Learning Dynamics Factors for Optimization-based State Estimation

Li-Yu Lo[1]

***Abstract*— Recently, optimization-based state estimators have played a dominating role in the field of mobile robotics. Structured as a factor-graph optimization problem (FGO), such estimator allows a tightly-coupled, easily integrated, and flexible framework to incorporate multi-sensors fusion. While sensory information such as cameras, IMUs, and LiDARs are widly seen in state-of-the-art FGO estimators, existing literatures seldom take the dynamics of robots into consideration, which potentially leave out crucial information like external forces or motion induced by actuation inputs. Another major motivation for the inclusion of dynamics could be also seen in offboard control scenarios, where onboard, high-frequency inertial data are lacking. In this report, we therefore attempt to integrate the "dynamic factors" based on known controller inputs into the estimator. Nevertheless, it is also known that the model-prediction motion are prone to having discrepancies with actual motion. Hence, in this project, dynamic factors are regressed by a Temporal Convolutional Network (TCN) with inputs of sequential control signal before it is included to the backend FGO. We implement a FGO-based visual-dynamic odometry (VDO) and evaluate the performance with conventional visual odometry (VO) in batch on a quadrotor toy dataset. *Without further tuning and limited project time*, the preliminary results show comparable performance with conventional method, providing a promising alternative when IMU data is not available during sensor fusion.**

I. Background

Simultaneous Localization and Mapping (SLAM) is one of the most well-known problems in the field of mobile robots; to navigate a robot, the control problem starts with a robust state estimation feedback. While a single sensor would have sufficed, to improve the performance, it is common to carry out the task with multi-sensor. Nevertheless, such setting also increases the complexity of the problem, and it is critical to provide a systematic algorithm to fuse the information from multiple sources. To solve the sensor fusion problem, various framework could be seen; popular methods include early filter-based method proposed by [1], [2] as well as recent development in optimization-based (graph-based) framework [3], [4]. It has also been proven that the latter provides a higher performance compared to the former via the inclusion of more data temporally [4]. Furthermore, with the advancements in embedded computing technologies, the improved computation power has made graph-based estimators become the mainstream solution.

For a light-weighted platform, e.g., a quadrotor, the most frequently adopted combination of sensor is camera and inertial measurement unit (IMU) due to their SWaP advantage (size, weighted and power). Representative work could be seen in [4], [5]. Yet in such settings, most literatures seldom consider the robot's dynamic information, which could potentially neglect useful information. The appearance of an estimator incorporating all visual, inertial, and dynamic configurations did not occur until the work introduced by [6]. Inspired by [6] and motivated by offboard control scenarios where high frequency IMU data is not available, this project works on a visual-dynamic odometry (VDO). As the name implies, we solely take camera images (both color and depth) and sequential control inputs to estimate the states of the robot. Furthermore, as mentioned, we formulate the estimator into a graph-SLAM, i.e., a factor-graph optimization (FGO) problem. Note that the term "factor" in FGO can also be understood as the constraints imposed by the vision sensors and control inputs within a time series. More details will be provided in section II.

On the other hand, the dynamic factor induced by sequential control inputs is basically a model-prediction motion, which, if applied directly through pre-integration theory [7], will result in large discrepancies with real system. Therefore, encouraged by literatures from the field of inertial odometry (IO) ([8], [9], [10], [11]) where learning-based method is exploited, we aim to regress the dynamic factor via neural network. Different from the IO algorithms, where the motion is regressed by sequential acceleration and angular rate measurement, we feed the model with sequential control signals. In a physical sense, both applications are taking

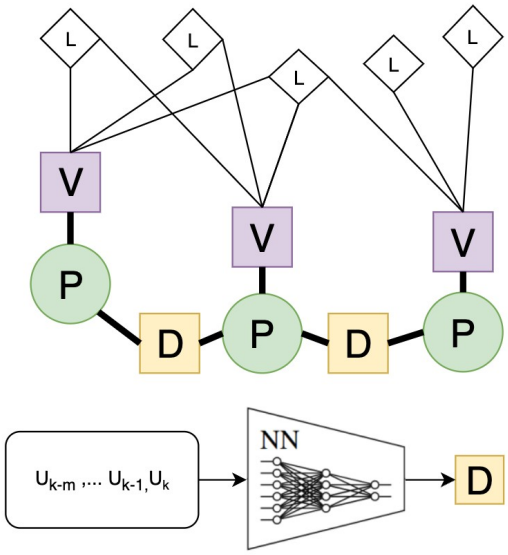

Fig. 1: The constellation of the LED markers onboard.

