ADVANCED INTELLIGENT SYSTEMS
Open Access
www.advintellsyst.com

Deep Neural Network–Based Loop Detection for Visual Simultaneous Localization and Mapping Featuring Both Points and Lines

Yang Li, Chao Ping Chen, Nizamuddin Maitlo, Lantian Mi, Wenbo Zhang, and Jie Chen*

Herein, a visual simultaneous localization and mapping (SLAM) is proposed in which both points and lines are extracted as features and a deep neural network is adopted for loop detection. Its working principles, including the representation, extraction, description, and matching of lines, initialization, keyframe selection, optimization of tracking and mapping, and loop detection using a deep neural network, are set forth in detail. The overall trajectory estimation and loop detection performance is investigated using the TUM RGB-D (indoor) benchmark and KITTI (outdoor) datasets. Compared with the conventional SLAMs, the experimental results of this study indicate that the proposed SLAM is able to improve the accuracy and robustness of trajectory estimation, especially for the scenes with insufficient points. As for loop detection, the deep neural network turns out to be superior to the traditional bag-of-words model, because it decreases the accumulated errors of both the estimated trajectory and reconstructed scenes.

1. Introduction

Over the past two decades, simultaneous localization and mapping (SLAM) has been gaining ground in applications such as augmented reality[1] and autonomous driving.[2] With SLAM, not only can the trajectory of the moving object be estimated, but also the surrounding 3D scene can be reconstructed in real time.[3–5] To date, a variety of SLAMs using various sensors—such as lasers, inertial measurement units, and cameras—have been proposed.[6] Visual SLAM, among others, refers to a type of SLAM that relies on cameras. It can be divided into two subcategories, i.e., feature-based and direct.[7] For example, MonoSLAM,[8] PTAM,[9] and ORB-SLAM[10] are feature-based SLAMs and line segment detection SLAM (LSD-SLAM)[11] and DTAM[12] are direct SLAMs. Compared to the direct ones, the feature-based SLAMs are more flexible, easier to initialize, and faster to run. Plus, they are more robust against image noise and large geometric distortions. Therefore, the feature-based SLAMs are the most widely adopted in recent years.[13–15] The entire process of feature-based SLAMs usually involves extraction, description, and matching of the features for tracking trajectories of the camera, reconstruction and maintenance of the local map, loop detection, and optimization of tracking and mapping for all estimated results.

Feature-based SLAMs, which extract points as features, are also called "point-based SLAMs." Unfortunately, the performance of point-based SLAMs deteriorates and even fails for scenes with insufficient points, because it is difficult to find enough reliable points among the images for matching and tracking. In addition to points, lines are another important feature, and can also be used to describe a scene or map.[16–18] In most cases, points represent the corners while lines represent the edges. Compared to points, lines are more accurate and reliable in situations with an obstacle or a moving visual angle, and they are less sensitive to the variation of lighting conditions. Therefore, adding lines to the current point-based SLAMs e could compensate for the lack of points in some scenes.[19–21] Furthermore, the reconstructed maps of scenes comprising both spatial points and lines can provide more structural information than those comprising points only.

Loop detection is another crucial process of visual SLAM, which is responsible for recognizing the places formerly visited by computing the similarity among the images.[22–24] It is used to reduce the accumulated error of the estimated trajectories through loop fusion. The traditional method for loop detection is known as the bag-of-words (BoW) model.[25] This model extracts features from images by relative feature detection methods and clusters their descriptors as words (using the *k*-means clustering method) to build a visual dictionary.[26] Then, the corresponding words are used to represent the observed images and compute the similarity among them.[27–29] Finally, the two most

Y. Li, Prof. C. P. Chen, N. Maitlo, L. Mi, W. Zhang, J. Chen
Smart Display Lab
Department of Electronic Engineering
Shanghai Jiao Tong University
Shanghai 200240, China
E-mail: ccp@sjtu.edu.cn

The ORCID identification number(s) for the author(s) of this article can be found under https://doi.org/10.1002/aisy.201900107.

© 2019 The Authors. Published by WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim. This is an open access article under the terms of the Creative Commons Attribution License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.

DOI: 10.1002/aisy.201900107

similar but discontinuous images are identified as a loop. However, the perceptual variability is not been handled well, because images from the same scene may change in response to the light conditions or the movement of the cameras, resulting in totally different features. Therefore, when it comes to the actual applications, BoW may incorrectly detect the loops in certain scenes. A wrong detection easily produces a larger error in both the position and the orientation of the cameras, which can usually be categorized as two types.[30] One is the false positive, which means that two different but similar scenes are treated as loops. The other is the false negative, which means that two same but different-look scenes are not treated as loops. A good loop detection should be able to distinguish between these two types of situations and return correct detection results.

Due to the limitations of BoW, the accumulated error of visual SLAMs could not be reduced. To improve the accuracy, a deep neural network (DNN) for loop detection is proposed.[31] The purpose of the deep neural network is to train a classification representation form from the original sequences and represent each image in the sequences through the trained features. If a representation is carefully trained from the sequence, it can recognize whether an upcoming observation is from the same or from different scenes.

In this study, a visual SLAM is introduced in which both points and lines are exploited to enhance accuracy and robustness. A DNN is used to detect the loops in scenes and consequently reduce the accumulated error of the estimated trajectories and reconstructed scenes. The experiments are conducted on open datasets. The experimental results indicate that the proposed SLAM outperforms the traditional visual SLAMs in terms of the accuracy and robustness of both trajectory estimation and loop detection, especially in scenes with insufficient points.

2. Basic Concepts of Lines

2.1. Representation of Lines

In a 3D space, the representation of lines is more complicated than that of points. A triplet of $(x, y, z)^{\mathrm{T}}$ is sufficient to uniquely represent a point. In contrast, lines have four degrees of freedom. Based on the summary done by Bartoli and Sturm,[32] several methods have been proposed to represent lines, and two of them are carefully chosen in our solution: the Plücker line coordinate and orthonormal representation. The former is mainly exploited to initialize the newly detected lines and to describe the projection process of the lines from the 3D space to the 2D image plane, whereas the latter is used for optimization. Their representations are shown in **Figure 1**.[16]

A 3D line in Plücker line coordinates can be represented by two vectors, namely $\mathcal{L} = (\mathbf{n}^{\mathrm{T}}, \mathbf{v}^{\mathrm{T}})^{\mathrm{T}}$, where $\mathbf{n}$ is the vector normal to the interpretation plane containing the origin and the line and $\mathbf{v}$ is the direction vector of the line. Both of them are unit vectors. The Plücker line coordinate is represented by two points $\mathbf{P} = [x_1, y_1, z_1, w_1]^{\mathrm{T}}$ and $\mathbf{Q} = [x_2, y_2, z_2, w_2]^{\mathrm{T}}$ (the homogeneous coordinate) on the line in the Cartesian coordinate

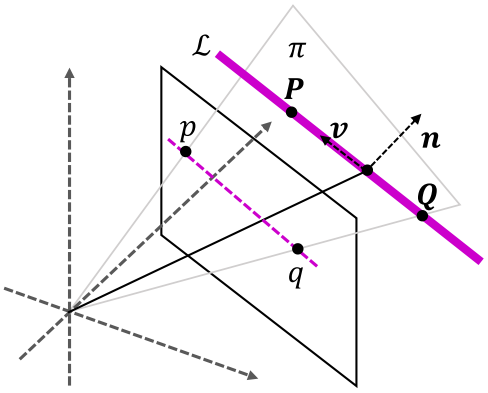

Figure 1. Two representations of a line: Plücker line coordinate and orthonormal representation. A 3D line $\mathcal{L}$ in Plücker line coordinates can be represented by two vectors ($\mathbf{n}$ and $\mathbf{v}$), and its orthonormal representation can be computed from the Plücker line coordinates by QR decomposition.

$$\mathcal{L} = \begin{bmatrix} \mathbf{P} \times \mathbf{Q} \\ w_2\mathbf{P} - w_1\mathbf{Q} \end{bmatrix} = \begin{bmatrix} \mathbf{n} \\ \mathbf{v} \end{bmatrix} \in \mathbf{R}^6 \tag{1}$$

