Learning to Bundle-adjust: A Graph Network Approach to Faster Optimization of Bundle Adjustment for Vehicular SLAM

Tetsuya Tanaka*
Socionext Inc.
tanaka.tetsuya@socionext.com

Yukihiro Sasagawa*
Socionext Inc.
sasagawa.yukihiro@socionext.com

Takayuki Okatani
Graduate School of Information Sciences, Tohoku University / RIKEN Center for AIP
okatani@vision.is.tohoku.ac.jp

Abstract

Bundle adjustment (BA) occupies a large portion of the execution time of SfM and visual SLAM. Local BA over the latest several keyframes plays a crucial role in visual SLAM. Its execution time should be sufficiently short for robust tracking; this is especially critical for embedded systems with a limited computational resource. This study proposes a learning-based bundle adjuster using a graph network. It works faster and can be used instead of conventional optimization-based BA. The graph network operates on a graph consisting of the nodes of keyframes and landmarks and the edges representing the landmarks' visibility. The graph network receives the parameters' initial values as inputs and predicts their updates to the optimal values. It internally uses an intermediate representation of inputs which we design inspired by the normal equation of the Levenberg-Marquardt method. It is trained using the sum of reprojection errors as a loss function. The experiments show that the proposed method outputs parameter estimates with slightly inferior accuracy in 1/60–1/10 of time compared with the conventional BA.

1. Introduction

Structure-from-Motion (SfM) and visual SLAM (simultaneous localization and mapping) have been successfully used in many real-world applications of computer vision, robotics, augmented reality, and related areas [25, 29]. To improve the accuracy and robustness of 3D reconstruction, researchers have considered several different approaches, such as feature-point-based methods [27], direct methods [7], learning-based methods [24, 23], and their hybrids [3].

Among these, the feature-point-based methods are currently the most widely used. Visual SLAM is often used in embedded systems with limited computational resources, narrowing choices to those using light-weight feature descriptors, such as ORB-SLAM [16, 17]. Many SfM systems also use feature-point-based methods; they are used to estimate camera poses for the subsequent step of multi-view stereo, in which the dense surfaces of objects/scenes are reconstructed.

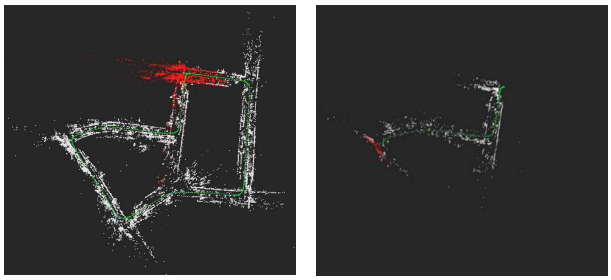


Figure 1. Limiting the computational time budget for local BA results in frequent tracking failures. Left: Without a limit. Right: With a limit. See the text for details.

Although it is comparatively smaller, the feature-point-based methods still have a high computational cost. What dominates in their execution time is the step of bundle adjustment (BA). It optimizes the unknown parameters, the 3D positions of the landmarks associated with the feature points and camera poses, refining their initial values to get accurate estimates. This step is usually the bottleneck in terms of the speed of visual SLAM and SfM systems. Taking ORB-SLAM, for instance, it frequently performs BA locally over several keyframes and their associated landmarks, maintaining the accuracy of the most recent reconstruction. BA occupies roughly 60–80% of the execution time needed for the mapping operation.

More importantly, the speed of local BA determines

*These authors contributed equally to this work.

the robustness of SLAM systems. Keyframe-based SLAM systems such as ORB-SLAM can track landmarks more stably when there are spatially denser keyframes. ORB-SLAM employs the strategy of issuing more than sufficient keyframes and culling unnecessary ones later. However, a new keyframe cannot be created while local BA is running due to the causality. Thus, the ability to complete local BA quickly is necessary for issuing keyframes densely, and it is a basis for robust SLAM. This requirement is more critical for embedded systems having low-speed processors. A remedy is to set a limit on the computational budget for local BA to balance the accuracy of SLAM and the robustness of tracking. For example, we can set the maximum iteration counts for local BA. However, this does not work well in practice, as shown in Figure 1.

In this paper, we consider a learning-based method that can perform BA more quickly. BA is essentially nonlinear minimization of the sum of squares, and conventionally the Levenberg-Marquardt method is employed. The method iteratively solves a linear equation to obtain a small parameter update and updates the parameters until convergence. When regarding the entire process as a black-box, which receives initial parameter values and outputs their optimal values, we replace it with computation by a graph network [30, 28]. To be specific, we train the graph network with a set of input-output pairs for some videos, i.e., the initial values fed to BA and their optimized values, aiming at computing the optimal parameter values in a shorter time.

We report the results of our experiments focusing on monocular SLAM, in which we used OpenVSLAM [22] as a testbed and evaluate the method on the KITTI dataset [8]. In the experiments, we trained our graph network on a few sequences from the dataset and tested it on the other sequences. We create training samples by the conventional BA's inputs (i.e., g2o [13]) applied to the training sequences. Using them, we train the graph network using the sum of reprojection errors as a loss. Our method achieved slightly lower accuracy with 1/60–1/10 of computational time compared with the original BA.

2. Related Work

2.1. Speeding up Bundle Adjustment

BA is the core element of SfM and visual SLAM. It computes camera poses and landmark positions that minimize the sum of reprojection errors, for which the Levenberg-Marquardt method is typically used. Its general algorithm starts with initial values and iteratively updates the parameters until convergence, where the updates are computed by solving the normal equation having the size of the parameters. In the case of BA, the equation's matrix has a particular block structure because there is no direct interaction between camera poses and landmarks. This procedure makes it possible to convert the normal equation into a reduced camera system (RCS), which is smaller and can be solved more efficiently [15, 10]

Several algorithms exist for solving the RCS, and the fastest one differs depending on the problem size. With the growing size of SfM, it has been examined how to solve a large-scale problem efficiently, leading to inexact solvers based on the conjugate gradient (CG) method [1]. For small to mid-size problems (i.e., a few hundred images according to [1]), exact solvers are superior, which use dense or sparse Cholesky decomposition. In parallel to these, researchers have also studied the implementation of the algorithms that best utilize CPU, and GPU parallelism [12, 26].

