2022 11th International Conference on Control, Automation and Information Sciences (ICCAIS) | 978-1-6654-5248-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICCAIS56082.2022.9990287

Real-time ROS Implementation of Conventional Feature-based and Deep-learning-based Monocular Visual Odometry for UAV

A. M. Nguyen*, D. T. Nguyen*, V. Q. Pham*, H. T. Nguyen*, D. T. Tran†, J.-H. Lee † and A. Q. Nguyen *

* School of Electrical and Electronic Engineering, Hanoi University of Science and Technology, Vietnam
† College of Information Science and Engineering, Ritsumeikan University, Kyoto, Japan
E-mail:tuan@fc.ritsumei.ac.jp,quang.nguyenanh@hust.edu.vn

***Abstract*—Localization or state estimation is one of the most important tasks for UAVs based on different kinds of sensors such as GPS, IMU, Lidar or cameras. However, localization based on only a monocular camera or visual odometry is one of the most challenging research topics. Conventional methods are proposed based on the detection of key features in each image and matching them on consecutive images to estimate the camera motions. Deep-learning methods have also been studied to solve the problem. Although the current learning-based visual odometry methods score high results on public datasets, there is a lack of real-time implementation of the methods in common robot operating systems such as ROS to integrate them into a navigation system. In this paper, we introduce a ROS implementation of state-of-the-art conventional feature-based method, ORB-SLAM3, together with a deep-learning-based method, SC-SfMLearner for real-time UAV localization. A photo-realistic simulator, Flightmare, is used to test the implementation together with another navigation task such as control. The implementation can evaluate both algorithms in real-time operation to compare their performances. Based on evaluation results from the simulated environments, the limitation or failure cases of the algorithms could be found, then, the best parameters of the algorithms can be adjusted to improve the algorithms to avoid failures in practical experiments.**

***Index Terms*—Visual Odometry, Deep Learning, ROS implementation, Flightmare, ORB-SLAM3**

I. Introduction

Autonomous UAVs have been widely studied for years, having great potential applications such as delivery, search and rescue, tracking and mapping and exploration. In autonomous UAV systems, localization or state estimation, which estimates the position and orientation of the UAV, is one of the most important and hard algorithms as they are essential inputs for other algorithms. Therefore, fast and accurate estimation algorithms have been proposed based on various types of sensors such as GPS, Inertial Measurement Unit (IMU), Lidar, cameras, and so on. Since IMU estimator faces large drift and noise issues; Lidar requires high computational ability as well as high cost meanwhile GPS has low accuracy, especially in indoor environments or outdoor environments with low altitudes. Therefore, cameras with low cost and abundant data information have been widely used for the localization of autonomous UAVs. Localization with cameras or **Visual Odometry, (VO)** is the process of determining the position and orientation of a robot by analyzing the associated camera images. The main goal of VO is to calculate transformation matrix $T_{t-1 \to t}$ from t^{th} image I_t to the previous image I_{t-1} and then integrate all the transformations to restore the entire path $C_{0:k}$ of the camera. Information extracted from the images could be the feature points (corners, edges, blocks) [1] or optical flow (pixel's gradient) [2] or the combination of both methods. Learning-based VO (or Deep learning-based VO) has become a novel approach in the field of Vision Robotics recently, making use of deep neural networks to capture non-linear mappings between information from images and the 3D spatial transformation matrix of camera motion. Some deep-learning-based VO methods are introduced such as SfMLearner [3] and its improved version SC-SfMLearner [4] which utilizes the inverse wrap process, which will be explained later, to estimate depth and ego-motion between target and source images, learning via minimizing photometric losses. The aforementioned attempts already achieved state-of-the-art performance on popular VO benchmark datasets such as KITTI [5], EuRoC [6] and CityScapes [7]. However, there is a lack of real-time implementation results when integrating the trained pose estimation model into a real-time navigation system, making it hard to compare these learning-based methods with the conventional feature-based methods in a thorough way.