Since vectors $\mathbf{n}$ and $\mathbf{v}$ are perpendicular to each other, there is a constraint, i.e. $\mathbf{n}^{\mathrm{T}}\mathbf{v} = 0$. This constraint makes the Plücker line coordinates unsuitable for the optimization of the visual SLAM. Therefore, another representation, the orthonormal representation, is chosen for the optimization. The orthonormal representation has at least four parameters to represent the 3D coordinate of a line. When the Plücker line coordinate of a line is defined, its orthonormal representation can be computed by the QR decomposition[33]

$$\mathcal{L} = [\mathbf{n}|\mathbf{v}] = \mathbf{U}\begin{bmatrix} \omega_1 & 0 \\ 0 & \omega_2 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} \frac{\mathbf{n}}{\|\mathbf{n}\|} & \frac{\mathbf{v}}{\|\mathbf{v}\|} & \frac{\mathbf{n}\times\mathbf{v}}{\|\mathbf{n}\times\mathbf{v}\|} \end{bmatrix} \begin{bmatrix} \|\mathbf{n}\| & 0 \\ 0 & \|\mathbf{v}\| \\ 0 & 0 \end{bmatrix} \tag{2}$$

and

$$\mathbf{W} = \begin{bmatrix} \omega_1 & -\omega_2 \\ \omega_2 & \omega_1 \end{bmatrix} \tag{3}$$

where $\mathbf{U}$ and $\mathbf{W}$ are the 3D and 2D rotation matrices, respectively. Inversely, the orthonormal representation can also be converted back to the Plücker line coordinates by

$$\mathcal{L} = (\mathbf{n}^{\mathrm{T}}, \mathbf{v}^{\mathrm{T}})^{\mathrm{T}} = (\omega_1 u_1^{\mathrm{T}}, \omega_2 u_2^{\mathrm{T}})^{\mathrm{T}} \tag{4}$$

where u_{i} is the ith column of $\mathbf{U}$, and ω_{i} is the parameter of $\mathbf{W}$. Because of the limitation of orthonormal representation in projection and transformation, it is only used in the optimization, whereas the Plücker line coordinates are used to represent the lines in other processes. When the external parameters—the pose ($\mathbf{R}$, $\mathbf{t}$) of the camera motion—are determined, the coordinate transformation of lines between the last frame i and the current frame j can be described in Equation (5) by the Plücker line coordinates, which is a linear operation, formulated as

© 2019 The Authors. Published by WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

$$\begin{bmatrix} n_j \\ v_j \end{bmatrix} = \begin{bmatrix} R_{ji} & [t_{ji}]_x R_{ji} \\ 0 & R_{ji} \end{bmatrix} \begin{bmatrix} n_i \\ v_i \end{bmatrix} \tag{5}$$

2.2. Extraction of Lines

Extracting the representative lines to represent each image of sequences can substantially reduce the computational complexity of SLAM. A fast line detection method, known as "line segment detection," which only has linear $O(n)$ time complexity, is exploited.[34] LSD detects lines through the gradient with a false detection control, and it can automatically detect the correct lines even for noisy images and complicated scenes without parameter tuning. However, LSD suffers from several problems in practical applications. For example, a complete line may be marked as several different lines in the image. Thus, directly employing the pristine LSD method in the SLAM to extract lines may result in matching and tracking failures. To improve the accuracy and robustness of line detection, an optimized LSD method is proposed.[21] If two lines detected by LSD are of the same direction and their endpoints are very close (within the specified threshold), they will be judged as the same line, and their endpoints will be merged. Consequently, the estimated trajectory error of the LSD method caused by line mismatching can be decreased.

2.3. Description of Lines

The description of lines is the foundation of line matching. To achieve correct and robust matching for a camera's pose estimation, the line-based eight-directional histogram feature (LEHF)[35] is chosen to describe the lines detected by LSD, which is based on the mean-standard deviation line descriptor (MSLD)[36] and uses a strategy similar to scale-invariant feature transform (SIFT).[37] The LEHF provides a fast and efficient way to describe detected lines and to match the corresponding lines between the 2D image plane and the 3D scene space. It can track the lines accurately and robustly even in scenes with local occlusion, rotation changes, image blur, and illumination changes.

LEHF descriptors describe the detected lines in terms of the gradient histograms of eight directions. First, a certain number of points around the line direction and the vertical direction of the line are selected, and their gradient vectors are calculated by differential values of x and y directions. Then, the gradient histogram of the eight directions can be computed through merging these gradient vectors. Finally, the rotation invariant of lines can be obtained by rectifying the direction of gradient vectors.

2.4. Matching of Lines

A sound matching strategy can improve the accuracy, robustness, and latency of SLAM. 2D–3D correspondences are established by matching the LEHF descriptors to estimate the current pose of the camera. To match the lines between adjacent images, the lines tracked in the 3D scene space in the previous frame are projected to the current 2D image plane, and the matched lines between the descriptors of projected lines and observed lines are found. According to a specific reprojection error threshold calculated in random sample consensus (RANSAC), each 3D line projected to the image plane will be labeled with several states: inlier, outlier, or out of the image plane.[35] In our solution, only the inliers are considered as valid lines, and other lines are discarded. The line matching process is shown in **Figure 2**.

To reduce the computational complexity, the making range of the line is restricted, and only the lines observed within the threshold are selected to calculate the correlative descriptors, which are represented by the red dashed ones in Figure 2. LEHF distances of these 2D lines are computed, and the lines matched with the 2D line corresponding to the 3D line should have the minimum Euclidean distance between LEHFs.

3. Deep Neural Network for Loop Detection

The proposed DNN for loop detection can be decomposed into two parts: 1) image feature training, in which an autoencoder is used to train the representative features of images, and [2]) loop detection, in which loops are detected by using a similarity or difference matrix.

3.1. Feature Training

In the proposed DNN-based loop detection, a stacked autoencoder[38] is adopted to train features from original images and to represent each image using the first hidden layer of the network. The process of training image features is shown in **Figure 3**.

Before training, the original images are divided into 10×10 patches. After being vectorized, the patches are set as vectors $\{x | x \in \mathbf{R}^{100}\}$ and are fed into the autoencoder to train the representative features. The autoencoder is an unsupervised training method of a DNN. It mainly contains three layers: 1) the first layer is the input layer x that directly connects to the vectorized original images; 2) the second layer is the hidden layer h; and 3) the third layer is the output layer y, and this layer will be discarded after the feature training process is completed. According to the definition of the autoencoder, the results of the output

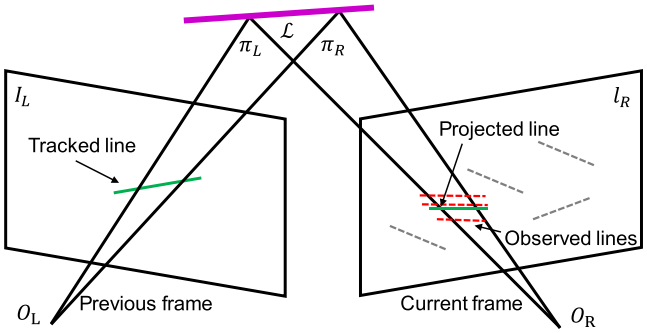

Figure 2. Matching process of lines of the adjacent images: 1) projecting the tracked 3D lines in the previous frame to the current 2D image plane and 2) finding the matched lines between the descriptors of projected lines and observed lines within the threshold, which are represented by the red dashed lines, and are to be used to compute the LEHF descriptors and to match with the descriptor of the projected line.

© 2019 The Authors. Published by WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

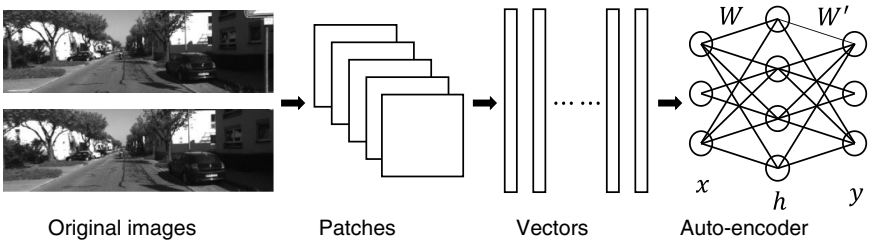

Figure 3. Process of training the image features. The deep neural network—the autoencoder—is adopted to train the features of images. Before training, the original images are divided into 10 × 10 patches. After being vectorized, the patches are set as vectors: $\{x|x \in \mathbf{R}^{100}\}$ and are fed into the autoencoder to train the representative features of images.

layer can usually recover the input as much as possible. A response a of the former layer with a weight vector $\mathbf{w}$ and a bias b through the activation function is computed in each unit of the layers. The activation function, i.e., sigmoid σ, written as

$$a = \sigma(\omega^{\mathrm{T}}x + b) = \frac{1}{1 + \exp\left(-\sum_{j=1}^{n} \omega_j x_j - b\right)} \tag{6}$$

is selected and added to increase the nonlinearity of the network. The process of feature training is modeled as an optimization problem, whose loss function is the distance between the input value x and the output value y. The iteration functions

$$(W^*, b^*) = (W, b) - \eta\left(\frac{\partial d}{\partial W}, \frac{\partial d}{\partial b}\right) \tag{7}$$

are conducted with the trained parameters (weight vector $\mathbf{w}$ and bias b), and the purpose of the iteration is to obtain the optimal network parameters, where the parameter η in Equation (7) is the learning rate and the d is the loss function if the input $x \in (0, 1)$, given by

$$d = KL(x, y) = \sum_{i=1}^{n} x_i \log\frac{y_i}{x_i} + (1 - x_i)\log\frac{1 - x_i}{1 - y_i} \tag{8}$$

To obtain more meaningful features of the network for the loop detection, the loss function of the autoencoder is optimized from three additional aspects: 1) denoising—as the images are usually affected by undesired noise, a corruption is added to the input x, which randomly masks a certain percentage of the input to zero; 2) sparsity—because the dimension of the hidden layer h is larger than the input x, a sparse structure is trained to avoid the overfitting; and 3) continuity—because it is the continuity during the movement of the camera. The denoising and sparsity are mainly aimed at the autoencoder, whereas the continuity is unique to the SLAM. These three additional penalty items are added to the original autoencoder. The corresponding loss function is represented by

$$J = d(x, g_\theta f_\theta(\widetilde{x}))) + \lambda c_{\mathrm{s}} + \gamma c_{\mathrm{c}} \tag{9}$$

where

$$d(x, g_\theta f_\theta(\widetilde{x})) = \sum KL(x, g_\theta f_\theta(\widetilde{x})) \tag{10}$$

$$c_{\mathrm{s}} = \sum_{i=1}^{nh} ||h_i - s_{\mathrm{h}}||_1 \tag{11}$$

$$c_{\mathrm{c}} = \frac{1}{k}\sum ||H - \widetilde{H}||_2 \tag{12}$$

in which $\widetilde{x}$ is the corrupted input by denoising. λ and γ are the respective penalty weights defined to balance the effect of the sparsity and continuity, and both of them are hyperparameters of the DNN.