Another approach uses the belief propagation (BP) algorithm, mainly loopy BP such as Gaussian BP, on the factor graph consisting of camera poses and landmarks, aiming at estimating the marginal distribution of each parameter [19, 6]. When implemented in CPU or GPU, it fails to yield comparable performance to the above methods. Recently, however, it has been shown [18] that a graph processor (i.e., Graphcore's Intelligence Processing Unit) can run Gaussian BP to perform BA at a competitive speed.

Our method is a learning-based method that uses a graph network, which is orthogonal to any of the above methods, although there is some similarity to each. The method iteratively updates the parameters from given initial values, as with the LM method. However, it does not solve the normal equation; it directly predicts the updates from several inputs (i.e., the latest parameters and a few derived terms) whose design is inspired by the Levenberg-Marquardt updates. This point is the major factor of the speed-up from the standard BA. The graph network is built upon the factor graph used in the BP-based BA.

2.2. Learning to Simulate Complex Dynamics by Graph Networks

Graph networks (GNs) [5] are a type of graph neural networks [21]. Receiving a graph as input, a GN outputs a graph of the same shape but with different attribute values at their nodes and edges. During the propagation from the input to the output, information is exchanged between its nodes and edges in the form of message-passing. By training a GN using pairs of the input and the output, it can represent various phenomena emerging from interactions between multiple entities, where how to pass messages is adjusted.

This approach has been proven to be effective for learning complex forward dynamics, such as rigid body collision [4], particle-based simulation of complex physics [20], robotic control [14], to name a few, as well as the behavior of non-physical systems.

The idea of using machine learning models to perform complex physics simulations is not new [9]. It learns to rep-

resent the state transition between two time-points using observations as training data. It aims to avoid difficulties with traditional simulation based on physical models, such as the computational complexity or physical models' inaccuracy. However, making a universal ML model learn mapping with large degrees of freedom in high-dimensional state space is not simple. The success of GNs is attributable to the just enough representation of the interaction between entities by a graph.

We apply GNs to BA, where the input is the specified initial values for camera poses and landmarks, and the output is their optimized values. We use the factor graph consisting of camera poses and landmarks to construct a GN and tailor the inputs for the message passing, inspired by the Hessian used in the Levenberg-Marquardt method, enabling end-to-end learning.

3. Proposed Method

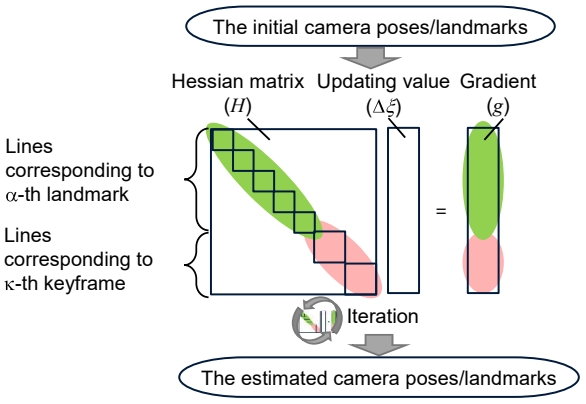
3.1. Learning to Bundle-adjust

Given the initial values of camera poses and landmarks' 3D coordinates, BA computes their estimates that best reproduce the given observations, i.e., the image coordinates of the landmarks. Visual SLAM systems frequently conduct small-/mid-size BA internally, which usually uses the Levenberg-Marquardt method. It iteratively updates the estimates by solving a linear equation. Its computational complexity increases with the number of parameters; 5-15 updates are necessary until convergence.

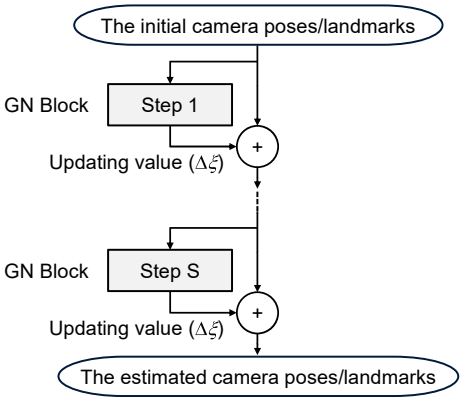
To complete this BA computation in a shorter time, we replace the Levenberg-Marquardt method with a learning-based method. Specifically, we train a model that receives the initial values of the camera poses and the landmarks and outputs their estimates. Such a model needs to fulfill the following requirements:

- It can handle a variable number of inputs/outputs, i.e., M camera poses and N landmarks, where M and N differ at each time. The provided observations also differ.
- It can predict the parameters as accurately as possible, despite that there are large degrees of freedom in the mapping from the input to the output.

To fulfill these requirements, we employ the graph network framework [5]. It uses a graph to make use of *inductive bias* of the problem at hand and performs inference on it; graphs are created adaptively depending on inputs. In this study, we use the popular graph representation with BA, i.e., a bipartite graph between two types of nodes, i.e., keyframes (i.e., camera poses) and landmarks, whose edges reflect the visibility of landmarks from keyframes. This graph handles all the possible interactions between



(a) The standard BA based on the Levenberg-Marquardt method



(b) The learning-based BA with graph networks

Figure 2. Comparison of the standard BA based on the Levenberg-Marquardt method and the proposed learning-based method.

keyframes and landmarks, making it possible to use the inductive bias in our problem. However, there remains a large degree of freedom in designing the components in the graph network. How should we choose a proper design?

Our solution is to mimic the Levenberg-Marquardt method; see Figure 2. As it will not contribute to speed-up if we perform the exact computation, we consider bypassing a part of the Levenberg-Marquardt computation with a learnable model. Our preliminary tests have found that it works to train a model to predict the update of parameters from the block diagonal elements of the Hessian matrix and the gradient.