In this paper, we present a real-time implementation of both the state-of-the-art conventional feature-based, ORB-SLAM3 [8], and deep-learning-based, SC-SfMLearner [4], monocular visual odometry to make a comprehensive comparison of both methods using simulated scenarios. The photo-realistic UAV simulator is the well-known Flightmare [9], which is not only fast in collecting and computing a large number of images, but also integrate an accurate UAV dynamic developed by ETH Zurich [10] on ROS platform. With Flightmare, a simulation environment can be created based on real images by using Rendering Engine (in Unity environment). Therefore, the discrepancy between simulation and real-world camera views

Corresponding Author: Anh Quang Nguyen

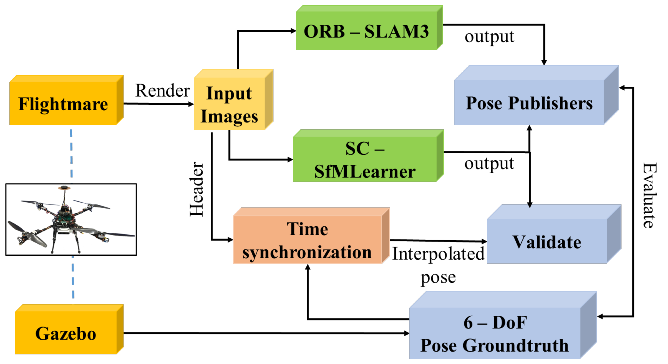


Fig. 1. Real-time ROS Implementation to compare conventional feature-based method and deep-learning method for monocular visual odometry based on a photo-realistic-rendering simulator.

can be minimized. The overview of our work is described in Fig. 1. The 6-DoF Pose estimation is real-time estimated by both ORB-SLAM3 and SC-SfMLearner using the same monocular input images from the Flightmare simulator. We attach ROS publishers on top of the original algorithms, making them suitable to be deployed in real-time situations. The pose messages are compared against the groundtruth provided by the simulator to evaluate their performances.

Based on evaluation results from the simulated environments, the limitation or failure cases of the algorithms could be found, then, the best parameters of the algorithms can be adjusted to improve the algorithms to avoid failures in practical experiments.

The remaining parts of the paper are organized as follows. Section II explains state-of-the-art conventional feature-based and deep-learning-based monocular visual odometry algorithms which are used in this paper. Section III describes our Implementation and Experiments, including data preparation, training, and ROS integration. Section IV shows comparison results of two pose publishers on different hardware specifications. Section V is our conclusion of the paper.

II. Conventional Feature-Based and Deep-Learning-Based Monocular Visual Odometry

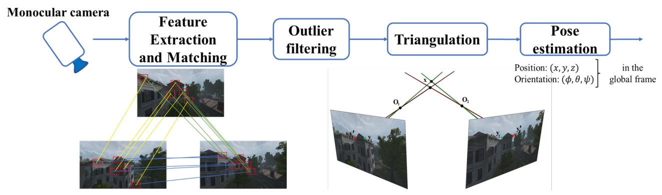


Fig. 2. Algorithms for Featured-based Monocular visual odometry to estimate the pose of the UAV based on consecutive captured images

A. ORB-SLAM3 [8] - a state-of-the-art conventional feature-based monocular VO

Conventional methods of monocular VO implementation can be mainly divided into 2 groups: feature-based methods [1], the direct tracking methods [11]. Most feature-based VO algorithms, including ORB-SLAM3, were implemented by following procedures as shown in Fig. 2.

Sensor: Camera is the only sensor in monocular VO. It is used to acquire and collect consecutive images that have scenes overlapping with each other. Each camera has a specific set of intrinsic parameters, represented as a 3x3 intrinsic calibration matrix **K** contains the intrinsic parameter: f_x and f_y are the focal lengths in the x and y directions, and c_x and c_y are the 2D coordinates of the projection center or as a pair of Field of View angles (FoV).

$$\mathbf{K} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Feature Extraction and Matching: To estimate the relative pose between images, feature points x_i are extracted using various methods. SIFT and SURF could bring good extraction results but they cost a lot of computational resources. Therefore, it is hard to use them in real-time applications. The ORB combines the advantages of Features from Accelerated Segment Test (FAST) [12] and Binary Robust Independent Elementary Features (BRIEF) [13], therefore it can achieve better-extracted features in terms of scale, rotation, and brightness. Moreover, this combination is very efficient, which reduces a lot of computational time. Therefore, ORB is much more fit into the real-time scheme. The extracted features from different views of images are matched together to create pairs of correspondences for triangulation. Matching is performed using the DBoW2 bag of word library.