3.2. Loop Detection

After training the DNN, the trained features are collected to construct a representation of the scenes. The features can be obtained from the response of the hidden layer and by discarding the output layer of the network. In this way, the images are represented in another form, which is difficult to intuitively understand but can be recognized easily and effectively by the computer. According to the principle that a similar input will output similar features, for any two images in the trajectories of camera movement, the difference matrix of two arbitrary scenes (m, n) can be computed from

$$\mathbf{D}(m, n) = ||\mathbf{z}^{\mathrm{T}}(h_m - h_n)||_1 \tag{13}$$

where $\mathbf{D}$ is a difference matrix and $\mathbf{z}$ is a weight matrix to reduce the effect of high average response

$$z_i = \log\frac{h}{h_i}, h = \sum_{j=1}^{n_h} h_j \tag{14}$$

According to the properties of the difference matrix, a more obvious matrix can be obtained by normalization

$$\bar{D}(m, \cdot) = \frac{\mathbf{D}(m, \cdot)}{\max \mathbf{D}(m, \cdot)} \tag{15}$$

and the rank can be reduced through the singular value decomposition. In the matrix of the decomposition, the larger eigen $k - 1$ values are discarded, and the effect of scene ambiguity can be reduced. The matrix used to detect the loops is formulated as

© 2019 The Authors. Published by WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

$$\mathbf{D}_R = \sum_{i=k}^{N} \lambda_i v_i v_i^{\mathrm{T}} \tag{16}$$

If $\mathbf{D}_R(m,n)$ is large, two scenes are thought to be dissimilar. Conversely, if $\mathbf{D}_R(m,n)$ is small, two scenes are thought to be similar.

4. Proposed SLAM

4.1. Architecture

A visual SLAM based on both points and lines is proposed to estimate the trajectory of the camera and to reconstruct unknown scenes. The DNN-based loop detection, on the other hand, is adopted to recognize the previous scenes. The proposed SLAM consists of the main process of the visual SLAM. The point-based ORB-SLAM[10] is used as the basic framework, and four mainly parallel threads are retained, on top of which the expanding strategies are implemented for points and lines, and loop detection.

Figure 4 outlines the architecture of the proposed SLAM, which mainly consists of four main threads. Thread 1. TRACKING: through this thread, the extraction and tracking of points and lines are completed; the camera's localization is obtained and the local map is tracked; and whether the current frame is a keyframe is determined. This thread exploits the optimization to minimize the reprojection error, and only the camera's pose in the local map is used as the state variables. Thread 2. LOCAL MAPPING: the local maps and keyframes are managed and optimized, including keyframe inserting and culling, and feature creating and culling. The 3D spatial features, i.e., points and lines, are assigned as optimization variables. Thread 3. LOOP DETECTION: the loops are detected by the proposed DNN, and the accumulated error through the loop fusion is consequently corrected. Thread 4. GLOBAL OPTIMIZATOIN: the purpose of this thread is to reduce the accumulated errors so as to correct the camera trajectory.

Figure 5 shows the computing process of the proposed SLAM. Two independently parallel threads for points and lines, respectively, are supposed to make reasonable use of the computing resources of the platform so as to decrease the computational complexity. After extracting and describing, the points and lines are used to carry out the feature matching, which is followed by the camera pose evaluation and scene reconstruction.

4.2. Initialization

The initialization is a procedure of the tracking, and its purpose is to compute the relative camera pose between two images and retrieve an initial map composed of points and lines. The points and lines in 2D image planes are projected back into the 3D space by the geometry relationship in the initialization process.

To reduce the computational complexity and increase the efficiency, the points are preferentially used to initialize. When the point-based initialization fails, the lines then step in to restart the initialization. If there are not enough valid points and lines, the whole initialization process will fail and the reference frame

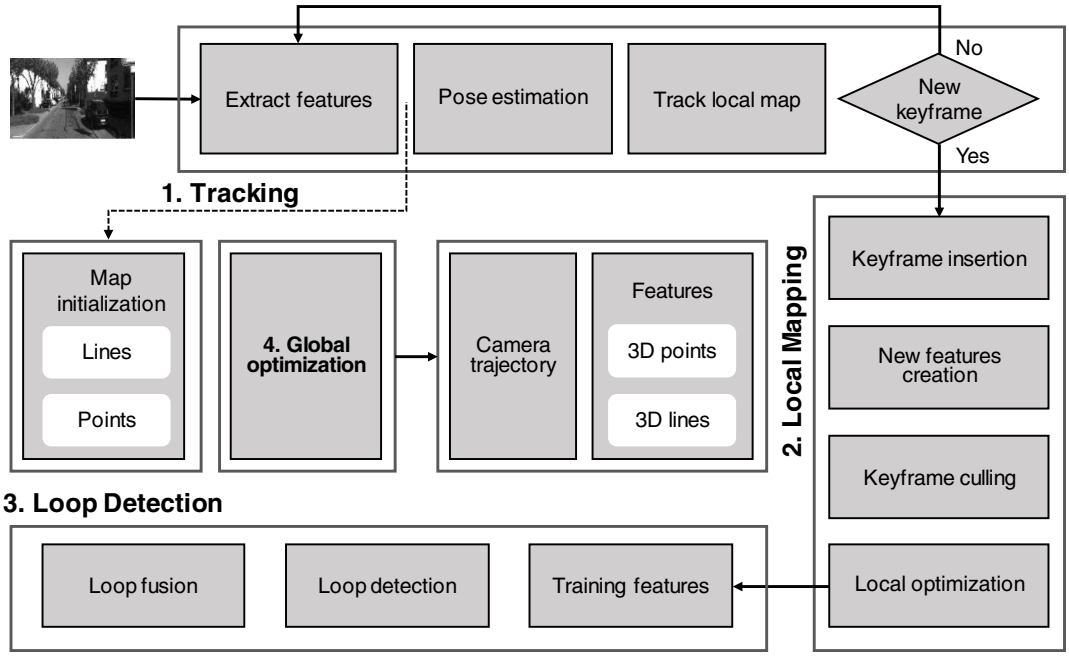

Figure 4. Architecture of the proposed SLAM. The point-based ORB-SLAM is used as the basic framework, and the expanding strategies are adopted based on it. The four parallel threads are retained—1) tracking: complete the extraction and tracking of features, obtain the camera's localization, track the local map, and finally determine whether the current frame is a keyframe. 2) Local mapping: generate local maps and manage the keyframes, including keyframe inserting and culling and feature creating. The 3D spatial features, i.e., points and lines, are used as optimization variables. 3) Loop detection: detect the loops with the deep neural network and correct the accumulated error through the loop fusion. 4) Global optimization: the purpose of this thread is to reduce the accumulated errors so as to correct the camera trajectory.

© 2019 The Authors. Published by WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

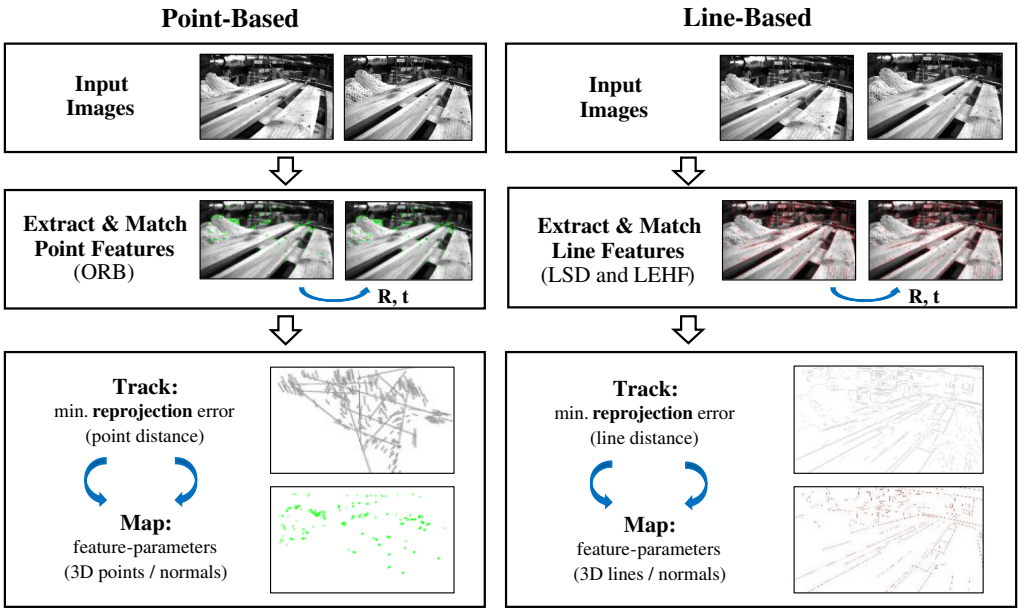

Figure 5. Computing process of both the point-based and the line-based threads. These two independently parallel threads are utilized for extraction, description, matching, and tracking, and then the camera pose evaluation and surrounding scene reconstruction can be achieved.