Following the Levenberg-Marquardt method, we update the parameters multiple times, while the count of updates is fixed, not 'until convergence.' We design a graph-network (GN) block to update the parameters once and stack them to perform multiple counts of updates. We train the stack of GN blocks so that the final output will minimize reprojection errors. Thus, note that we do *not attempt or expect* to make each block predict a Levenberg-Marquardt update.

3.2. Predicting Parameter Updates

The updating equation of the Levenberg-Marquardt method is written as

$$(\mathbf{H} + \lambda \mathbf{I})\Delta\xi = \mathbf{g}, \tag{1}$$

where $\mathbf{H}$ is the Hessian matrix[1]; $\mathbf{g}$ is the gradient vector; and $\Delta\xi$ is the updates of the parameter ξ. We use $h^{xx}_{\alpha\alpha}$ ($\alpha = 1, \ldots, N$) and $\{h^{\omega\omega}_{\kappa\kappa}, h^{\omega t}_{\kappa\kappa}, h^{t\omega}_{\kappa\kappa}, h^{tt}_{\kappa\kappa}\}$ ($\kappa = 1, \ldots, M$) to denote the diagonal blocks of $\mathbf{H}$. Omitting the off-diagonal blocks and the gradient descent term $\lambda\mathbf{I}$, we rewrite the left hand side of (1 as

$$\begin{bmatrix} h^{xx}_{11} & & & & & & & & \\ & \ddots & & & & & & & \\ & & h^{xx}_{NN} & & & & & & \\ & & & h^{\omega\omega}_{11} & h^{\omega t}_{11} & & & & \\ & & & h^{\omega t}_{11} & h^{tt}_{11} & & & & \\ & & & & & \ddots & & & \\ & & & & & & h^{\omega\omega}_{MM} & h^{\omega t}_{MM} & \\ & & & & & & h^{t\omega}_{MM} & h^{tt}_{MM} & \end{bmatrix} \begin{bmatrix} \Delta\mathbf{x}_1 \\ \vdots \\ \Delta\mathbf{x}_N \\ \Delta\omega_1 \\ \Delta\mathbf{t}_1 \\ \vdots \\ \Delta\omega_M \\ \Delta\mathbf{t}_M \end{bmatrix}, \tag{2}$$

where $\Delta\mathbf{x}_\alpha$'s are the updates of the landmarks' 3D coordinates and $\{\Delta\omega_\kappa, \Delta\mathbf{t}_\kappa\}$'s are those of the camera poses.

The block $h^{xx}_{\alpha\alpha}$ and the associated component of $\mathbf{g}$ are as follows:

$$h^{xx}_{\alpha\alpha} = \sum_{j=1}^{M} I_{\alpha j}\mathbf{R}_j\mathbf{W}_{\alpha j}\mathbf{W}_{\alpha j}^\top\mathbf{R}_j^\top, \tag{3a}$$

$$g^{x}_{\alpha} = 2\sum_{j=1}^{M} I_{\alpha j}\mathbf{R}_j\mathbf{W}_{\alpha j}\mathbf{e}_{\alpha j}, \tag{3b}$$

where $I_{\alpha j} \in \{0, 1\}$ indicates the visibility of α-th landmark from j-th camera pose; $\mathbf{R}_j$ is the rotation matrix of j-th camera pose; $\mathbf{W}_{\alpha j}$ is a 3×2 matrix calculated from α-th landmark and j-th camera pose; and $\mathbf{e}_{\alpha j}$ is the difference from the reprojected image point and its observation.

The blocks $h^{\omega\omega}_{\kappa\kappa}$ and $h^{\omega t}_{\kappa\kappa}$ and the associated component of $\mathbf{g}$ are as follows:

$$h^{\omega\omega}_{\kappa\kappa} = \sum_{i=1}^{N} I_{i\kappa}(\mathbf{x}_i - \mathbf{t}_\kappa)\times\mathbf{R}_\kappa\mathbf{W}_{i\kappa}\mathbf{W}_{i\kappa}^\top\mathbf{R}_\kappa^\top\{(\mathbf{x}_i - \mathbf{t}_\kappa)\times\mathbf{I}\}, \tag{4a}$$

$$h^{\omega t}_{\kappa\kappa} = \sum_{i=1}^{N} I_{i\kappa}(\mathbf{x}_i - \mathbf{t}_\kappa)\times\mathbf{R}_\kappa\mathbf{W}_{i\kappa}\mathbf{W}_{i\kappa}^\top\mathbf{R}_\kappa^\top, \tag{4b}$$

$$g^{\omega}_{\kappa} = 2\sum_{i=1}^{N} I_{i\kappa}(\mathbf{x}_i - \mathbf{t}_\kappa)\times\mathbf{R}_\kappa\mathbf{W}_{i\kappa}\mathbf{e}_{i\kappa}, \tag{4c}$$

where $\mathbf{x}_i$ is i-th landmark and $\mathbf{t}_\kappa$ is the translation vector of κ-th camera pose.