Li-Yu Lo, sID: 20997405, {lloac}@{connect.ust.hk},
COMP5212 Machine Learning @ Fall 2023 HKUST,
December 15, 2023

acceleration information as data input. As for the model family, we adopt a temporal convolutional network (TCN) [12] to carry out the prediction. The inferred dynamic factor will then be fed back to the FGO optimizer to conduct the final state estimation.

The rest of the report is structured as following: section II first elucidates the FGO problem, whereas section III will supplement training algorithm. Successively, section IV presents the preliminary results. Lastly, a conclusion for this project is drawn in V.

II. Keyframe FGO

A. Keyframe Selection

The batch estimation does not include all states at any time step. Instead, only the information at "keyframes" are taken into consideration. In particular, we follow the selection criterion proposed in [4], where a keyframe is generated when the overlapping area of the convex hull of landmarks detection between current image and previous keyframe image is less than a threshold. In addition to that, we add a keyframe activation condition based on time difference between current frame and previous keyframe. Then, when a keyframe is selected, we stack the measurements and control signals into the optimization buffer. The batch optimization is carried out after all the keyframes are selected.

B. FGO

The problem is essentially as the following: given a batch of keyframe measurements of landmarks, and the dynamic factors between each neighboring keyframe, infer the 6 degree of freedom (DoF) states of the keyframes at keyframe $k = 0, 1, ..., n$, as well as the positions of all landmarks $i = 0, 1, ..., m$. The batch estimation of keyframes is expressed mathematically as follows:

$$\begin{aligned} \mathcal{X} &= [\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_k, l_0, l_1, ..., l_i], \\ &\quad where \;\; k \in [0, n], i \in [0, m], && (1) \\ \mathbf{x}_k &= [\mathbf{p}, \mathbf{R}] \in \mathcal{M}, && (2) \\ \mathcal{M} &= \mathbb{R}^3 \times \mathbb{SO}(3) = \mathbb{SE}(3), && (3) \\ l_i &\in \mathbb{R}^3. && (4) \end{aligned}$$

In particular, $\mathbf{x}_k$ is expressed on manifold $\mathbb{SE}(3)$, which contains the translation and rotation information. Note that manifold $\mathbb{SE}(3)$ is also categorized as a lie group with its logarithm mapping being a lie algebra [13]. The VDO factor graph optimization problem could therefore be written as the following:

$$\mathcal{X} = \arg\min_{\mathcal{X}} \{ r_{\mathcal{D}}(\mathcal{X}, \mathcal{D}, \mathbf{R}_d) + r_{\mathcal{V}}(\mathcal{X}, \mathcal{L}, \mathcal{Z}, \mathbf{R}_v) \}, \quad (5)$$

where

$$\begin{aligned} r_{\mathcal{D}}(\cdot) &= \sum_{k=1}^{n} \| (\mathbf{x}_k \boxminus \mathbf{x}_{k-1}) \boxminus \mathbf{d}_{k-1}^k \|^2_{\mathbf{R}_d} && (6) \\ r_{\mathcal{V}}(\cdot) &= \sum_{k=0}^{n} \sum_{i=0}^{m'_k} \| \mathbf{K}\mathbf{x}_k l_i - \mathbf{z}_i \|^2_{\mathbf{R}_p}, && (7) \\ \mathcal{D} &= \{ \mathbf{d}_{k-1}^k | k = 1, ...n \}, \\ \mathcal{L} &= \{ \mathbf{l}_i | i = 1, ...m \}, \\ \mathcal{Z} &= \{ \mathcal{Z}_k | k = 1, ...n \}, \\ \mathcal{Z}_k &= \{ \mathbf{z}_i | i = 1, ...m'_k \}, \\ \mathbf{R}_d &\in \mathbb{R}^{3\times3}, \\ \mathbf{R}_v &\in \mathbb{R}^{6\times6}, \\ \mathbf{K} &\in \mathbb{R}^{3\times3}. && (8) \end{aligned}$$

From above, $\mathcal{D}$ is the dynamic factor set, which contains the factors between each neighboring keyframe. The factor is regressed based on all the sequential control inputs between keyframe $k-1$ to k. In addition, $\mathcal{L}$ is the landmarks set, and $\mathcal{Z}$ is the measurement set of all keyframes. Note that $|\mathcal{Z}_k| \leq m'_k$, where it depends on the measurement number of the detection at k. $\mathbf{R}_d$ and $\mathbf{R}_v$ are the covariance matrices (information matrices), while $\mathbf{K}$ being the intrinsic matrix of the pinhole model of the camera. In eqn. 6, the $\boxminus$ operation is defined as:

$$\mathbf{A} \boxminus \mathbf{B} = \mathbf{Log}(\mathbf{B}^{-1} \circ \mathbf{A}) = \Delta, \mathbf{A}, \mathbf{B} \in \mathcal{M}, \Delta \in \mathbb{R}^6, \quad (9)$$

where the $\mathbf{Log}(\cdot)$ is the logarithm mapping of manifold $\mathcal{X}$ to its lie algebra. As for $\circ$, it is the group composition operation within the Lie group.

