.

[2] J. Zhang and S. Singh, "Loam: Lidar odometry and mapping in real-time," in *Proceedings of Robotics: Science and Systems (RSS '14)*, July 2014.

[3] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. M. Montiel, and J. D. Tardós, "ORB-SLAM3: an accurate open-source library for visual, visual-inertial and multi-map SLAM," *CoRR*, vol. abs/2007.11898, 2020.

[4] T. Shan and B. Englot, "Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4758–4765, 2018.

[5] J. Nubert, S. Khattak, and M. Hutter, "Self-supervised learning of lidar odometry for robotic applications," *CoRR*, vol. abs/2011.05418, 2020.

[6] M. Velas, M. Spanel, M. Hradis, and A. Herout, "CNN for IMU assisted odometry estimation using velodyne lidar," *CoRR*, vol. abs/1712.06352, 2017.

[7] Y. Tu and J. Xie, "Undeeplio: Unsupervised deep lidar-inertial odometry," *CoRR*, vol. abs/2109.01533, 2021.

[8] H. Zhan, R. Garg, C. S. Weerasekera, K. Li, H. Agarwal, and I. D. Reid, "Unsupervised learning of monocular depth estimation and visual odometry with deep feature reconstruction," *CoRR*, vol. abs/1803.03893, 2018.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

[10] P. Wei, G. Hua, W. Huang, F. Meng, and H. Liu, "Unsupervised monocular visual-inertial odometry network," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20* (C. Bessiere, ed.), pp. 2347–2354, International Joint Conferences on Artificial Intelligence Organization, 7 2020. Main track.

[11] Y. Almalioglu, M. Turan, A. E. Sari, M. R. U. Saputra, P. P. B. de Gusmão, A. Markham, and N. Trigoni, "Selfvio: Self-supervised deep monocular visual-inertial odometry and depth estimation," *CoRR*, vol. abs/1911.09968, 2019.

[12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017.

[13] M. Valente, C. Joly, and A. de La Fortelle, "An lstm network for real-time odometry estimation," in *2019 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1434–1440, IEEE, 2019.

[14] Q. Li, S. Chen, C. Wang, X. Li, C. Wen, M. Cheng, and J. Li, "Lo-net: Deep real-time lidar odometry," *CoRR*, vol. abs/1904.08242, 2019.

[15] J. Behley and C. Stachniss, "Efficient surfel-based slam using 3d laser range data in urban environments," in *Proc. of Robotics: Science and Systems (RSS)*, 2018.

[16] Y. Cho, G. Kim, and A. Kim, "Deeplo: Geometry-aware deep lidar odometry," *CoRR*, vol. abs/1902.10562, 2019.

[17] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.