Similarly, the blocks $h^{t\omega}_{\kappa\kappa}$ and $h^{tt}_{\kappa\kappa}$ and the associated component of $\mathbf{g}$ are:

$$h^{t\omega}_{\kappa\kappa} = \sum_{i=1}^{N} I_{i\kappa}\mathbf{R}_\kappa\mathbf{W}_{i\kappa}\mathbf{W}_{i\kappa}^\top\mathbf{R}_\kappa^\top\{(\mathbf{x}_i - \mathbf{t}_\kappa)\times\mathbf{I}\}, \tag{5a}$$

$$h^{tt}_{\kappa\kappa} = \sum_{i=1}^{N} I_{i\kappa}\mathbf{R}_\kappa\mathbf{W}_{i\kappa}\mathbf{W}_{i\kappa}^\top\mathbf{R}_\kappa^T, \tag{5b}$$

$$g^{t}_{\kappa} = 2\sum_{i=1}^{N} I_{i\kappa}\mathbf{R}_\kappa\mathbf{W}_{i\kappa}\mathbf{e}_{i\kappa}. \tag{5c}$$

We use these block-diagonal components of $\mathbf{H}$ and the associated block components of $\mathbf{g}$ to predict the updates of landmarks and camera poses. To be specific, we use a predictor $\phi^{\texttt{LM}}$ to predict the update of landmark as $\Delta\mathbf{x}_\alpha = \phi^{\texttt{LM}}(h^{xx}_{\alpha\alpha}, g^{x}_{\alpha})$. We use a predictor $\phi^{\texttt{KF}}$ to predict the update of the rotation and translation components as $[\Delta\omega_\kappa^\top, \Delta\mathbf{t}_\kappa^\top]^\top = \phi^{\texttt{KF}}(h^{\omega\omega}_{\kappa\kappa}, h^{\omega t}_{\kappa\kappa}, h^{t\omega}_{\kappa\kappa}, h^{tt}_{\kappa\kappa}, g^{\omega}_{\kappa}, g^{t}_{\kappa})$. For each of these predictors, we employ a four-layer neural network; details will be given below. We train $\phi^{\texttt{LM}}$ and $\phi^{\texttt{KF}}$ to predict those updates so that S iterations of the updates yield the optimal parameters minimizing the sum of the reprojection errors.

3.3. Implementation with Graph Network

We implement our learning-based BA using the graph network framework [5]. One may think that it receives a graph with initial parameter values and outputs the same graph but with optimal parameter values. As mentioned above, our graph is a common bipartite graph whose vertices are keyframes (i.e., camera poses) and landmarks; its edges connect these two types of vertices according to the visibility of the landmarks, as shown in Figure 3.

We denote the κ-th keyframe vertex by $V^{\texttt{KF}}_\kappa$ and the α-th landmark vertex by $V^{\texttt{LM}}_\alpha$, respectively. We use $E_{\alpha\kappa}$ to denote the edge connecting $V^{\texttt{KF}}_\kappa$ and $V^{\texttt{LM}}_\alpha$, if any. The vertices $V^{\texttt{KF}}_\kappa$ and $V^{\texttt{LM}}_\alpha$ retain the parameter values $(\mathbf{R}_\kappa, \mathbf{t}_\kappa)$ and $\mathbf{x}_\alpha$ as their attributes, respectively. Each of them also retains a flag named *un-fixed*, using which we control which keyframe or landmark to lock to resolve the gauge freedom, as is commonly done in BA. The edge $E_{\alpha\kappa}$ retains normalized image coordinates of keypoints (i.e., the observation of a landmark) as its attribute.

As is shown in Figure 2, the parameter values are updated by a series of S GN blocks. We use an identical GN block, including learnable weights for all the blocks. Figure 3 shows the design of a single GN block. It receives the whole graph, i.e., the graph structure and the parameter values stored at the vertices. It consists of five components, i.e., the two vertex predictors $\phi^{\texttt{KF}}$ and $\phi^{\texttt{LM}}$, which are explained above, two vertex aggregators $\rho^{\texttt{KF}}$ and $\rho^{\texttt{LM}}$, and one edge predictor $\phi^{\texttt{E}}$. The latter three (i.e., $\rho^{\texttt{KF}}$, $\rho^{\texttt{LM}}$, and $\phi^{\texttt{E}}$) do not have learnable weights; they exist to calculate the inputs to $\phi^{\texttt{KF}}$ and $\phi^{\texttt{LM}}$ according to the graph structure. They also enable backpropagation of gradients from the final output to the input through the stack of GN blocks.

More specifically, $\phi^{\texttt{E}}$ operates independently on each edge, as shown in Figure 4(a), receiving as inputs $(\mathbf{R}_\kappa, \mathbf{t}_\kappa)$

[1] We mean the Gauss-Newton approximation of the full Hessian here.

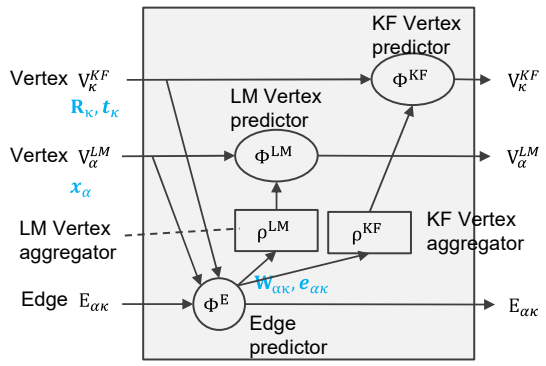
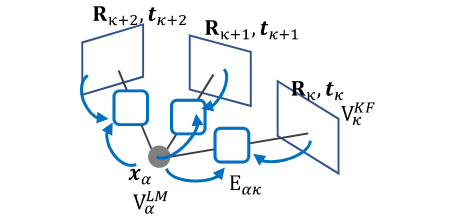
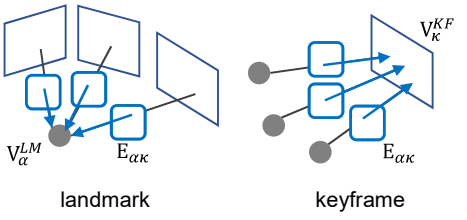


Figure 3. Overview of the GN block.



(a) Edge predictor



(b) Vertex aggregator

Figure 4. The information flow of (a) the edge predictor ϕ^{E} and (b) the vertex aggregators ρ^{LM} and ρ^{KF}.

and $\mathbf{x}_\alpha$ from its connecting verteces and calculating the component terms shared in the right hand sides of (3), (4), and (5), i.e., $\mathbf{e}_{\alpha\kappa}$, $\mathbf{R}_\kappa \mathbf{W}_{\alpha\kappa}$, and $(\mathbf{x}_\alpha - \mathbf{t}_\kappa)$, which are fed to ρ^{LM} and ρ^{KF}.