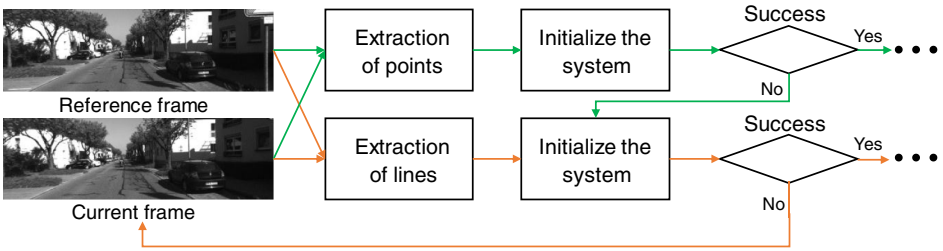

Figure 6. Initialization process with points or lines. The green line represents the initialization with points, which is prioritized to be used, while the orange one represents the initialization with lines. If there are not enough points and lines in the frame, the initialization will fail and the reference frame will need to be reset.

needs to be reset. The entire initialization process is plotted in **Figure 6**. The green line represents the initialization process of points, while the yellow one represents the initialization process of lines.

The point-based initialization is shown in **Figure 7**, where two scenes need to be considered. When using points, a planar scene is assumed by a homography matrix, and a nonplanar scene is assumed by a fundamental matrix. The homograph matrix $\mathbf{H}_{cr}$ and fundamental matrix $\mathbf{F}_{cr}$ are calculated in parallel as

$$x_c = \mathbf{H}_{cr} x_r = x_c^{\mathrm{T}} \mathbf{F}_{cr} x_r \tag{17}$$

where x_c and x_r are the matching points in the current frame and the reference frame, respectively. The score S_M for the model $\mathbf{M}$ (the $\mathbf{M}$ represents the homograph matrix or fundamental matrix) is computed by

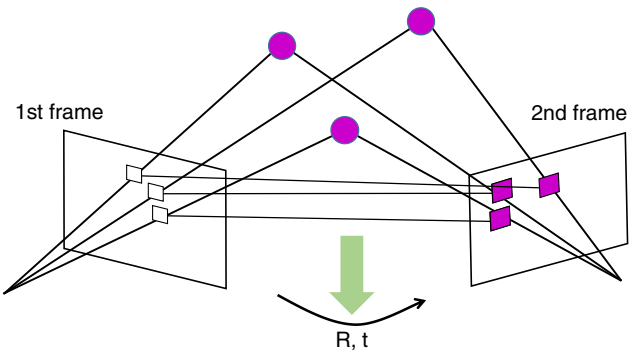

Figure 7. Initialization by points from two consecutive images. There are two scenes that need to be considered when using points; i.e., a planar scene is assumed by a homograph matrix and a nonplanar scene is assumed by a fundamental matrix. A robust heuristic parameter is defined by computing these two scenes to judge which one of the matrices can be used to compute the camera's pose: **R** and **t**.

© 2019 The Authors. Published by WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

$$S_{\mathrm{M}} = \sum_i (\rho_{\mathrm{M}}(d^2_{\mathrm{cr}}(x^i_{\mathrm{c}}, x^i_{\mathrm{r}}, \mathbf{M})) + \rho_{\mathrm{M}}(d^2_{\mathrm{rc}}(x^i_{\mathrm{c}}, x^i_{\mathrm{r}}, \mathbf{M})) \tag{18}$$

where the ρ_{M} is

$$\rho_{\mathrm{M}} d^2 = \begin{cases} \Gamma - d^2 & if \quad d^2 < T_{\mathrm{M}} \\ 0 & if \quad d^2 > T_{\mathrm{M}} \end{cases} \tag{19}$$

A robust heuristic parameter R_{H}, defined as

$$R_{\mathrm{H}} = \frac{S_{\mathrm{H}}}{S_{\mathrm{H}} + S_{\mathrm{F}}} \tag{20}$$

is used to obtain the specific model. For example, $R_{\mathrm{H}} > 0.45$ indicates that the captured image is planar and has low parallax, and the homograph matrix will be selected to complete the initialization. Otherwise, the fundamental matrix will be chosen.

The initialization process of lines is shown in **Figure 8**. Three consecutive images containing the same line are used for initialization. The rotation between the consecutive camera poses is assumed to be small and equally continuous. These three camera rotations are marked as $R_1 = \mathbf{R}^{\mathrm{T}}$, $R_2 = \mathbf{I}$, and $R_3 = \mathbf{R}$. Under the assumption in the consecutive three frames of Figure 8, there is a constraint

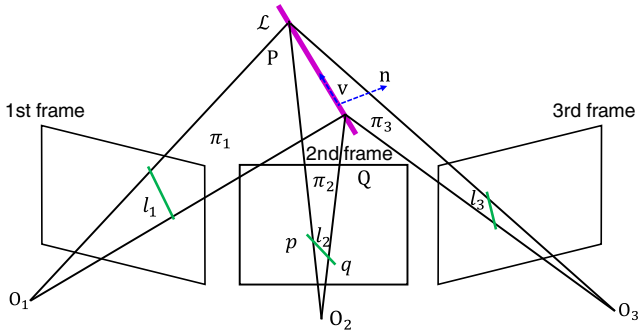

Figure 8. Initialization by lines from three consecutive images containing the same line. The rotation between the consecutive camera poses is assumed to be small and equally continuous. Through the constraints among these images, the camera pose **R** could be computed.

$$I_2^{\mathrm{T}}((\mathbf{R}^{\mathrm{T}} I_1) \times (\mathbf{R} I_3)) = 0 \tag{21}$$

In addition, for small rotations, **R** is represented as

$$\mathbf{R} = \begin{pmatrix} 1 & -r_3 & r_2 \\ r_3 & 1 & -r_1 \\ -r_2 & r_1 & 1 \end{pmatrix} \tag{22}$$

The transformation relationship can be computed by the total number of matched features in these three consecutive images, and then each value of **R** can be obtained.

4.3. Keyframe Selection

When the current image no longer matches the latest keyframe, a new one should be selected and inserted into the local map. Since the ORB-SLAM only focuses on the points, several changes are made according to the structural properties of the lines. A scale factor α[39] is added to describe the ratio of the motion estimation entropy between the previous keyframe $i-u$ and the current one i to the entropy between the previous frame $i-u$ and its consecutive frame $i-u+1$, which is written as

$$\alpha = \frac{h(\xi_{i-u}, \xi_i)}{h(\xi_{i-u}, \xi_{i-u+1})} \tag{23}$$

where the ξ represents the Lie group of the relative pose,[40] and the entropy at each time is given by

$$h(\xi) = 3(1 + \log(2\pi)) + 0.5 \log(|\Sigma_\xi|) \tag{24}$$

If the value of α is lower than the preestablished threshold, which is set as 0.9, the current frame is inserted to the map as a new keyframe, i.e., the input of the local mapping.

4.4. Graph Optimization

Graph optimization is used to estimate each state, including the trajectory of the camera and the localization of the detected points and lines in the scenes. The graph consists of vertices and edges. The vertices represent the variables that need to be optimized, while the edges represent the corresponding relations among these optimized variables. **Figure 9** is an illustration of

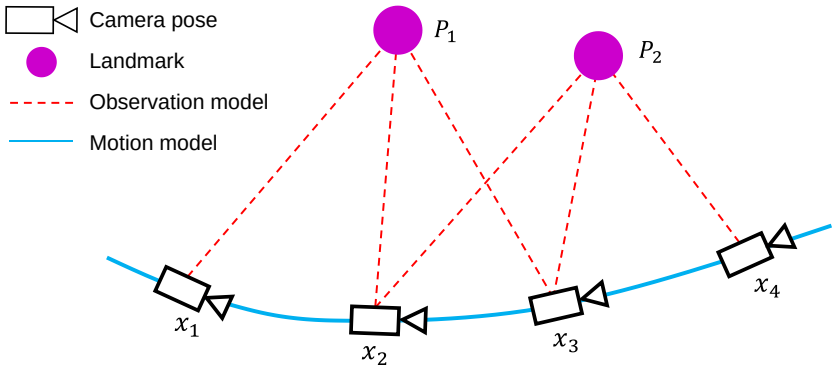

Figure 9. Graph optimization with both points and lines. The circles denote the landmarks observed by the camera, and they constitute the vertices of the graph along with the camera poses. The blue lines are the motion model of the camera. The red dashed lines are the observation model of each image. These two kinds of lines represent the optimized edges of the graph.