Outlier filtering and Triangulation: Outliers (or false matches) are defined as the incorrectly matched feature points between two frames. To reduce the number of outliers in a set of observed data, the Random Sampling Consensus (RANSAC) algorithm [14] is the most common method. Based on feature correspondences extracted and matched from previous steps, RANSAC utilizes 5-point or 8-point algorithm to triangulate and computes best possible fundamental matrix **F** that can filter out the most number of outliers.

Pose Estimation: Having taken the **F** and **K** from the previous step, the essential matrix **E** is derived to capture the geometric relationship between a pair of images, where:

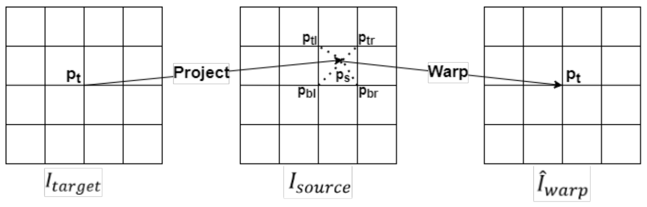
$$\mathbf{E} = \mathbf{K}^T\mathbf{F}\mathbf{K} \tag{1}$$

Taking singular value decomposition (SVD) of **E**, we can estimate rotation and translation between 2 image planes

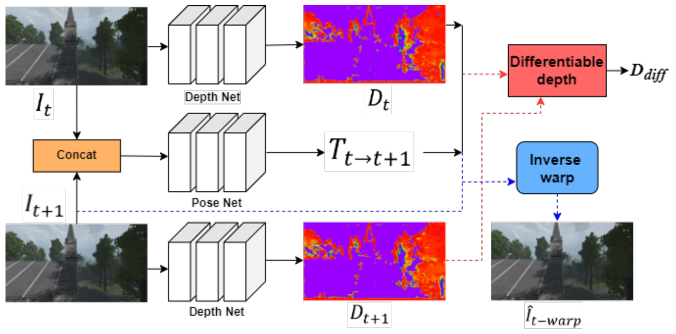
$$\mathbf{E} = [t]_\times \mathbf{R} \tag{2}$$

where **R** is the rotation matrix, **t** is the translational vector given by: $t = [t_x, t_y, t_z]^T$ and $[t]_\times$ is the skew symmetric matrix given by:

$$[\mathbf{t}]_\times = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & t_y \\ -t_y & t_x & 0 \end{bmatrix}$$



(a) Inverse warp



(b) SC-SfMLearner

Fig. 3. Illustration of the self-supervised learning-based methods for depth and ego-motion estimation. (a) Inverse warping process. First, use the predicted depth and pose to project each point p_t in the I_{target} onto I_{source}. Then, use bilinear interpolation to get a warped point corresponding to p_t . (b) SC-SfMLearner: the depth network takes two consecutive images (T_t, T_{t+1}) in the subsequence as input, and outputs their depth map (D_t, D_{t+1}). To get relative pose, first concatenate the two consecutive images; then, pass it through pose network. The predicted depth maps and relative poses of both methods are then used to reconstruct the target view based on novel view synthesis.

B. SC-SfMLearner

The idea of applying novel view synthesis for solving camera pose estimation is heavily inspired by direct VO methods. and could be expressed as follows: For each point in 3D camera frame $X_C = [X, Y, D]^T$, we have a projection on 2D image plane $x_{im} = [x, y]^T$. Therefore, in order to recover a 3D point from its corresponding 2D projection on the image frame, given the estimated depth value $\widehat{D}$, we can perform the inverse transformation:

$$X_C = \widehat{D}\mathbf{K}^{-1}x_{im} \tag{3}$$

With the estimated camera rotation $\widehat{\mathbf{R}}$ and translation $\widehat{t}$, we can find the 2D coordinates of a target x_{im} onto a novel image plane as

$$D^{'}x^{'}_{im} = \mathbf{K}\widehat{\mathbf{R}}\mathbf{K}^{-1}\widehat{D}x_{im} + \mathbf{K}\widehat{t} \tag{4}$$