As shown in Figure 4, the vertex aggregator ρ^{LM} operates independently on each landmark vertex to perform the summation over the connecting keyframe verteces $j = 1, \ldots, M$, as in (3). Similarly, ρ^{KF} operates independently on each keyframe vertex to perform the summation over $i = 1, \ldots, N$ as in (4) and (5).

The outputs of these aggregators are fed to the two vertex predictors ϕ^{LM} and ϕ^{KF}, which predict the updates as $\Delta\mathbf{x}_\alpha = \phi^{\mathrm{LM}}(h^{\mathrm{LM}}_{\alpha\alpha}, g^{\mathrm{LM}}_\alpha)$ and $[\Delta\omega_\kappa^\top, \Delta\mathbf{t}_\kappa^\top]^\top = \phi^{\mathrm{KF}}(h^{\omega\omega}_{\kappa\kappa}, h^{\omega t}_{\kappa\kappa}, h^{t\omega}_{\kappa\kappa}, h^{tt}_{\kappa\kappa}, g^{\omega}_\kappa, g^{t}_\kappa)$, respectively, as mentioned earlier. For each of these predictors, we employ a four-layer MLP (multi-layer perceptron); its details are shown in Table 1. It employs layer normalization [2] at the first layer and three linear layers followed by ReLU and one linear layer for the output. The numbers indicate the hidden units and the outputs in the MLP. Note that learnable

Table 1. Design of the neural network used for the two vertex predictors.

Predictor	Hidden units / Activation			Outputs
ϕ^{LM}	12/ReLU	9/ReLU	9/ReLU	3
ϕ^{KF}	42/ReLU	18/ReLU	18/ReLU	6

weights exist only in these predictors. Figure 5 shows a diagram illustrating more details of the GN block.

3.4. Training the Graph Network

We train the graph network as follows. Choosing a monocular image sequence, we first apply a conventional SLAM system to it and extract local BA runs; each run possesses a multi-view system consisting of several keyframes and landmarks observed from them, initial values of the parameters, and the observations (i.e., the landmarks image coordinates from the keyframes).

We treat each BA run as an independent training sample. We construct a graph based on the structure of the multi-view system, feed it along with initial parameter values to the graph network, obtaining their updated estimates; see Figure 6. As mentioned earlier, the graph network is a stack of S GN blocks, and each block is identical, including the learnable weights in the two vertex predictors. As it predicts only parameter updates, we add them to the initial values to get the parameter estimates. We then calculate the loss for the estimate.

For the loss, we use the sum of the reprojection errors used in the standard BA, i.e., the sum of the differences of the image coordinates of the landmarks computed by the estimates from their observations. Following ORB-SLAM, we employ the Huber loss to ensure robustness in the presence of erroneous point correspondences, which is given by

$$L(e_x, e_y) = \rho(e_x) + \rho(e_y) \tag{6a}$$

$$\rho(e) = \begin{cases} \frac{1}{2}e^2 & \text{for } |e| \le \delta, \\ \delta(|e| - \frac{1}{2}\delta) & \text{otherwise,} \end{cases} \tag{6b}$$

where e_x and e_y are the reprojection error in x and y directions in an image; we set δ to 2.0 as in ORB-SLAM. We compute the sum of loss values over multiple BA runs in a minibatch and use the Adam optimizer [11] to reduce the loss.

4. Experimental Results

We conduct experiments to test the proposed method.

4.1. Experimental Configuration

We evaluate the performance of the proposed method on the monocular video sequences from the KITTI dataset [8].

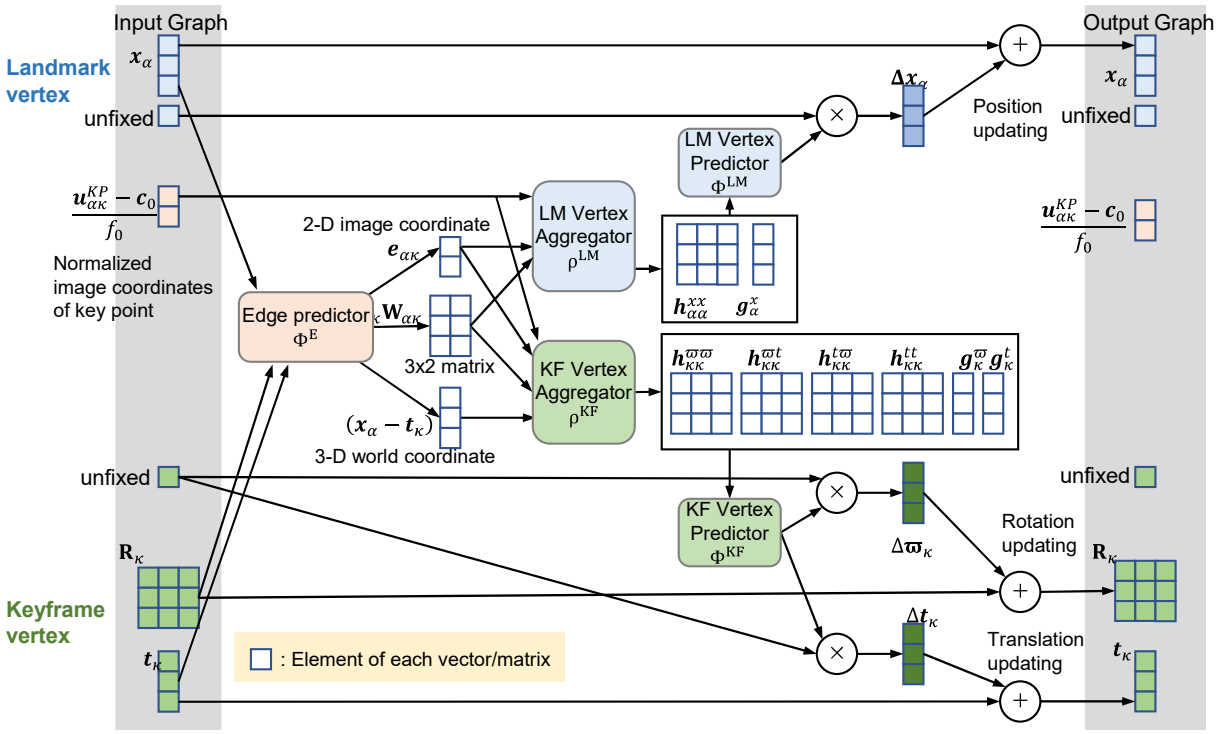


Figure 5. The internal structure of the GN block.