The nonlinear optimization problem is then solved by either Gaussian-Newton (GN) or Levenberg–Marquardt (LM) algorithm, where the system is linearized, iteratively, at $\mathcal{X}_s$ (s is the solving step), forming and solving a linear system:

$$\mathbf{J}(\mathcal{X}_s)^T \mathbf{J}(\mathcal{X}_s) \xi_s = -\mathbf{J}(\mathcal{X}_s)^T r(\mathcal{X}_s), \quad (10)$$

$$(\mathbf{J}(\mathcal{X}_s)^T \mathbf{J}(\mathcal{X}_s) + \lambda \mathbf{I}) \xi_s = -\mathbf{J}(\mathcal{X}_s)^T r(\mathcal{X}_s) \quad (11)$$

Note that $\xi_s \in \mathbb{R}^6$ is the perturbation of the manifold, which is retrieved through exponential and logarithm mapping of Lie group and algebra. Eqn. 10 is used in GN algorithm, whereas eqn. 11 is used in LM algorithm. ξ_s is also the step for acquiring $\mathcal{X}_{s+1}$:

$$\mathcal{X}_s \leftarrow \mathcal{X}_s \boxplus \xi_s \quad (12)$$

the $\boxplus$ operation is defined as:

$$\mathbf{A} \boxplus \Delta = \mathbf{A}\mathbf{Exp}(\Delta), \mathbf{A} \in \mathcal{M}, \Delta \in \mathbb{R}^6, \quad (13)$$

where $\mathbf{Exp}(\cdot)$ is the exponential mapping that maps Lie algebra of $\mathcal{M}$ to $\mathcal{M}$, which is also the inverse mapping of $\mathbf{Log}(\cdot)$. After reaching convergence criterion, usually when $\xi_s < \epsilon$, the system is then solved.

It is admitted that a few derivations of operations on lie group are omitted here, and we refer readers to [13] for more details.

III. TCN

As mentioned, we adopt TCN to regress the dynamic factors. Due to time limitation, in this project, we only regress the 3D displacement, as rotation information requires further fine-tuning in terms of the coordinate selection (i.e., within body frame or inertial frame).

A. Data Preparation

We collected 12 flight logs, in which each flight log contains the information of a flight mission that is around 1 minute. Among all flight mission, we set different lateral velocities and vertical movements. We fixed the yaw angle to reduce the effect from rotation, as we are only regressing the displacement between keyframes. Also note that the control frequency is fixed at 100 Hz.

Within the logs, we retrieved control inputs $u_t \in \mathbb{R}^4$, and set sequential window sizes that are varied from 40 to 200. This also implies the time-based keyframe activation condition is 2s. Through applying different window sizes on the 12 dataset, we retrieved around 10 million data input points. We also performed data augmentation by perturbing the inputs with a uniform noise distribution.

B. Model Training

We opt for TCN mainly due to its outperforming results when compared to RNN and LSTM on sequential data. The operation mechanism of TCN is similar to conventional CNNs, only with the convolution filter being set to `CONV1D` and being dilated to have a larger receptive field. TCN is also set to be having the same input and output size through size settings of `CONV1D` and `PADDING`. The above-mentioned characteristics are derived from the 2 main principles of TCN [12]: 1. `INPUT_LENGTH` = `OUTPUT_LENGTH` and 2. No data leakage from the future.

The detail of our model are described as follows. As we are forwarding the control inputs with $\mathbb{R}^4$, with a varied sequential size from 40-200, we fix the input tensor with a shape of $[4, 200]$. For window size < 200, the input is padded with zeros. Meanwhile, the output size of the regression model is $\mathbb{R}^3$. We implement the TCN with 7 convolution blocks, where the `FILTER_SIZE` is 2, while the channel size being respectively $[64, 64, 64, 64, 128, 128, 128]$. A fully connected layer `LINEAR` is then attached at the end of all the temporal convolution blocks.