Replicating this transformation for all pixels in the original image frame, we can synthesize a new 2D image, with all pixels moving to new 2D coordinates acquired from (4). Since these new coordinate values are continuous, the new intensity value will be calculated with respect to neighboring pixels in a novel view, using bilinear interpolation. The whole process is called **inverse warp**. The goal of Direct VO is to estimate $\widehat{\mathbf{R}}$, $\widehat{t}$ by minimizing the photometric reconstruction loss between the original target image and its synthesized counterpart after being wrapped.

$$L = \frac{1}{HW}\sum_{i=1}^{HW}|I_{target}(i) - I_{warp}(i)| \tag{5}$$

To minimize (5), Gauss-Newton or Levenberge-Marquardt algorithms could be applied. For self-supervised monocular VO, there are two neural networks are used: a Depth Net to estimate the 2D Depth plane from the monocular target image and a Pose Net to regress 6 DoF ego-motion from the target frame to neighboring frames as shown in Fig. 1. During forwarding pass, the estimated $\widehat{\mathbf{R}}$, $\widehat{t}$, $\widehat{D}$ are then used for inverse wrap process to calculate differentiable loss function in 5 which will be optimized for every step of backpropagation.

One of the first attempts to implement the aforementioned idea is SfMLearner [3]. However, SfMLearner is vulnerable to dynamic scenes, its output frequently suffers from both scale consistency and scale ambiguity. Therefore, an improved version of SfM, named SC-SfMLearner [4], proposed a geometry consistency loss [4] to solve the scale-inconsistency since scale ambiguity cannot be solved in monocular visual odometry. SC-SfMLearner [4] method is based on the task of novel view synthesis, in which output depth maps and relative poses are learned via optimization of photometric reconstruction loss between synthesized image and original target one. The whole process of SC-SfMLearner is shown in Fig.3. For pose estimation, the pose network takes two consecutive images that are concatenated and outputs the relative pose between them $(T_{t\to t+1})$.

III. Implementation and Experiments

A. Data Acquisition on Flightmare

Flightmare provides us not only an extensive source of photo-realistic image data to train deep learning neural networks, but also a collection of accurate and continuous streams of ground truth poses used for validating the performances of VO algorithms. We gather a total of 7 trajectories across 2 simulated environments in Flightmare: Forest and Industry. Some sample images from the aforementioned environments could be seen in Fig. 4. It is obvious to see the difference between two environments from the number of extracted features in each environment. For each trajectory, we use Flightmare ROS Wrapper [9] to extract ROS topics of RGB, Depth Image and groundtruth poses, saving them into ROSBAG format. We use body rate controller and thrust-mixing scheme developed in [15] to fly the UAV along different trajectories. The intrinsic camera matrix $\mathbf{K}$ are in-directly calculated from FoV angles and the resolution of training images, using the following formula:

$$f_x = f_y = \frac{h}{2\tan\frac{FoV}{2}}, c_x = \frac{w}{2}, c_y = \frac{h}{2} \tag{6}$$

where w, h is the width and height of the images.

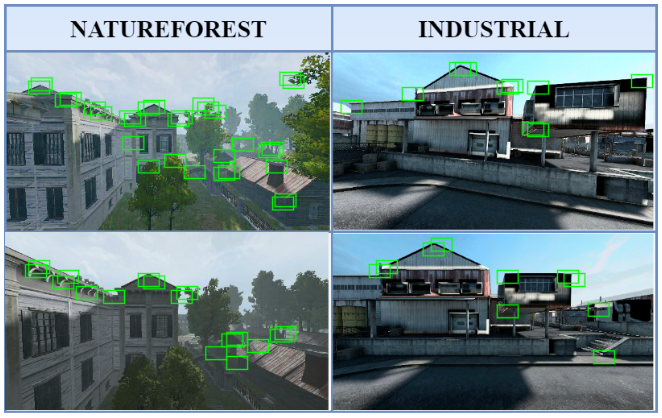


Fig. 4. First, we travel around the whole Natureforest map using a simulated UAV, collecting a total of 5 trajectories with different shapes and lengths. RGB scenes from this environment contain a rich and diverse source of features, making it an ideal source of training materials. For the Industrial environment, we want to create a decent validation set that can evaluate both the robustness of Monocular ORB-SLAM3 and the generalizability of 2 deep learning models across an entirely different scene.

TABLE I. Natureforest and Industrial

Table	Natureforest	Industrial
Number of trajectories	5	2
Frame rate (Hz)	10.5	24.3
Number of images	9426	6500
Image resolution (pixel by pixel)	360x640	
Camera FoV (degree)	90	
Groundtruth rate (Hz)	100	

B. Training SC-SfMLearner [4]

1) Training/Validate Dataset Preparation: With each simulation, we converted the RGB images stream into 15926 individual images that would be fed into the DataLoader of SC-SfMLearner Pytorch codebases. This conversion is done via CV Bridge, taking each received ROS Image.msg and outputting its numpy array counterparts.

As can be seen from Table I, the publish rate of groundtruth poses is much higher than the frame rate of RGB topics, which means that during a particular interval of time, the number of poses recorded is always greater than that of RGB images. To validate the output of SC-SfMLearner against the groundtruth, the precise groundtruth poses corresponding to each captured image are required, which is a challenging task. Therefore, we perform time synchronization between the image topic and the ground truth topic, assigning each of the image messages with a single pose at the same timestamp, or with the interpolation of two neighbor poses residing at the two nearest timestamps as shown in Fig. 6a. The detail of the time synchronization algorithm is described as follows:

- Step 1: Assuming that we have a vector $V = [v_0, v_1, ..., v_M]^T$ of length M containing all timestamps of every image message, and a vector $P = [p_0, p_1, ..., p_N]$ of length N corresponding to N timestamps of N poses collected from Flightmare. The groundtruth pose vector is denoted as $G_p = [g_{p_0}, g_{p_1}, ..., g_{p_N}]^T$. Normally $N \gg M$ due to high pose publishing rate.
- Step 2: We construct an M by N time difference matrix **U** via broadcasting V against P :

$$\mathbf{U} = \begin{bmatrix} v_0 - p_0 & v_0 - p_1 & \dots & v_0 - p_N \\ v_1 - p_0 & v_1 - p_1 & \dots & v_1 - p_N \\ \dots & \dots & \dots & \dots \\ v_M - p_0 & v_M - p_1 & \dots & v_M - p_N \end{bmatrix}$$

- Step 3: We find the minimum positive and minimum negative values of each row in **U**, their indices form a M by 2 *mask* that points to the nearest neighbor poses of each entry in the original G_p.
- Step 4: The *mask* is used to recover the neighbor poses from G_p and their corresponding timestamps from P, creating also M by 2 matrices.
- Step 5: From neighbor poses and their timestamps collected from Step 4, we can use linear interpolation to create a vector of length M which will be assigned to M original RGB images.

The whole synchronization algorithm is fully vectorized, making it suitable to be integrated to Pytorch Dataset class. Fig 5 illustrates one of the trajectory in our dataset, with both original poses from Flightmare and interpolated ones are visualized against each others.

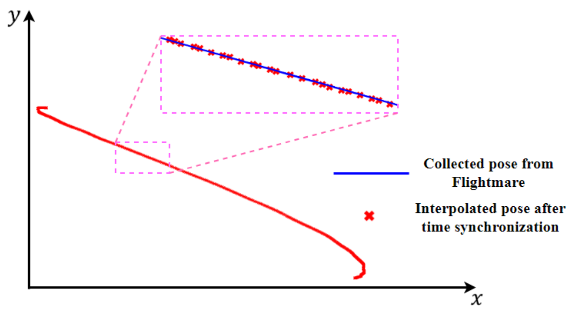
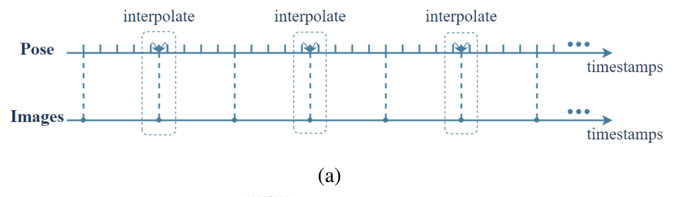