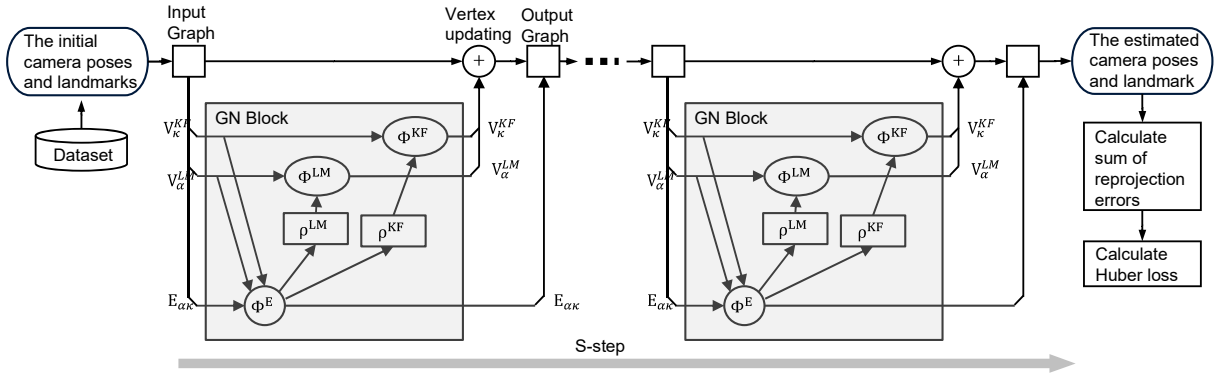


Figure 6. The information flow from the input to the output of the graph network at training time.

It contains multiple independent sequences of a vehicle running on different urban streets. We select some sequences from them for training our graph network and test it on other sequences.

We employ OpenVSLAM [22], an ORB-based visual SLAM using g2o [13], for BA [15] as a testbed. We replace the g2o::SparseOptimizer class (with related classes and functions in g2o performing BA) with our method, and evaluate and compare its performance with the original BA by g2o. To do this, we first run the visual SLAM with g2o on the selected sequence and extract all the runs of local BA. We use them for training our graph networks.

In the experiments reported in Sec. 4.2, we create a test dataset in the same way as above from the other sequences and report the residual Huber loss of the final estimates. In the experiments of Sec. 4.3, we report the performance of a complete SLAM system in which the g2o BA is replaced with the proposed method. We run it on the KITTI sequences from the start to the end (or until tracking is lost) and evaluate its accuracy.

We report the residual Huber loss below, which are obtained when the horizontal images size (i.e., 1,226 pixels) is normalized to 1.0. We use the Graph Nets library [5] for the implementation of the graph network. For the training with Adam optimizer, we set the initial learning rate to 1.0×10^{-2}, and the learning rate decay $= 0.99$.

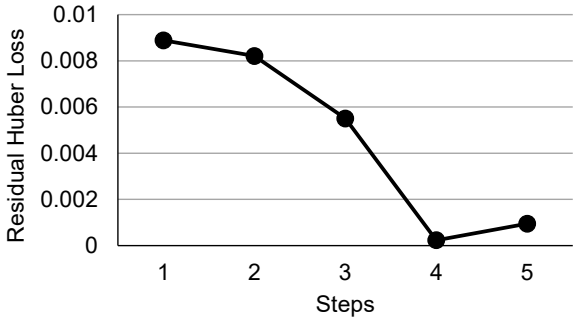


Figure 7. Residual Huber loss for different number of GN blocks.

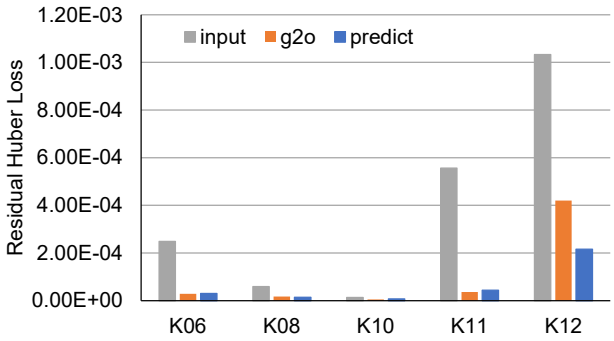


Figure 8. The averaged (residual) Huber loss over the local BA runs of each sequence for the initial value, g2o, and the proposed method.

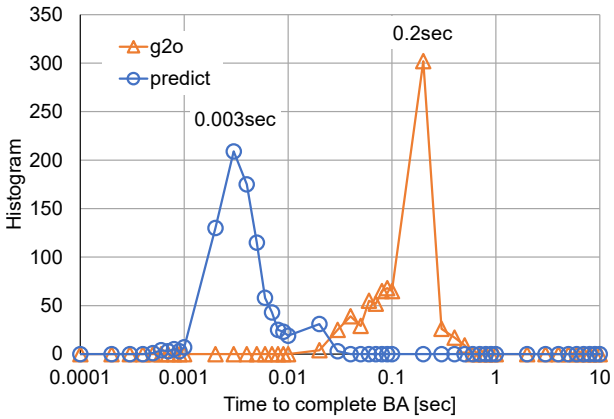


Figure 9. Computational time (erapsed real time) needed for each BA run by the proposed method and the g2o BA.