The dataset is split $[0.8, 0.2]$ for, respectively, training subset and testing subset. `ADAM` optimizer is used for training, while the training was performed on various batch size of $[128, 256, 512, 1024]$. We also implemented a Long Short-Term Memory LSTM for comparison. All the training was carried on GeForce RTX 3090. `EPOCH` number was set to be 100.

IV. Preliminary Results

A. Implementation

To evaluate the algorithm, we implement the batch estimator in ROS C++/Python, and perform the inference on a toy VDO dataset. The reason that it is referred as a "toy" dataset is because the landmarks are set artificially with infrared LED markers, while the camera is installed with an infrared filter. To run the TCN model in C++, we converted the `.pt` file to C++ executable format through `libtorch`. As for the optimization task, it is done through GTSAM 4.0 package [14]. Also, all ground truth data were recorded through VICON motion capture system.

B. Training Results

In this section, the following will be presented:

- General training result of TCN and LSTM models.
- Comparison between TCN and LSTM results.
- Comparison between different batch size on TCN model.

Note that only the results from the first 40 epochs were presented here.

From figure 2a and figure 2b, it could be seen that the overall training results converge. TCN converged quite swiftly after a few epochs, while LSTM took a bit longer to converge. Referring to figure 2c, it could be seen that LSTM generally has a lower cost; nevertheless, the generalization capability is observed to be worse than TCN, where the testing loss fluctuates and exceed the training loss. Therefore, despite lower losses, during the final FGO implementation, we still adopted TCN for dynamic factor regression.

Lastly, figure 2d illustrates the training results on different batch size. It could be seen that with a lower batch size, the model could learn deeper, resulting in a lower cost.

C. Batch State Estimation

Here we present the final batch state estimation results in terms of absolute trajectory error:

$$e_{ATE} = \frac{1}{N}\sum_{i=1}^{N}\sqrt{\frac{1}{K}\sum_{k=1}^{K}\|\mathbf{x_{k,i}} \boxminus \hat{\mathbf{x}}_{\mathbf{k,i}}\|}, \tag{14}$$

where $\mathbf{x_{k,i}}$ is the ground truth data, and $\hat{\mathbf{x}}_{\mathbf{k,i}}$ is the estimated states.

Below table then presents the numerical comparison between VDO and VO on 4 dataset. From above, it could be

TABLE I: Results from Batch FGO

	VDO	VO
SLOW	0.0139	0.0127
MID	0.0262	0.0254
FAST	0.0376	0.0376
SFAST	0.1152	0.1137

seen that the final numerical results of VDO has a slight larger error than VO. We conjecture the reason being the lack of rotation in dynamic factor regression mentioned in section III, which therefore induce erroneous dynamic factor in terms of rotation during the final batch optimization. The second reason that causes this might be the time synchronization between camera image and control signals. It is also considered that, the adopted toy dataset is rather

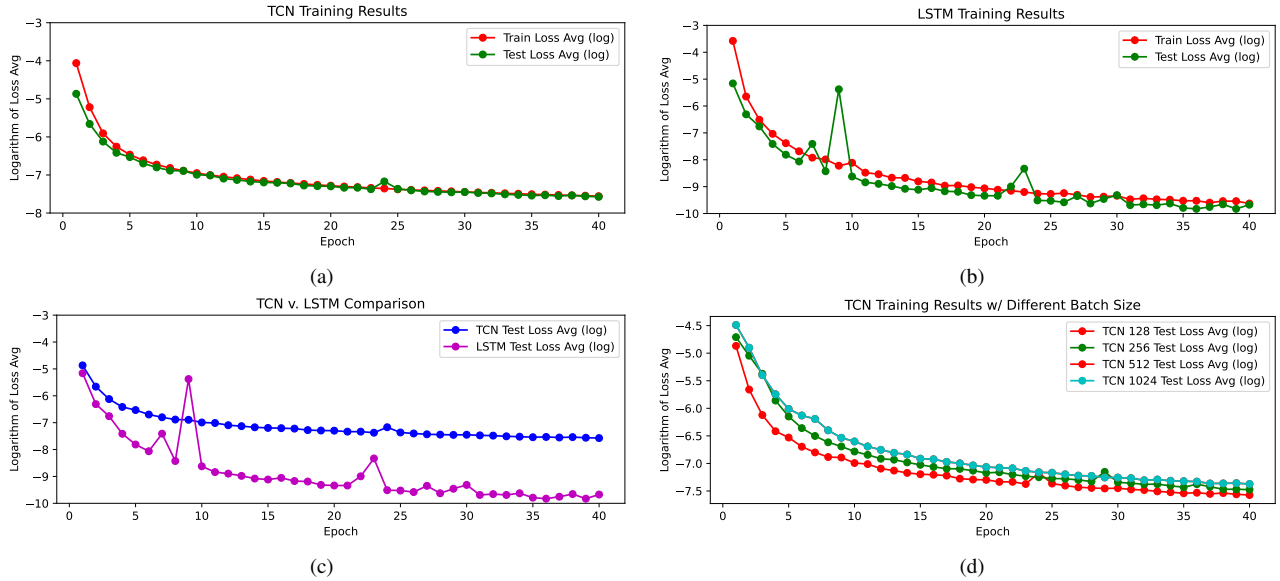

Fig. 2: Training results. (a) TCN training results. (b) LSTM training results. (c) TCN and LSTM comparison. (d) TCN training comparison between different batch size.

ideal and friendly for visual odometry, as the landmarks and their correspondences can be readily found, rejecting the necessity of conducting sensor fusion. In normal deployment, the detection and feature matching tasks are much more challenging; we contended that the VDO might perform better VO in that scenario.

Nevertheless, the above results still provide a promising alternative strategy for sensor fusion when onboard IMU data is not available. It also offers an outlook to a real-time, sliding-window based estimator for offboard control framework.

V. Conclusion

In this project, we proposed a graph-based estimator by using solely visual factors and dynamic factors. The implementation was carried through formulating the problem as a FGO, where the dynamic factor is retrieved from a TCN model. The training as well as the batch estimation results are then presented and discussed.

In near future, we plan to include rotational regression into the model, while attempting to add more layers to TCN to test the performance. Due to the Gaussian filtering nature of FGO problem, we also propose to include covariance training. Ultimately, we wish to deploy the system in an online fashion for an offboard-controlled quadrotor.

Supplementary Material

Software of this report is available at: `https://github.com/pattylo/ledvo.git` and `https://github.com/pattylo/ledvo_train.git`.

References

[1] T. Bailey, J. Nieto, J. Guivant, M. Stevens, and E. Nebot, "Consistency of the ekf-slam algorithm," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2006, pp. 3562–3568.

[2] A. I. Mourikis and S. I. Roumeliotis, "A multi-state constraint kalman filter for vision-aided inertial navigation," in *Proceedings 2007 IEEE international conference on robotics and automation*. IEEE, 2007, pp. 3565–3572.

[3] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "Orb-slam: a versatile and accurate monocular slam system," *IEEE transactions on robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[4] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale, "Keyframe-based visual–inertial odometry using nonlinear optimization," *The International Journal of Robotics Research*, vol. 34, no. 3, pp. 314–334, 2015.

[5] T. Qin, P. Li, and S. Shen, "Vins-mono: A robust and versatile monocular visual-inertial state estimator," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 1004–1020, 2018.

[6] B. Nisar, P. Foehn, D. Falanga, and D. Scaramuzza, "Vimo: Simultaneous visual inertial model-based odometry and force estimation," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2785–2792, 2019.

[7] T. Lupton and S. Sukkarieh, "Visual-inertial-aided navigation for high-dynamic motion in built environments without initial conditions," *IEEE Transactions on Robotics*, vol. 28, no. 1, pp. 61–76, 2011.

[8] C. Chen, X. Lu, A. Markham, and N. Trigoni, "Ionet: Learning to cure the curse of drift in inertial odometry," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[9] H. Yan, S. Herath, and Y. Furukawa, "Ronin: Robust neural inertial navigation in the wild: Benchmark, evaluations, and new methods," *arXiv preprint arXiv:1905.12853*, 2019.

[10] W. Liu, D. Caruso, E. Ilg, J. Dong, A. I. Mourikis, K. Daniilidis, V. Kumar, and J. Engel, "Tlio: Tight learned inertial odometry," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 5653–5660, 2020.

[11] G. Cioffi, L. Bauersfeld, E. Kaufmann, and D. Scaramuzza, "Learned inertial odometry for autonomous drone racing," *IEEE Robotics and Automation Letters*, vol. 8, no. 5, pp. 2684–2691, 2023.

[12] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," *arXiv preprint arXiv:1803.01271*, 2018.

[13] J. Sola, J. Deray, and D. Atchuthan, "A micro lie theory for state estimation in robotics," *arXiv preprint arXiv:1812.01537*, 2018.

[14] F. Dellaert and G. Contributors, "borglab/gtsam," May 2022. [Online]. Available: https://github.com/borglab/gtsam)