© 2019 The Authors. Published by WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

the graph optimization.[41] The circles are the landmarks observed by the camera, and they constitute the vertices of the graph along with the camera poses. The blue lines are the motion model of the camera, and the red dashed lines represent the observation model of each image. These two kinds of lines represent the optimized edges of the graph.

The detected landmarks consist of both points and lines, and the edges involve the reprojection error of tracked points and lines in the scenes. As shown in **Figure 10**, an observation model is built to describe the reprojection error of the points and lines. For points, the camera pose is marked as T_{cw}, and the intrinsic parameter of the camera is marked as K. A 3D point X in the world coordinate is projected to the camera coordinate X_c, and then the point x' in the image coordinate can be computed according to the intrinsic K. Finally, the error between the matched point and the projected point can be calculated. The whole process is mathematically expressed as

$$X_c = T_{cw}X \tag{25}$$

$$x' = KX_c \tag{26}$$

$$e_p = ||x' - x|| \tag{27}$$

The reprojection error of the lines can be obtained by a similar process. A space line $\mathcal{L}$ is projected into the image plane, which is marked as l', and the reprojection error is represented by the two endpoints of the line x_s and x_e in the image. The whole process is mathematically expressed as

$$\mathcal{L}_c = \begin{bmatrix} R_{cw} & [t_{cw}]_X R_{cw} \\ 0 & R_{cw} \end{bmatrix} \mathcal{L} \tag{28}$$

$$l' = \begin{bmatrix} f_v & 0 & 0 \\ 0 & f_u & 0 \\ -f_v c_u & f_u c_v & f_u f_v \end{bmatrix} n_c \tag{29}$$

$$e_l = d(\mathbf{z}, \mathbf{l}') = \left[\frac{\mathbf{x}_s^T \mathbf{l}'}{\sqrt{l_1^2 + l_2^2}}, \frac{\mathbf{x}_e^T \mathbf{l}'}{\sqrt{l_1^2 + l_2^2}} \right] \tag{30}$$

Ideally, the projected points and lines coincide with the observed points and lines. However, due to the errors of both projection and observation, they are not completely coincident. For this reason, both errors are treated as edges between the vertices in the graph optimization. These two edges are determined by

$$ep_{k,i} = x_{k,i} - KT_{kw}X_{w,i} \tag{31}$$

and

$$el_{k,j} = d(z_{k,j}, \mathcal{K}n_c[\mathcal{H}_{cw}\mathcal{L}_{w,j}]) \tag{32}$$

There is an assumption that the errors satisfy the Gaussian distribution, and the covariance matrices of the points and lines are Σ_p and Σ_l, respectively. The final objective function C to be optimized is the sum of all the point projection errors and the line projection errors, i.e.

$$C = \sum_{k,i} \rho_p (Ep_{k,i}^T \sum p_{k,i}^{-1} Ep_{k,i}) + \sum_{k,j} \rho_l (El_{k,j}^T \sum l_{k,j}^{-1} El_{k,j}) \tag{33}$$

where the ρ_p and ρ_l are the Huber cost functions, which are used to reduce the outliers of the objective function and to keep the observed features as close to the projected features as possible.

5. Results and Discussion

Our experiments are composed of two parts: the accuracy of the loop detection in the proposed SLAM based on the DNN is evaluated in the first part and the second part demonstrates the accuracy and robustness of the trajectory estimation. The experiments are conducted on the TUM RGB-D (indoor) benchmark[42] and KITTI (outdoor) datasets.[43]

5.1. Loop Detection

The accuracy of the DNN-based loop detection in the proposed SLAM is validated through a set of experiments. TensorFlow, an open-source software library of deep learning,[44] is used to build the autoencoder, and the datasets come from the TUM RGB-D benchmark, which contains both the RGB-D images and their ground truth trajectories. Each sequence of the images is

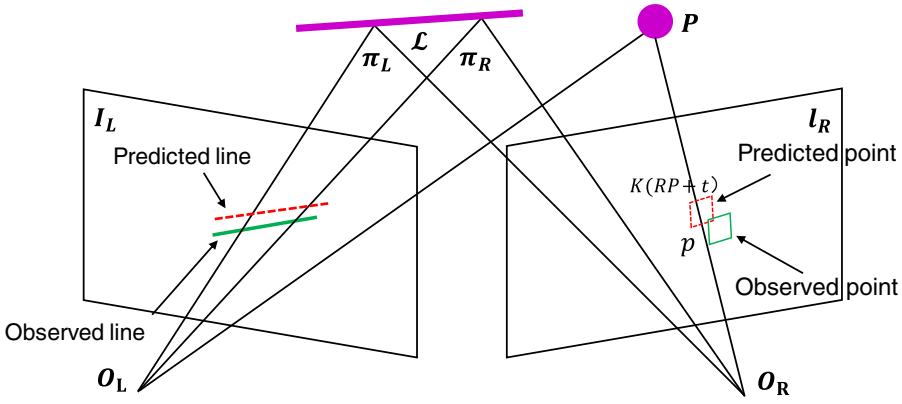

Figure 10. Observation model with both points and lines. The detected landmarks consist of both points and lines, and the edges involve the reprojection error of tracked points and lines in the scenes. The observation model is built to describe the reprojection error of points and lines and to minimize the distance between the projected features and the observed features.

© 2019 The Authors. Published by WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

randomly sampled, and 30% of them are used for training, while 20% of them are used for testing the ability of the network to represent other images. To obtain accurate and robust features of the trained network, several hyperparameters related to the DNN are used in the process of training visual features, which are itemized in **Table 1**.

The autoencoder comprises a total of three hidden layers, and the units in each layer are 200, 50, and 50, respectively. First, the influence of some hyperparameters on the performance of the neural network is evaluated. The corruption level, which corresponds to the denoising, is added to the weight matrix **W** of the first hidden layer. Since noises always exist in the images, adding the corruption is indispensable for increasing the robustness and preventing the network from being overfitted.

In the absence of corruption, namely, when the corruption level is 0, in the weight matrix from the hidden layer, the meaningful information in the image will be clouded by the noise, as shown in **Figure 11**a. When corruption is present, as shown in Figure 11b–d, sparse edge features of the images can be observed. Out of the different corruption levels, the optimal corruption level is obtained at 12%.

Figure 12 shows the average response of the images in the hidden units. The impact of adding the corresponding penalties on the solution is judged from the perspective of sparsity. With a sparse penalty in the loss function, most of the useful information in the trained results is preserved, and the redundant features in the network, which are irrelevant to the current training task, are discarded. Adding the sparsity penalty can prevent the network from being overfitted and eliminate the redundant features in the network. In addition, it can reduce the

Table 1. Hyperparameters used for training features.

Parameter	Value	Description
n_{hidden}	[200, 50, 50]	Units in each layer
η	0.1	Learning rate
γ	0.002	Penalty factor of continuity
λ	0.005	Penalty factor of sparsity
l_c	0.12	Corruption level
s_{h}	0.05	Sparsity threshold
n_{batch}	10	Batch size of stochastic gradient descent
n_{epoch}	20	Times of iteration

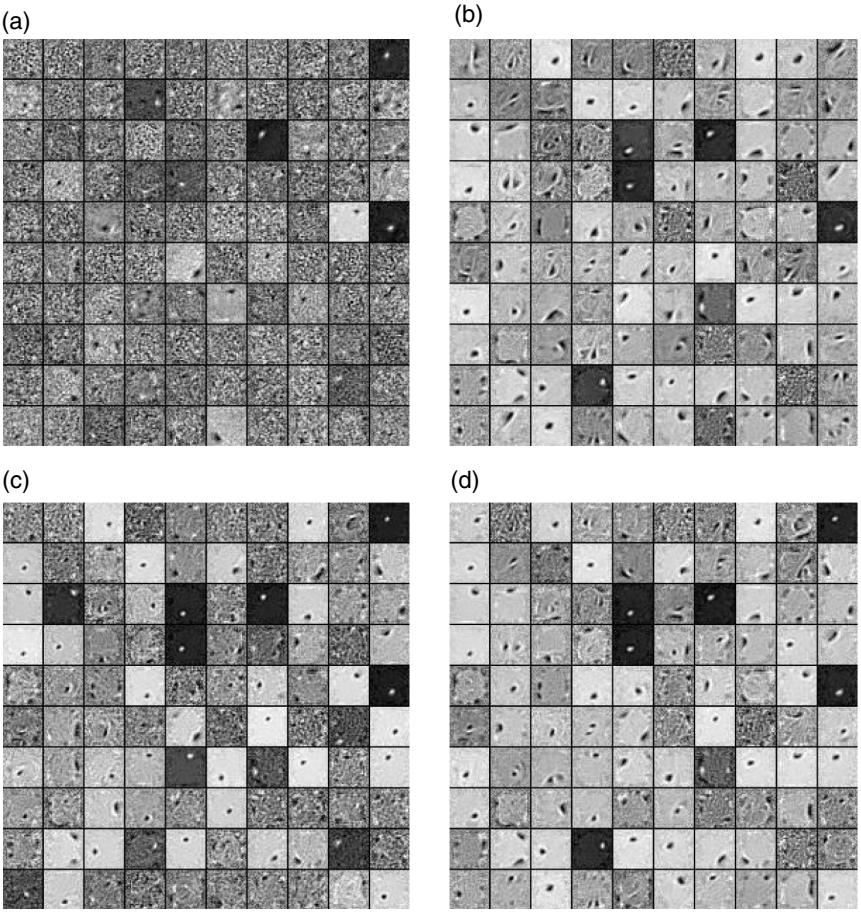

Figure 11. Visualization of the structures trained (10 × 10 units in the result are selected) from the hidden layer with four different corruption levels: a) 0%, b) 5%, c) 12%, and d) 30%. Experiments show that although some details of the images are slightly lost at the 12% level, it can nevertheless obtain the useful feature representation.