4.2. Performance as a Bundle-adjuster

As mentioned above, we first evaluate the performance of the proposed method when applied to each of the BA runs extracted from the test sequences. We use here a KITTI sequence K05 for training and K06, K08, K10, K11, and K12 for test. We chose these sequences, since they contain relatively more frequent scene changes. The former has 850 BA runs and the latter has 3030 BA runs in total.

4.2.1 GN Steps and Error Decrease

As explained in Sec. 3.3, our method uses the graph network consisting of the S-stack of GN blocks sharing the same parameters. Figure 7 shows the residual Huber loss for different S's. It decreases as S increases, but increases at $S = 5$. A similar phenomenon is reported in a previous study of applying graph networks to physics simulation [20]; a larger number of steps lead to worse results. We choose $S = 4$ from the result and use it in the experiments below.

4.2.2 Performance on Individual BA Runs

Figure 8 shows the residual Huber loss averaged over all the BA runs of each sequence for the proposed method and the original g2o BA. It can be seen that the proposed method decreases the loss for all the sequences, although it is modestly worse than the g2o BA. Note that there is a certain amount of gap between the test sequences. This evaluation validates the generalization ability of the proposed method. We obtained similar results for different combinations of sequences for training and test.

4.2.3 Computational Time

We also evaluated the computational time of our method. We measured the elapsed real-time from the start to the end of each BA run on a PC with an Intel Core i7-2600K CPU @ 3.40GHz. Figure 9 shows the histogram of the elapsed time for the proposed method and the g2o BA. The results show that our method takes only about 1/60–1/10 of the time needed by the g2o BA (0.2 $\rightarrow$ 0.003sec). Since the g2o BA needs 5–15 iterations for each run, restricting its iterations to 1/10 to save computational time is not realistic. As shown in Figure 1(b), even more mild constraint on the computational budget leads to frequent tracking failures. The proposed method runs five times faster (i.e., from 0.2 to 0.04sec in the histogram peak) than the g2o BA when it is incorporated in the full SLAM pipeline.

4.3. Performance of Complete SLAM

The performance on individual BA runs will not fully represent how well the entire SLAM system works. Thus, we test the SLAM system with the g2o BA replaced with the proposed method. This evaluation is important, since the proposed method shows slightly worse performance on individual BA runs than the original BA. The worse performance could lead to the accumulation of errors, or even tracking failures.

We evaluate here the residual Huber loss of the landmarks that are tracked at each frame. If the proposed method fails to optimize the parameters, then the residual loss will increase as time goes. Figure 10 shows an example for a sequence K07, while the graph network is trained

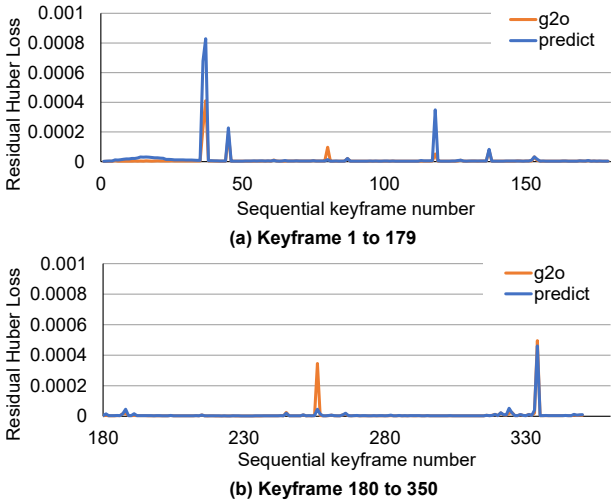


Figure 10. An example of how the (mean) residual Huber loss of the tracked landmarks varies within a sequence (KITTI K07).

on K04, K05, and K06. It is seen that the residual Huber loss temporarily increases from frame 1 to 20. However, it decreases to a reasonable level after that (i.e., from frame 21 to 35), showing that the proposed method optimizes the parameters well. It is also seen that there occasionally emerges very large errors, e.g., at frames 37, 45, 80, 118, 137, 256, and 334; these are mostly caused by large appearance changes at these frames. Our method can handle these cases properly. Overall, the residual loss is controlled to be within a low range.

Figure 11 shows the trajectory and the landmarks estimated by the SLAM systems with the original g2o BA and the proposed method. Figure 12 shows the absolute trajectory error measured using the ground truth vehicle trajectory. The RMSE for the system with the original g2o BA is 8.26m, and that for the proposed method is 11.90m. Although the proposed method yields slightly worse trajectory accuracy, the result shows that the proposed method serves as a valid bundle adjuster. Checking its performance on various combinations of sequences for trainng and test, we confirm the robustness of the proposed method at the cost of slightly lower accuracy.

5. Conclusion

In this study, we have explored the possibility of replacing the standard Levenberg-Marquardt method for BA with a learning-based approach. To precisely model the structure of the multi-view systems dealt with in local BA, we design a graph network implementing the bipartite graph composed of the nodes of keyframes and landmarks and the edges corresponding to the landmarks' visibility. We train it to predict the geometric parameters.

We design the graph network as a stack of multiple identical GN (graph network) blocks. For the design of the GN

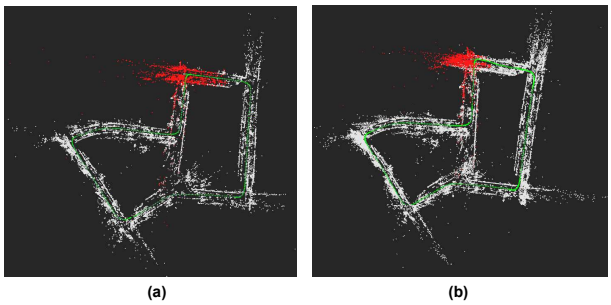


Figure 11. Reconstructed trajectories and landmarks by (a) the conventional BA (g2o) and (b) the proposed method.