Fig. 5. Visualization of interpolated pose against the original once. The pose will be used to validate the learning algorithms. Due to Flightmare being not able provide evenly distributed messages, we have to perform interpolation on every instance of the collected image.

2) Training Configuration: The whole training process is implemented on the original codebases of SC-SfMLearner which is written in PyTorch. Each sample loaded into the pose networks is a stacked sequence of 5 consecutive images, with the center image acting as the target frame and the other 4 as source frames. Adam optimizer [16] is used with the learning rate of 10^{-4} to train both depth and pose networks for a total of 100 epochs. Both networks are built upon Resnet50 [17] backbones, being trained and evaluated with a single GPU Nvidia GTX 1080Ti (11GB DDR5X Vram, 3584 CUDA cores) on Ubuntu 20.04.

C. ROS Implementation

To analyze the real-time performance of the algorithms, it is necessary that we integrate them into ROS. We created two ROS nodes, one for ORB-SLAM3 Monocular Tracker, and one



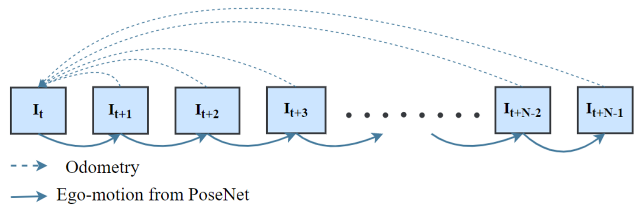


Fig. 6. a) Illustration of groundtruth time synchronization process. b) Ego-motion from the PoseNet of SC-SfMLearner is converted to camera odometry.

for SC-SfMLearner pose network, that subscribes to the RGB images topics from the Flightmare simulator and publishes the calculated odometry.

1) ORB-SLAM3 Monocular Tracking ROS: For ORB-SLAM3, we examine the estimated pose by utilizing the tracking module of the original code base. We build our own odometry publisher on top of the original monocular tracker. It is also worth mentioning that output from the tracker represents the pose in the camera frame; therefore, we multiply it with the transformation matrix $T_{b \to c}$ between the UAV body and the camera, before adding a translation component to move the predicted pose into world frame. The whole calculation is done via Sophus, and the final result is then converted into PoseStamped ROS message format.

2) SC-SfMLearner ROS: As SC-SfMLearner takes 2 images for each iteration of inference, a queue structure is implemented that push-backs new data each time an image message is received and pop-fronts the image from previous timestamps after a pose message is published. The PyTorch module only executes the forward pass whenever the queue length is equal to 2, outputting 6 DoF ego-motion tensor between the two currently queued RGB inputs. The estimated ego-motion is then multiplied with the camera pose from the last execution to become the odometry, before being converted into a world pose and being published as a PoseStamped message as shown in Fig. 6b. We use GPU to accelerate the inference speed of the neural net module.

IV. Evaluation Results

RPG evaluation tool [18] is used to compare the performance of two pose estimators from the above self-written ROS nodes. We evaluate both ORB-SLAM3 and SC-SfMLearner on the aforementioned trajectories recorded from Flightmare. The evaluation process for each of the trajectories could be detailed in the following steps:

- Start the simulators, apply proper control and planning for each trajectory.
- Launching ROS nodes of the pose estimators which subscribe to the */rgb* topics from the simulator.
- Recording every PoseStamped message published from each of the pose estimators.
- Save the recorded estimated poses along with their timestamps into text format to feed into the RPG evaluation tool.
- The RPG evaluation tool performs time synchronization, and scale alignment between the estimated and ground-truth poses, and visualizes them. The scale factor is calculated using the Umeyama method [19]

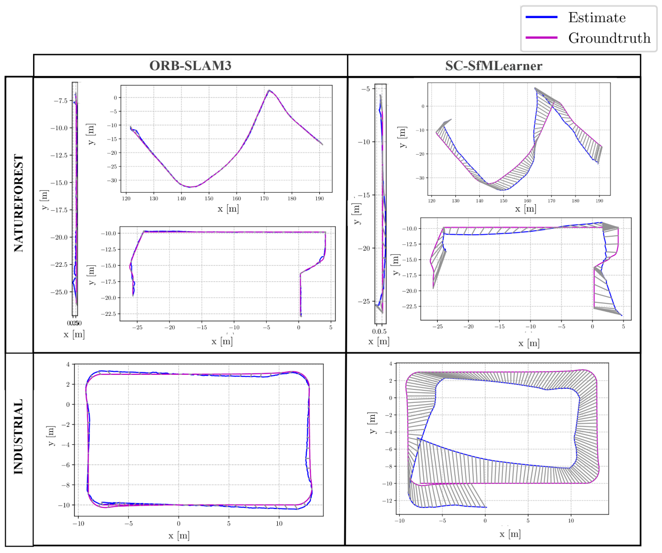


Fig. 7. Evaluation results visualized from RPG toolbox. In general, ORB-SLAM3 provides a smoother and more accurate estimation. SC-SfMLearner is vulnerable to trajectories that contain a large number of yaw rotations. The failure case of SC-SfMLearner is shown clearly with the rectangular trajectory, as the Industry environment since not only many yaws operations but also a poor-scene (fewer features) environment.

As seen from Fig. 7, ORB-SLAM3 outperforms SC-SfMLearner, in general. For three trajectories recorded from the Natureforest environment, ORB-SLAM3 achieve outstanding results, with the predicted trajectories being smooth and strictly sticking to ground-truth ones. SC-SfMLearner outputs rougher trajectories; however, the estimated poses still follow closely to the ground-truth, in terms of both shape and direction. It is noted that our implementation of SC-SfMLearner is trained on interpolated data; however, the evaluation results on original poses in the dataset are comfortably optimistic. On the other hand, SC-SfMLearner suffers heavily from scale drift when experimented on in the Industry environment. This tendency could stem from the fact that image data recorded from this environment contain fewer features compared to the Natureforest scenes. Also, it can be obviously observed that SC-SfMLearner is much more sensitive to yaw rotations. With no yaw rotations (straight flight), SC-SfMLearner can accurately follow the ground truth. However, the higher the number of yaw rotations of the drone and the greater the yaw angles, the more likely that the algorithm would fail to predict a satisfactory result. This limitation of SC-SfMLearner may originate from the lack of yaw rotation in the data used to train the pose network. Regarding ORB-SLAM3, the monocular tracker of the algorithm accomplishes high-accuracy estimation, with the predicted path bearing the same rectangular shape as the ground-truth trajectory. This result can confirm the power of ORB-SLAM3 as a versatile and accurate pose estimator. On other hand, more training data and advanced algorithms are necessary to be added to learning methods to avoid yaw-drift. Proper-control conditions can be also proposed based, for example, on very slow yaw rotations, to optimize the performance of the learning VO method in practical use.

TABLE II. Real-time ROS implementation

	ORB-SLAM3		**SC-SfMLearner**	
Device	CPU	Jetson NX	GPU	Jetson NX
Frame rate (FPS)	41	22.76	244	37.89
Maximum inference (ms)	50	110.74	12.8	61.42
Minimum inference (ms)	20	29.63	4	22.27

Regarding the running performance, we estimate three metrics for real-time ROS implementation of both publishers: frame rate, maximum and minimum inference times, and results are shown in the Table II. ORB-SLAM3 is estimated on CPU intel i5 8th and SC-SfMLearner is estimated on GPU GTX 1080Ti. From the table, we can see ORB-SLAM3 costs much more inference time than SC-SfMLearner. These results may stem from the fact that ORB-SLAM3 has not been optimized for parallel computing on GPU. The architecture of SC-SfMLearner is written fully on PyTorch, making it deployable and parallelizable via CUDA and CUDNN backend. In order to prove the real-time performance of our pose publishers, we measure both methods on Jetson Navier NX. Even though the computation capabilities of the embedded NX board is limited, both algorithms are able to reach real-time performance. The ability to perform real-time pose prediction at high rate of SC-SfMLearner is more apparent when we convert the trained weights to TensorRT format, optimizing it to utilize the full performance of the embedded GPU, reaching 37.89 FPS.