© 2019 The Authors. Published by WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

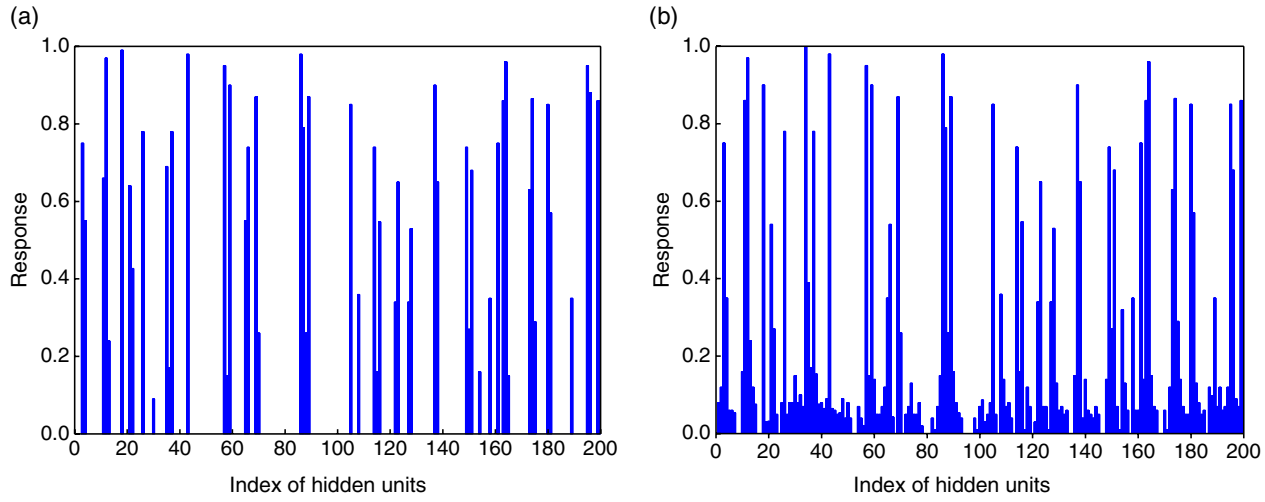

Figure 12. Average response of the hidden units in the images: a) with the sparsity penalty and b) without the sparsity. Adding the sparsity penalty can prevent the model from being overfitted. It can also effectively eliminate the redundant features in the network, which are unrelated to the current training task.

computational complexity in the training of the network parameters and make the network more interpretable.

The difference matrix computed by the first hidden layer response is used for detecting the loops, and a value lower than the threshold d indicates that a loop candidate has been detected. The similarity matrix, which is calculated by points and lines, is used to fuse the loops between these candidate frames and current frames. **Figure 13** shows the comparison results of the feature graph and their differences in two scenes. In the same scene, the differences of the feature representation are very small, whereas in different scenes, the feature differences are relatively large.

To evaluate the performance of trained features for loop detection, the precision-recall curve is calculated. Precision-recall curves of two competing methods on the TUM RGB-D benchmark are shown in **Figure 14**. It can be seen that the performance of the DNN is better than that of the BoW. In practical scenes, once a loop is detected, its neighboring frames will also be detected with the current frame; therefore, a recall rate approaching 50% is sufficient. When the recall equals 50%, the accuracy of

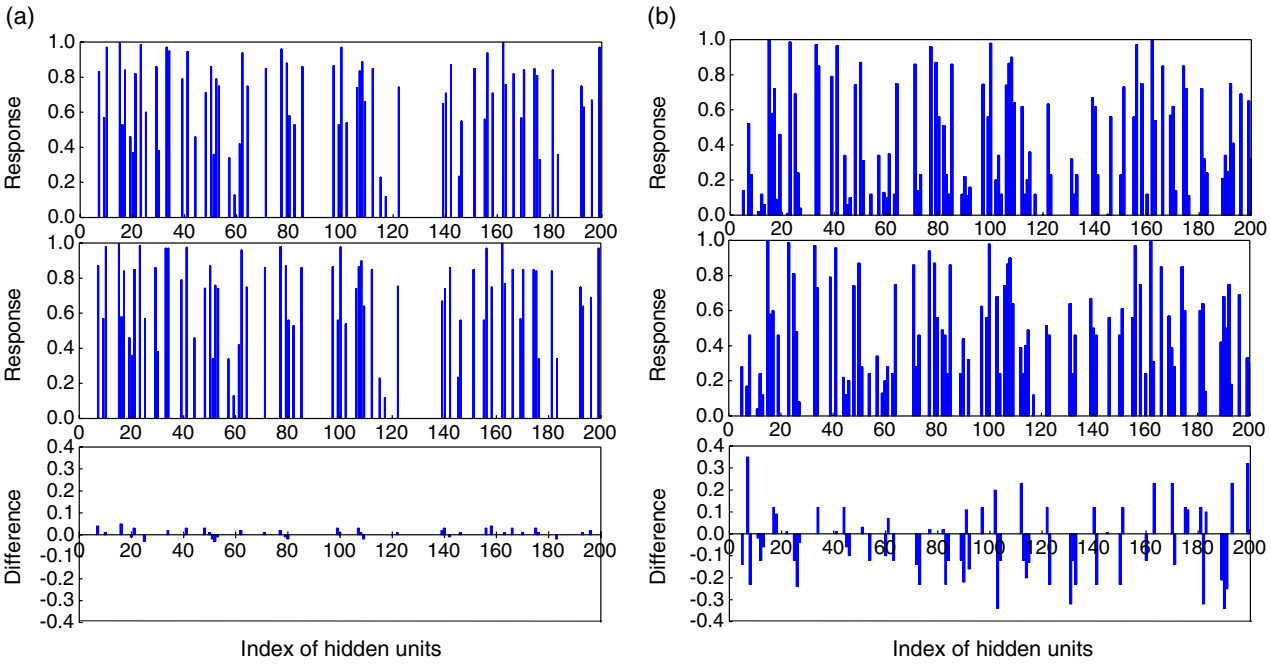

Figure 13. Comparison of the feature graphs in two scenes. a) In the same scenes, the differences of the feature representation are very small. b) In the different scenes, the feature differences of the feature representation are relatively large.

© 2019 The Authors. Published by WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

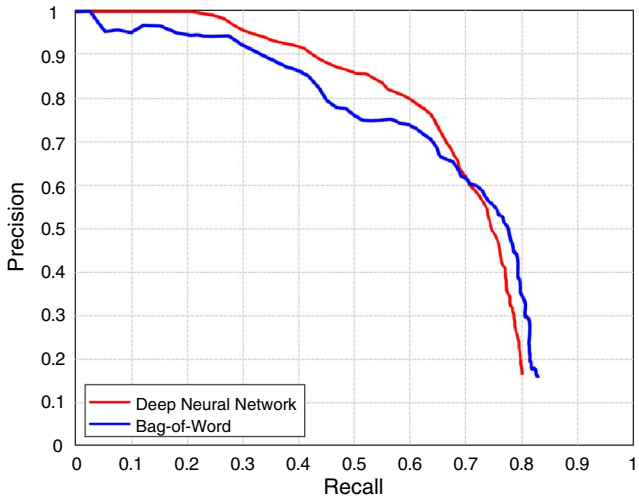

Figure 14. Precision-recall curves of the DNN and BoW. The red curve is the result of loop detection with the DNN. The blue curve is the result of loop detection with the BoW. It can be seen that the performance of the DNN is better than that of the BoW. When the recall equals 50%, the precision of the DNN is 10% higher than that of the BoW.

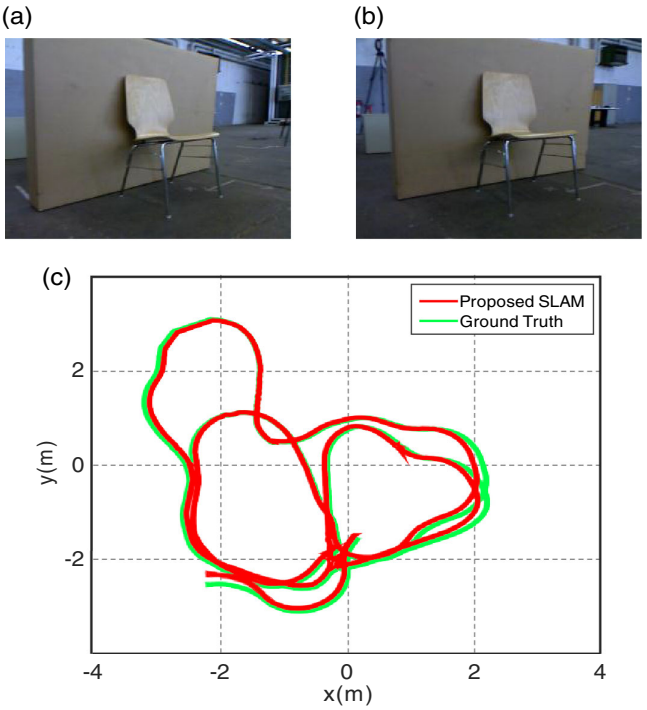

Figure 15. Estimated trajectories with loops. Image (a) and image (b) are captured in the same location at different time. c) the estimated trajectory versus the ground truth. The proposed method can accurately detect the loops of the scene, and its estimated trajectory overlaps well with the ground truth.

the DNN-based loop detection is about 10% higher than that of the traditional BoW.