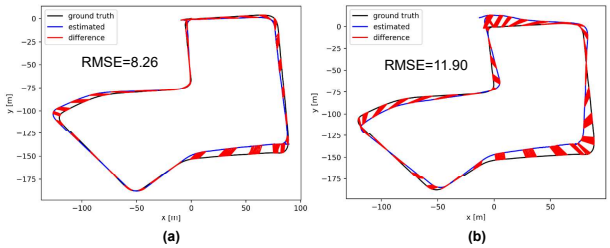


Figure 12. Errors of estimated trajectories of (a) the conventional BA (g2o) and (b) the proposed method.

block, we employ an approach inspired by the normal equation of the Levenberg-Marquardt method. We use its components (i.e., the block-diagonal elements of the Hessian and the gradients) as the 'intermediate representation' of inputs to predict the parameters. We design two vertex predictors, which predict keyframe and landmark parameters, and implement them as neural networks. We also design auxiliary components to compute the inputs to the predictors. We then train the whole graph network to predict the parameter updates that minimize the sum of reprojection errors.

We have confirmed through experiments on the KITTI dataset that the proposed method works well as a bundle adjuster. It is slightly worse in the estimation of geometric parameters than the standard BA but works robustly. We will examine the performance of the proposed method in a more variety of training and test data in the future. There remains much room for improvements such as more optimal network structures and better design of intermediate representation of inputs fed to the predictors (i.e., neural networks).

Acknowledgement

This work was partly supported by JSPS KAKENHI Grant Number 20H05952 and JP19H01110.

References

[1] Sameer Agarwal, Noah Snavely, Steven M. Seitz, and Richard Szeliski. Bundle adjustment in the large. In Kostas Daniilidis, Petros Maragos, and Nikos Paragios, editors,

Computer Vision – ECCV 2010, pages 29–42, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[3] Dan Barnes, Will Maddern, Geoffrey Pascoe, and Ingmar Posner. Driven to distraction: Self-supervised distractor learning for robust monocular visual odometry in urban environments. *CoRR*, abs/1711.06623, 2017.

[4] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, and Koray kavukcuoglu. Interaction networks for learning about objects, relations and physics. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 45094517, Red Hook, NY, USA, 2016. Curran Associates Inc.

[5] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. `https://github.com/deepmind/graph_nets.git`, 2018.

[6] D. Crandall, A. Owens, N. Snavely, and D. Huttenlocher. Discrete-continuous optimization for large-scale structure from motion. In *CVPR 2011*, pages 3001–3008, 2011.

[7] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 834–849, Cham, 2014. Springer International Publishing.

[8] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361, 2012.

[9] Radek Grzeszczuk, Demetri Terzopoulos, and Geoffrey Hinton. Neuroanimator: Fast neural network emulation and control of physics-based models. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, page 920, New York, NY, USA, 1998. Association for Computing Machinery.

[10] Y. Jeong, D. Nister, D. Steedly, R. Szeliski, and I. Kweon. Pushing the envelope of modern methods for bundle adjustment. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(8):1605–1617, 2012.

[11] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[12] Kurt Konolige. Sparse sparse bundle adjustment. In *Proceedings of the British Machine Vision Conference*, pages 102.1–102.11. BMVA Press, 2010. doi:10.5244/C.24.102.

[13] R. Kmmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. G2o: A general framework for graph optimization. In *2011 IEEE International Conference on Robotics and Automation*, pages 3607–3613, 2011.

[14] Y. Li, J. Wu, J. Zhu, J. B. Tenenbaum, A. Torralba, and R. Tedrake. Propagation networks for model-based control under partial observation. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 1205–1211, 2019.

[15] Manolis I. A. Lourakis and Antonis A. Argyros. Sba: A software package for generic sparse bundle adjustment. *ACM Trans. Math. Softw.*, 36(1), Mar. 2009.

[16] R. Mur-Artal, J. M. M. Montiel, and J. D. Tards. Orb-slam: A versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.

[17] R. Mur-Artal and J. D. Tards. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017.

[18] J. Ortiz, M. Pupilli, S. Leutenegger, and A. J. Davison. Bundle adjustment on a graph processor. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2413–2422, 2020.

[19] Ananth Ranganathan, Michael Kaess, and Frank Dellaert. Loopy sam. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, page 21912196, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[20] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter W. Battaglia. Learning to simulate complex physics with graph networks, 2020.

[21] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[22] Shinya Sumikura, Mikiya Shibuya, and Ken Sakurada. Openvslam. *Proceedings of the 27th ACM International Conference on Multimedia*, Oct 2019.

[23] K. Tateno, F. Tombari, I. Laina, and N. Navab. Cnn-slam: Real-time dense monocular slam with learned depth prediction. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6565–6574, 2017.

[24] Sen Wang, Ronald Clark, Hongkai Wen, and Niki Trigoni. Deepvo: Towards end-to-end visual odometry with deep recurrent convolutional neural networks. *CoRR*, abs/1709.08429, 2017.

[25] Ying-mei Wei, Lai Kang, Bing Yang, and Ling-da Wu. Applications of structure from motion: a survey. *Journal of Zhejiang University SCIENCE C*, 14(7):486–494, Jul 2013.

[26] C. Wu, S. Agarwal, B. Curless, and S. M. Seitz. Multicore bundle adjustment. In *CVPR 2011*, pages 3057–3064, 2011.

[27] E. Wu, L. Zhao, Y. Guo, W. Zhou, and Q. Wang. Monocular vision slam based on key feature points selection. In *The 2010 IEEE International Conference on Information and Automation*, pages 1741–1745, 2010.

[28] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *CoRR*, abs/1901.00596, 2019.

[29] Khalid Yousif, Alireza Bab-Hadiashar, and Reza Hoseinnezhad. An overview to visual odometry and visual slam: Applications to mobile robotics. *Intelligent Industrial Systems*, 1(4):289–311, Dec 2015.

[30] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Graph neural networks: A

review of methods and applications. *CoRR*, abs/1812.08434, 2018.