V. Conclusion

This paper has provided a real-time implementation and a comprehensive and thorough comparison between a conventional feature-based (ORB-SLAM3) and a self-supervised learning-based (SC-SfMLeaner) VO method. The results confirm the outperforming of the features-based ORB-SLAM3 over learning-based counterpart, SC-SfMLearner. The whole evaluation process is done by integrating the aforementioned pose estimators into a real-time navigation system, benchmarking on streams of photo-realistic synthesized RGB data acquired from Flightmare. This work also points out that learning-based algorithms are vulnerable to failure cases in which the drone trajectory contains excessive yaw rotations. Drawn from that conclusion, future works could be extended in order to construct an appropriate UAV controller, such that the onboard VO system could provide the most reliable estimation. The learning method can be improved to not only solve the limitation of yaw rotation issue but also to combine with other sensors such as IMU, and LIDAR to tackle the scale-ambiguity of monocular VO and provide more robustness and efficient learning-based state estimator.

Acknowledgment

This research is funded by the Vietnam Ministry of Science and Technology (MoST) under project number NDT/AT/22/09.

References

[1] Nistér *et al.*, "Visual odometry," in *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, vol. 1. Ieee, 2004, pp. I–I.

[2] I. Matthews and S. Baker, "Lucas-kanade 20 years on: A unifying framework," *International journal of computer vision*, vol. 56, no. 3, pp. 221–255, 2004.

[3] Zhou *et al.*, "Unsupervised learning of depth and ego-motion from video," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 6612–6619.

[4] Bian *et al.*, "Unsupervised scale-consistent depth and ego-motion learning from monocular video," *Advances in neural information processing systems*, vol. 32, 2019.

[5] Geiger *et al.*, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 3354–3361.

[6] Burri *et al.*, "The euroc micro aerial vehicle datasets," *The International Journal of Robotics Research*, vol. 35, 01 2016.

[7] Cordts *et al.*, "The cityscapes dataset for semantic urban scene understanding," *CoRR*, vol. abs/1604.01685, 2016. [Online]. Available: http://arxiv.org/abs/1604.01685

[8] Campos *et al.*, "Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam," *IEEE Transactions on Robotics*, vol. 37, no. 6, pp. 1874–1890, 2021.

[9] Song *et al.*, "Flightmare: A flexible quadrotor simulator," *arXiv preprint arXiv:2009.00563*, 2020.

[10] Furrer *et al.*, *RotorS – A Modular Gazebo MAV Simulator Framework*, 01 2016, vol. 625, pp. 595–625.

[11] Engel *et al.*, "Direct sparse odometry," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 3, pp. 611–625, 2017.

[12] D. G. Viswanathan, "Features from accelerated segment test (fast)," in *Proceedings of the 10th workshop on image analysis for multimedia interactive services, London, UK*, 2009, pp. 6–8.

[13] Calonder *et al.*, "Brief: Binary robust independent elementary features," in *European conference on computer vision*. Springer, 2010, pp. 778–792.

[14] Yousif *et al.*, "An overview to visual odometry and visual slam: Applications to mobile robotics," *Intelligent Industrial Systems*, vol. 1, no. 4, pp. 289–311, 2015.

[15] M. Faessler, D. Falanga, and D. Scaramuzza, "Thrust mixing, saturation, and body-rate control for accurate aggressive quadrotor flight," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 476–482, 2016.

[16] Kingma *et al.*, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[18] Z. Zhang and D. Scaramuzza, "A tutorial on quantitative trajectory evaluation for visual (-inertial) odometry," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 7244–7251.

[19] S. Umeyama, "Least-squares estimation of transformation parameters between two point patterns," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 4, pp. 376–380, 1991.