The sequence fr2/pioneer_slam in the TUM RGB-D benchmark is used to evaluate the performance of the loop detection. The estimated trajectory and the ground truth are compared in **Figure 15**. It can be seen that the proposed method accurately detects the loops of the scene, and the estimation trajectory agrees well with the ground truth.

The time of detecting the loops is another crucial indicator. Computed on a CPU of Intel Core i7–6600U@2.60 Ghz, the times for loop detection of the BoW and DNN are compared in **Table 2**, where the detecting time is the average time for detecting the loops in the sequence of fr2/poineer_slam inclusive of 2198 images. The detection process is to compare the current frame with all the keyframes, which are maintained in the reconstructed map. The time of the BoW method is 10.24 ms, whereas that of the DNN is 14.76 ms, which is only 4.53 ms longer than the BoW.

Table 2. Average time of loop detection.

Method	Time [ms]
BoW	10.24
Deep neural network	14.76

5.2. Localization

Localization is evaluated in terms of the accuracy and robustness of trajectory estimation. Based on the indoor datasets known as the TUM RGB-D benchmark, the proposed SLAM is compared with other visual SLAMs, including PTAM, LSD-SLAM, and ORB-SLAM, to evaluate their localization accuracies. The image sequences that are not suitable for monocular visual SLAM in the TUM RGB-D benchmark are removed, and consequently only ten relative sequences are selected. The absolute median RMS error is used as a metric for the comparison; it can be computed through a Python script provided by the benchmark. Before calculating the error, similarity transformation is applied to all the trajectories to align the data. For the sake of reducing the amount of calculation, only the keyframes extracted from the image sequence are calculated. **Table 3** summarizes the results of the absolute median RMS errors of the camera trajectories executed over five times in each sequence. In all these ten indoor

Table 3. Absolute median RMS errors (cm) of camera trajectories on the TUM RGB-D benchmark.

Sequence	ORB-SLAM	PTAM	LSD-SLAM	Proposed SLAM
fr1_xyz	1.41	1.23	9.21	1.34
fr1_desk	1.69	x	10.65	1.66
fr1_floor	5.64	x	38.07	3.43
fr2_xyz	0.64	0.49	2.15	0.56
fr2_desk_person	0.63	x	31.73	6.34
fr2_360_kidnap	4.99	2.63	x	3.78
fr3_office	4.35	x	38.53	2.13
fr3_sit_xyz	0.80	0.95	7.84	0.66
fr3_walk_xyz	1.64	x	12.44	1.78
fr3_walk_halfsph	1.92	x	x	1.70

© 2019 The Authors. Published by WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

scenes, the proposed SLAM exhibits a high accuracy in the trajectory of the camera compared with others. In particular, in the floor and office scenes containing abundant lines, the localization accuracy of the proposed method is much higher than that of the point-based ORB-SLAMs. By contrast, PTAM was unreliable in several scenes because it failed in six out of ten sequences. In addition, LSD, which is a direct-based method, also loses its trajectory in two scenes, and its localization accuracy is much lower than that of our SLAM.

As for the outdoor performance, KITTI, an outdoor dataset with the ground truth, is used for the experiments. It contains 22 sequences collected from outdoor autonomous driving scenes. To highlight the performance of the SLAM after adding the lines into the features, four representative sequences 01, 03, 07, and 09 with sufficient lines are picked as test sequences. The trajectories obtained by the proposed SLAM, ORB-SLAM, and points and lines based stereo visual odometry (PLSVO)[20] are compared with the ground truth, as shown in **Figure 16**. The trajectories of our SLAM overlap better with the ground truth than those of the ORB-SLAM and PLSVO. Furthermore, the proposed SLAM exhibits good robustness of loop detection at the same time. As can be seen from Figure 16a, our SLAM works well in sequence-01 (highway), whereas the ORB-SLAM fails. This is because the number of points appearing on the highway scene is insufficient. On the contrary, our SLAM being added with the lines, the number of features available for matching and tracking increases substantially. Especially in the linear running stage, the estimated trajectory of our SLAM agrees well with the ground truth, since there is an adequate amount of lines. In addition, PLSVO, which adopts the violent matching between the extracted features in the visual odometry, suffers greatly from the accumulated errors due to the absence of loop detection. Though able to work in sequence-01, the trajectories of PLSVO deviate far from the ground truth as the distance increases.

Table 4 lists the results of the absolute median RMS errors of keyframe trajectories in each sequence, from which it can be seen that the proposed SLAM shows better accuracy and robustness of localization than the traditional point-based ORB-SLAM

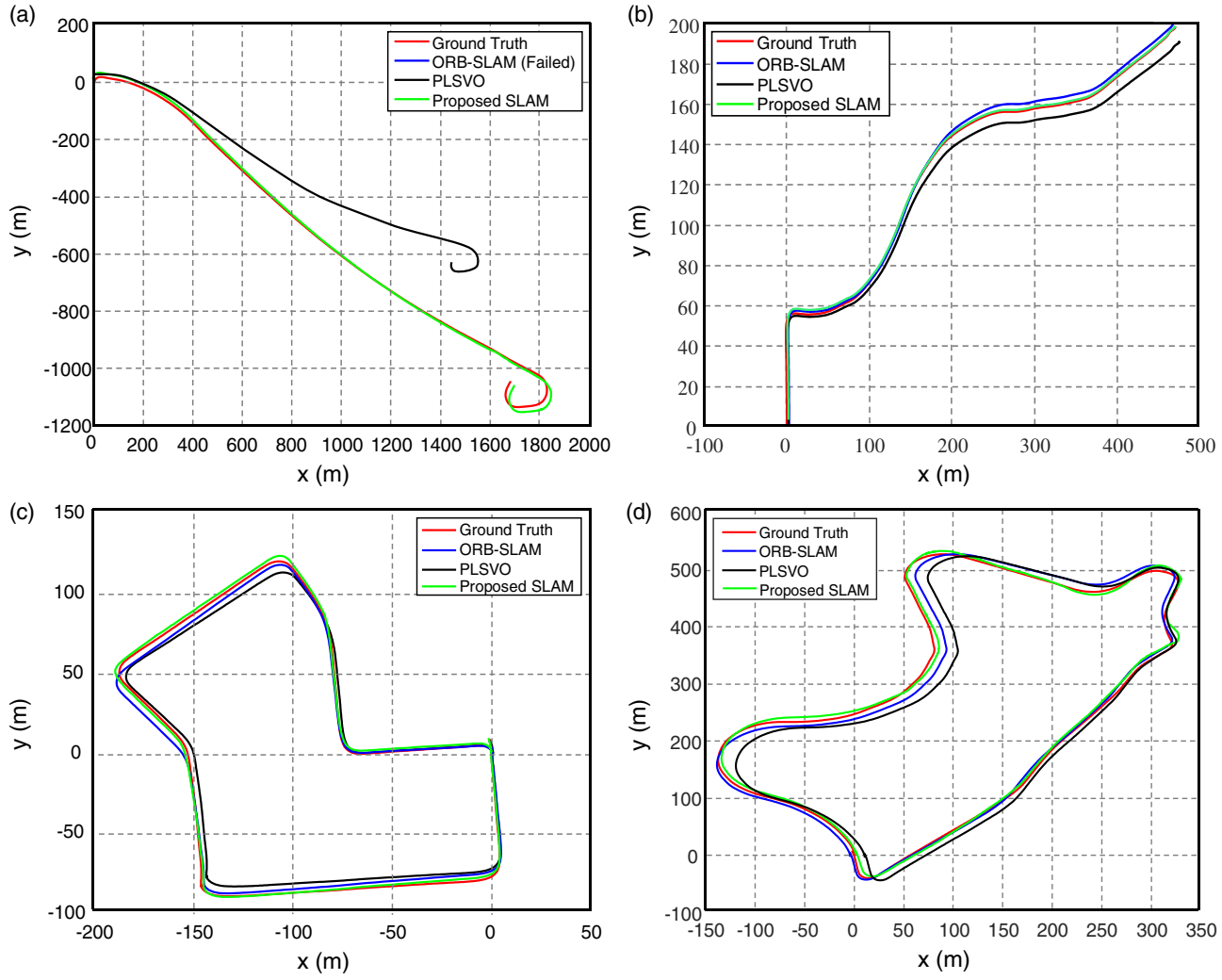

Figure 16. Estimated trajectories of the proposed SLAM, PLSVO, ORB-SLAM, and ground truth for the four representative sequences from KITTI: a) sequence-01, b) sequence-03, c) sequence-07, and d) sequence-09. The trajectories estimated by our SLAM match better with the ground truth than other methods. It should be noted that, for sequence-01 (highway), our SLAM works well, whereas the ORB-SLAM fails and the PLSVO deviates substantially.

© 2019 The Authors. Published by WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

Table 4. Absolute median RMS errors (m) of keyframe trajectories on the KITTI dataset.

Sequence	ORB-SLAM	PLSVO	Proposed SLAM
01	N/A	314.07	19.48
03	2.63	23.41	2.39
07	3.45	12.81	3.93
09	7.72	25.96	5.27

and direct-based PLSVO in both indoor and outdoor scenes. However, the proposed SLAM needs to consume more time than both ORB-SLAM and PLSVO for the extra computational load incurred by lines. By average, the time consumptions of ORB-SLAM, PLSVO, and our SLAM are 0.5, 1, and 1.2 s, respectively.

6. Conclusions

In this article, a visual SLAM is proposed in which both points and lines are extracted as features to estimate the trajectory of the camera and reconstruct the map of scene in real time. In addition, a deep neural network—an autoencoder—is adopted for the loop detection. The overall performance regarding the trajectory estimation and loop detection has been investigated using the TUM RGB-D (indoor) benchmark and KITTI (outdoor) datasets. Our experimental results indicate that the proposed SLAM, compared to the conventional SLAMs, shows better accuracy and robustness of trajectory estimation than the traditional point-based or direct-based SLAMs, especially for the scenes with insufficient points. As for the loop detection, the DNN turns out to be superior to the traditional BoW, as it could decrease the accumulated error of the estimated trajectories and reconstructed scenes. Merits aside, we shall also mention that our SLAM can underperform in the presence of noise and non-well-defined geometries and even fail if both points and lines are insufficient.

Acknowledgements

This work was supported by the National Natural Science Foundation of China under grant 61831015 and the Science and Technology Commission of Shanghai Municipality under grant 19ZR1427200.

Conflict of Interest

The authors declare no conflict of interest.

Keywords

deep neural networks, loop detection, navigation, robot control, simultaneous localization and mapping

Received: September 12, 2019
Revised: October 30, 2019
Published online: December 13, 2019

[1] V. Gay-Bellile, S. Bourgeois, D. Larnaout, M. Tamaazousti, *Fundamentals of Wearable Computers and Augmented Reality*, 2nd ed. (Ed: W. Barfield), CRC Press, Boca Raton, FL **2016**, Ch. 17.
[2] H. Lategahn, A. Geiger, B. Kitt, in *Proc. IEEE Int. Conf. on Robotics and Automation*, IEEE, New York, NY **2011**.
[3] H. Durrant-Whyte, T. Bailey, *IEEE Robot. Autom. Mag.* **2006**, *13*, 99.
[4] Y. Li, B. Yu, C. P. Chen, N. Maitlo, W. Zhang, L. Mi, in *Proc. OSA 3D Image Acquisition and Display: Technology, Perception and Applications*, OSA, Washington, DC **2018**.
[5] B. Yu, Y. Li, C. P. Chen, N. Maitlo, J. Chen, W. Zhang, L. Mi, in *Proc. SID Display Week*, SID, Los Angeles, CA **2018**.
[6] T. Bailey, H. Durrant-Whyte, *IEEE Robot. Autom. Mag.* **2006**, *13*, 108.
[7] P. de la Puente, D. Rodriguez-Losada, *Robot. Auton. Syst.* **2015**, *63*, 80.
[8] A. J. Davison, I. D. Reid, N. D. Molton, O. Stasse, *IEEE Trans. Pattern Anal. Mach. Intell.* **2007**, *29*, 1052.
[9] G. Klein, D. Murray, in Proc. 6th IEEE and ACM Int. Symp. on Mixed and Augmented Reality, Nara, Japan, November **2007**.
[10] R. Mur-Artal, J. M. M. Montiel, J. D. Tardos, *IEEE Trans. Robot.* **2015**, *31*, 1147.
[11] J. Engel, T. Schöps, D. Cremers, in *Proc. 13th European Conf. on Computer Vision*, Springer, Zurich **2014**.
[12] R. A. Newcombe, S. J. Lovegrove, A. J. Davison, in *Proc. IEEE Int. Conf. on Computer Vision*, IEEE, Barcelona **2011**.
[13] J. J. Leonard, H. F. Durrant-Whyte, I. J. Cox, *Int. J. Robot. Res.* **1992**, *11*, 286.
[14] M. W. M. G. Dissanayake, P. Newman, S. Clark, H. F. Durrant-Whyte, M. Csorba, *IEEE Trans. Robot. Automat.* **2001**, *17*, 229.
[15] J. E. Guivant, F. R. Masson, E. M. Nebot, *Robot. Auton. Syst.* **2002**, *40*, 79.
[16] C. Rother, in *Proc. 9th IEEE Int. Conf. on Computer Vision*, IEEE, Los Alamitos, CA **2003**.
[17] T. Lemaire, S. Lacroix, in *Proc. IEEE Int. Conf. on Robotics and Automation*, IEEE, Piscataway, NJ **2007**.
[18] G. Zhang, J. H. Lee, J. Lim, I. H. Suh, *IEEE Trans. Robot.* **2015**, *31*, 1364.
[19] Y. Lu, D. Song, in *Proc. IEEE Int. Conf. on Computer Vision*, IEEE, New York, NY **2015**.
[20] R. Gomez-Ojeda, J. Gonzalez-Jimenez, in *Proc. IEEE Int. Conf. on Robotics and Automation*, IEEE, Piscataway, NJ **2016**.
[21] X. Zuo, X. Xie, Y. Liu, G. Huang, in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, IEEE/RSJ, Vancouver **2017**.
[22] A. Angeli, D. Filliat, S. Doncieux, J.-A. Meyer, *IEEE Trans. Robot.* **2008**, *24*, 1027.
[23] M. Cummins, P. Newman, *Int. J. Robot. Res.* **2011**, *30*, 1100.
[24] M. Labbe, F. Michaud, *IEEE Trans. Robot.* **2013**, *29*, 734.
[25] D. Gálvez-López, J. D. Tardos, *IEEE Trans. Robot.* **2012**, *28*, 1188.
[26] Z. Zhang, *IEEE Trans. Pattern Anal. Mach. Intell.* **2000**, *22*, 1330.
[27] D. Falliat, in *Proc. IEEE Int. Conf. on Robotics and Automation*, IEEE, Piscataway, NJ **2007**.
[28] B. Williams, G. Klein, I. Reid, *IEEE Trans. Pattern Anal. Mach. Intell.* **2011**, *33*, 1699.
[29] H. Kwon, K. M. A. Yousef, A. C. Kak, *Robot. Auton. Syst.* **2013**, *61*, 749.
[30] G. Csurka, C. R. Dance, L. Fan, J. Willamowski, C. Bray, in *Proc. 8th European Conf. on Computer Vision*, Springer, Prague **2004**.
[31] X. Gao, T. Zhang, *Auton. Robot.* **2017**, *41*, 1.
[32] A. Bartoli, P. Sturm, *Int. J. Comput. Vis.* **2004**, *57*, 159.
[33] A. Bartoli, P. Sturm, *Comput. Vis. Image Underst.* **2005**, *100*, 416.
[34] R. G. von Gioi, J. Jakubowicz, J.-M. Morel, G. Randall, *IEEE Trans. Pattern Anal. Mach. Intell.* **2010**, *32*, 722.
[35] K. Hirose, H. Saito, in *Proc. 23rd British Machine Vision Conf.*, BMVA, Guildford, UK **2012**.
[36] Z. Wang, F. Wu, Z. Hu, *Pattern Recognit.* **2009**, *42*, 941.
[37] D. G. Lowe, *Int. J. Comput. Vis.* **2004**, *60*, 91.

© 2019 The Authors. Published by WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

[38] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.-A. Manzagol, *J. Mach. Learn. Res.* **2010**, *11*, 3371.
[39] C. Kerl, J. Sturm, D. Cremers, in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, IEEE/RSJ, Piscataway, NJ **2013**.
[40] V. S. Varadarajan, *Lie Groups, Lie Algebras, and Their Representations*, 1st ed., Springer, New York, NY **1984**.
[41] J.-A. Fernández-Madrigal, J. L. B. Claraco, *Simultaneous Localization and Mapping for Mobile Robots: Introduction and Methods*, IGI Global, Hershey, PA **2012**.
[42] J. Sturm, N. Engelhard, F. Endres, W. Burgard, D. Cremers, in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, IEEE/RSJ, New York, NY **2012**.
[43] A. Geiger, P. Lenz, C. Stiller, R. Urtasun, *Int. J. Robot. Res.* **2013**, *32*, 1231.
[44] K. Wongsuphasawat, D. Smilkov, J. Wexler, J. Wilson, D. Mane, D. Fritz, D. Krishnan, F. B. Viegas, M. Wattenberg, *IEEE Trans. Vis. Comput. Graph.* **2018**, *24*, 1.

© 2019 The Authors. Published by WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim