Lidar-Monocular Visual Odometry using Point and Line Features

Shi-Sheng Huang[1], Ze-Yu Ma[1], Tai-Jiang Mu[1], Hongbo Fu[2], and Shi-Min Hu[1]

***Abstract*— We introduce a novel lidar-monocular visual odometry approach using point and line features. Compared to previous point-only based lidar-visual odometry, our approach leverages more environment structure information by introducing both point and line features into pose estimation. We provide a robust method for point and line depth extraction, and formulate the extracted depth as prior factors for point-line bundle adjustment. This method greatly reduces the features' 3D ambiguity and thus improves the pose estimation accuracy. Besides, we also provide a purely visual motion tracking method and a novel scale correction scheme, leading to an efficient lidar-monocular visual odometry system with high accuracy. The evaluations on the public KITTI odometry benchmark show that our technique achieves more accurate pose estimation than the state-of-the-art approaches, and is sometimes even better than those leveraging semantic information.**

I. Introduction

Lidar-visual odometry has been an active research topic due to its wide applications such as robotics, virtual reality, and autonomous driving, etc. The combination of visual sensors and lidar sensors as lidar-visual odometry achieves the benefits of both types of sensors,and thus has been gaining more and more research interests in the computer vision, computer graphics, and robotics communities nowadays [1].

Recently, tightly coupled fusion algorithms like V-LOAM [2] show impressive performance for visual-enhanced lidar odometry. Those algorithms follow the odometry framework without using SLAM techniques such as bundle adjustment and loop closure. The subsequent works such as DEMO [3], LIMO [4] and DVL-SLAM [5], [6] utilize bundle adjustment techniques to achieve much higher motion estimation accuracy. Although the camera tracking front-end may be different for these approaches (with DEMO [3] and LIMO [4] following the *feature-based* visual SLAM like ORB-SLAM2 [7], and DVL-SLAM [5] following the *direct-based* visual SLAM (as like DSO [8]) for camera tracking), all of them take sparse point based bundle adjustment as back-end to correct camera tracking in an accurate way. However, the accuracy of those point-only systems is still not very satisfactory and some of them like LIMO [4] require extra semantic information as input, which, however, is computationally expensive to obtain.

The recent techniques for 3D reconstruction using structure from motion ([9], [10]) and accurate motion estimation of visual SLAM systems ([11], [12]) show that utilizing more structural information from real environments, such as line features, leads to more accurate camera pose estimation. Besides, line features have conditional benefits that they are less sensitive with problems such as noise [13], wide range of view angle [14], and motion blur [15], which are the main drawbacks for the point-only systems such as LIMO [4]. This motivates us to combine point and line features together for an accurate lidar-visual odometry system. However, there is an open issue for line-based visual SLAM systems since line-based 3D triangulation can be sensitive during camera tracking [16], thus causing an unstable visual SLAM system without satisfactory pose estimation accuracy. Although line features might richly exist in various scene environments (especially in urban environments), it is nontrivial to directly adopt line features for the lidar-visual odometry.

In this paper, we provide a robust and efficient lidar-monocular visual odometry method combining both point and line features in a purely geometric way to extract more structural information from scene environments than point-only systems. More specifically, our system fuses the point and line features as landmarks during camera tracking and formulates the point-based and line-based landmarks' reprojection errors as factors for bundle adjustment in the back end. During sensor fusion, we provide a robust method to extract the depth of the points and lines from the lidar data, and use the depth prior to guide camera tracking. In this way, we avoid the creation of 3D landmarks solely based on the possibly ambiguous 3D triangulation, especially for the 3D lines. The depth prior is also formulated as prior factors in the point-line bundle adjustment to further improve the pose estimation accuracy.

Besides, to overcome the scale drift in the frame-to-frame odometry, we recover the scale in each keyframe using a scale correction optimization before bundle adjustment. In this way, we achieve an efficient but driftless lidar-visual odometry system. Benefited from a rich set of structural 3D landmarks, we achieve much higher pose estimation accuracy compared with the other purely geometric approaches (such as DEMO [3] and DVL-SLAM [5]) evaluated on the KITTI odometry benchmark [17], and comparable accuracy with LIMO [4] but without using any extra semantic information. To the best of our knowledge, we give the first efficient lidar-monocular odometry approach using the point and line features together in a purely geometric way.

II. Methodology

Preprocessing. Given a monocular image sequence and a lidar sequence, we assume that the intrinsic and extrinsic parameters of the two sensors have been calibrated, and both two sensor data have been temporally aligned. We set the

[1]Shi-Sheng Huang, Ze-Yu Ma, Tai-Jiang Mu, and Shi-Min Hu are with the Department of Computer Science and Technology, Tsinghua University and BNRist, China.

[2]Hongbo Fu is with the School of Creative Media, City University of Hong Kong, China.

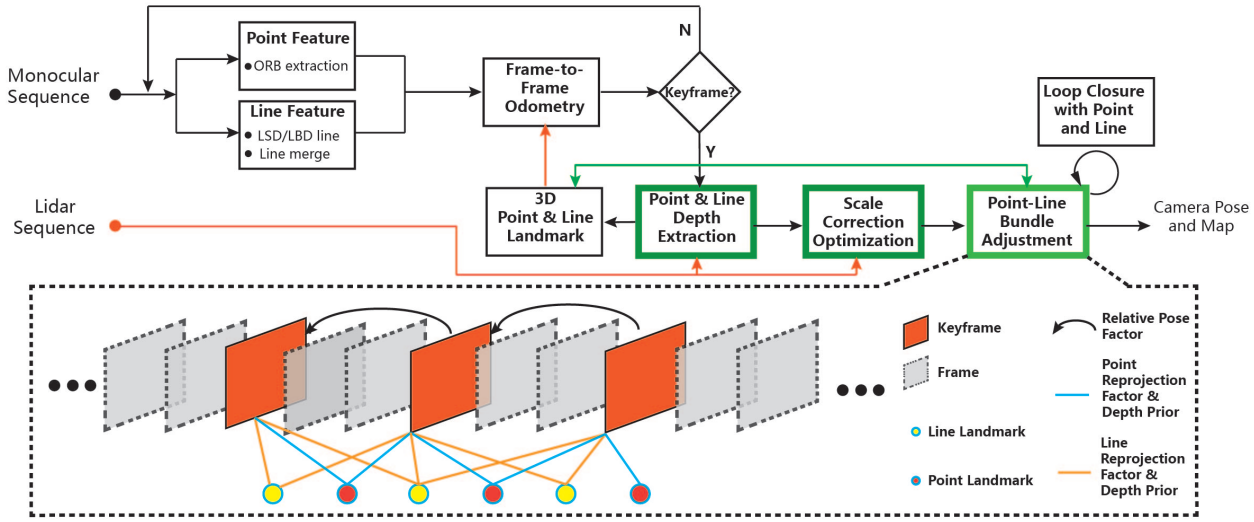

Fig. 1. The system framework of our approach. Given the input monocular image sequence and lidar sequence, we extract the point and line features for each image, and track image frames using our frame-to-frame odometry with scale corrected by our scale correction optimization. For each keyframe, the depth priors for point and line landmarks are extracted and fed to the point-line bundle adjustment. Loop closure with the point and line features is used for further pose estimation correction. The graph structure for bundle adjustment is illustrated in the dashed box with different factors. The components highlighted in green are our main contribution in this work.

camera's local coordinate as the body coordinate, and the world coordinate as the beginning of the body coordinate.

Overview. Fig. 1 shows the framework of our system, which contains three running threads: a motion tracking thread (front-end), a bundle adjustment thread (back-end), and a loop closure thread. The front-end first extracts the point and line features in every frame (Section II-A), then estimates the feature's depth prior in every keyframe (Section II-B), and finally estimates the camera poses with a frame-to-frame odometry (Section II-C). A scale correction optimization (Section II-D) is performed to correct the scale drift after the frame-to-frame odometry. The back-end performs point-line bundle adjustment with point-line constraint factors (Section II-E). A bag-of-words based loop closure [18] with point and line features is also performed to further refine the poses of keyframes (Section II-F).

A. Feature Extraction

Point Feature. Various point features (SIFT, SURF, ORB etc) can be used to serve as tracking features. We adopt the Oriented fast and Rotated BRIEF (ORB) feature as the point feature for efficiency as done in ORB-SLAM2 [7]. During detection, the ORB features are required to be evenly distributed in the image as much as possible.

Line Feature. For each image, we use the popular line feature detector, Line Segment Detector (LSD) [19], to detect line segments, for which the Line Band Descriptor (LBD) [20] will be extracted.

The LSD algorithm often breaks a long line segment into several short ones as shown in Fig. 2 (Left). Besides, some detected line segments may be quite near, such as the border edges of a thin area as shown in Fig. 2 (Right). The existence of such detected line segments often makes the subsequent line matching task complex, thus increasing the uncertainty

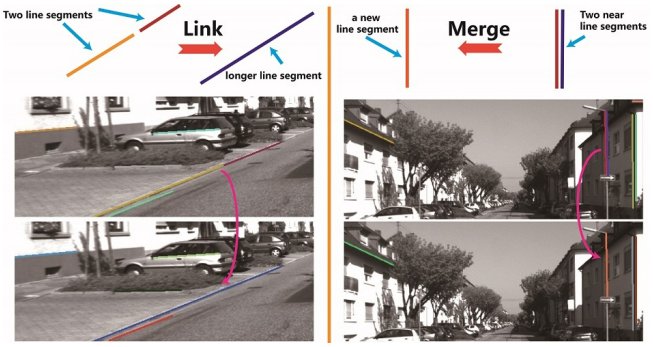

Fig. 2. We chain short line segments into a longer one (Left), or merge near line segments into a new one (Right) to enhance the quality of lines returned by LSD. The images come from KITTI dataset sequence 00.

of 3D line triangulation. To address these issues we propose to improve the results of the LSD algorithm by merging such "bad" line segments, as shown in Fig. 2. More specifically, for a line segment pair (l_i, l_j), if only one endpoint of l_i is near to l_j's endpoint, we link l_i and l_j to form a longer line segment. If both endpoints of l_i are near to those of l_j and the distance between l_i and l_j is under a given threshold (10px in our implementation), we merge l_i and l_j as a new line segment. The LBD line descriptor is also updated for the newly linked or merged line segment.

B. Point and Line Depth Extraction

In this section we describe a method to extract the point and line depth from the lidar data. Here the depth for a 2D point feature means the depth of its corresponding 3D point, and for a 2D line feature means the depth of two endpoints' corresponding 3D landmarks

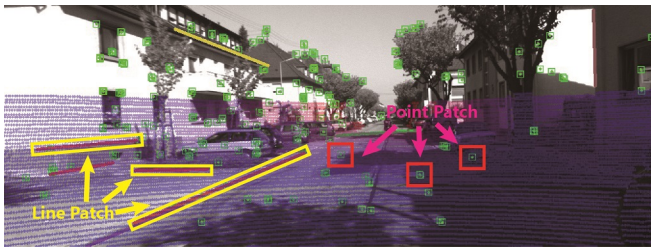

Fig. 3. A simple illustration for point and line depth extraction. After the sparse lidar data (grey points) is aligned to the image plane, the point depth and line depth are extracted in the point neighbor patch and line neighbor patch separately.

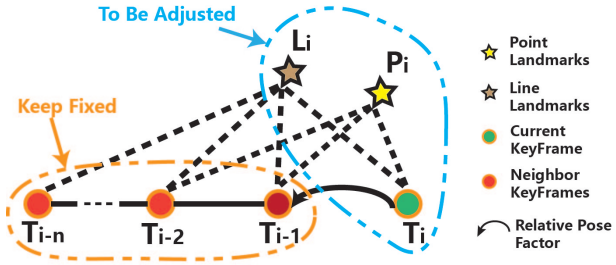

Fig. 4. The graph structure for scale correction optimization.

For each detected 2D point or line segment, its depth prior is estimated using a neighbor patch separately as shown in Fig. 3. More specifically, the lidar points belonging to a point patch are first segmented into foreground points and background points using the method introduced by Tateno et al. [21]. Then a plane is fitted using the foreground points, and the depth of the feature point is that of the intersection point between the point ray and the plane. For the lidar points $P = \{p_i\}$ belonging to a line patch of an image line segment l_j, we first estimate its corresponding 3D line L_j^* by minimizing the following energy

$$L_j^* = \arg\min_{L_j} \sum_i d(p_i, L_j) + e(L_j, l_j), \tag{1}$$

where $d(p_i, L_j)$ is the Euclidean distance between lidar point p_i and L_j, and $e(L_j, l_j)$ is the line reprojection error as described in Section II-E. Here for initialization, we fit the lidar points belonging to the line patch with a 3D line L_j' as the initialization of L_j. After L_j^* is estimated, the two end points priors (as shown in Fig. 5(left)) can be obtained by using endpoint trimming as described in [10], [11]. Besides, for points and line segments locating beyond the view of LiDAR data, we use the point triangulation and line trimming for the depth prior estimation [10], [11].

We adopt the Plücker coordinates [9] to represent a 3D line during the motion tracking, and the orthonormal representation [9] as the minimum parameters. The optimization can be efficiently solved by Levenberg-Marquardt algorithm[22].

C. Frame-to-Frame Odometry

We use a pure visual Frame-to-Frame Odometry to estimate very frames' camera pose, which is more efficient than other ICP-based odometry like V-LOAM[2]. For a new frame F_i, we estimate its camera pose $T_i \in SE(3)$ using the previous frame and 3D (point and line) landmarks optimized by the bundle adjustment (Section II-E). The detected 2D point features (ORB) and line features (LBD) in F_i are matched with the previous frame F_{i-1}'s tracking features by performing 2D-2D point matching and line matching, respectively. The estimated 3D point landmarks and 3D line landmarks are extracted to estimate F_i's camera pose by solving a perspective-n-point/line problem [23].

Since the descriptor distance might not be so descriptive for 2D-2D line matching, in practice we use several geometric criteria to filter out 'poor' line matching pairs to improve the matching accuracy. Specifically, a good candidate line matching pair (l_i, l_j) needs to satisfy : (1) the angular difference is smaller than a given threshold; (2) the length difference is also below a given threshold; (3) the distance between the middle points of l_i and l_j should be small enough. After solving the perspective-n-point/line problem, the current frame F_i's camera pose T_i is updated. The 3D point landmarks and line landmarks are re-projected to frame F_i's image plane to check whether the 3D-2D matching is outlier or not. Outliers are removed once detected. As for the details of line camera projection, endpoints trimming, and line triangulation, please refer to [10], [11].

D. Scale Correction Optimization

Since the estimated scale might drift from its real physical scale, we propose to correct the scale using a scale correction optimization. Our key idea is to fuse the relative camera poses computed from the ICP alignment step to adjust the newly estimated keyframe's camera pose and the related 3D landmarks via a scale correction optimization.

This problem is formulated as a graph based optimization as illustrated in Fig. 4. For a newly selected keyframe with the estimated camera pose $T_i \in SE(3)$, we build its neighboring keyframe set $\mathcal{T}_i = \{T_i, T_{i-1}, ..., T_{i-n}\}$, the 3D point landmarks set $\mathcal{P}_i$, and 3D line landmark set $\mathcal{L}_i$ which are visible from the keyframe set $\mathcal{T}_i$. Our goal is to adjust the keyframe's camera pose T_i, 3D point landmarks $\mathcal{P}_i$, and 3D line landmarks $\mathcal{L}_i$ while keeping fixed for other keyframes' camera poses $\mathcal{T}_i \backslash \{T_i\}$. Specifically, this is achieved by minimizing the following error function:

$$E^i = e^{pose}_{i,i-1} + \sum_{P_j \in \mathcal{P}_i} \sum_{T_k \in \mathcal{T}_i} e^P_{j,k} + \sum_{L_j \in \mathcal{L}_i} \sum_{T_k \in \mathcal{T}_i} e^L_{j,k}, \tag{2}$$

where e^{pose}, e^P, and e^L are the relative pose factor error, point re-projection factor error, and line re-projection factor error as described in Section II-E, respectively. This optimization can be solved efficiently using the *g2o* library [24].

Using our scale correction, the scale drift can be reduced efficiently (as shown the evaluation in Section III). For efficiency, the time-consuming ICP alignment is only performed between keyframes. Note that the relative camera

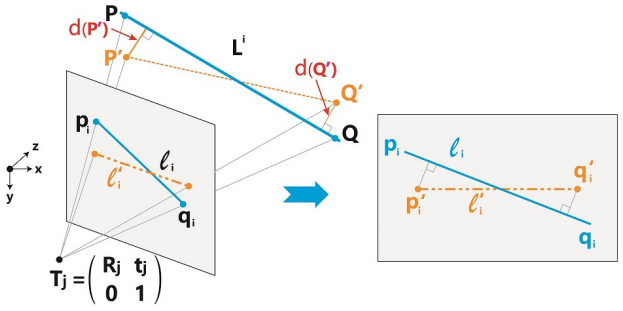

Fig. 5. A 3D line landmark L_w^i is re-projected onto the image plane yielding a 2D line l_i matched with a line segments l_i'(Left), with P' and Q' are the priors computed from the depth prior extracted in Section II-B. The error between the re-projected line l_i and the matched line segment l_i' is defined by the distances from its two endpoints to the line l_i (Right).

pose computed by the frame-to-frame odometry can be used as pose initialization to accelerate the ICP alignment.

E. Point-Line Bundle Adjustment

We form a point-line bundle adjustment in the back-end between keyframes of a sliding neighbor window, similar to ORB-SLAM2 [7]. Specifically, when a new keyframe arrives, bundle adjustment is performed in its neighboring keyframe set $\mathcal{T}$, with 3D point landmark set $\mathcal{P}$ and 3D line landmark $\mathcal{L}$. As illustrated in Fig. 1, we formulate bundle adjustment as a graph based optimization with the following three factors, aiming at adjusting the keyframes' camera poses $\mathcal{T}$, 3D point landmark position $\mathcal{P}$, and 3D line landmark position $\mathcal{L}$.

Relative Pose Factor. For a pair of adjacent keyframes with the estimated camera poses $\{T_i, T_j\} \subset \mathcal{T}$, and the relative camera pose $T_{i,j}^{icp}$ computed from the point cloud ICP alignment step, we formulate the difference vector $\xi_{i,j}$ between the relative camera pose $T_{i,j}^{pose} = T_i * T_j^{-1}$ and $T_{i,j}^{icp}$ as: $\xi_{i,j} = Log(T_{i,j}^{pose} * T_{i,j}^{icp^{-1}}) \in \mathbb{R}^6$, where $Log(\cdot)$ is the logarithmic map of $SE(3)$ [25]. The relative pose error is computed as $e_{i,j}^{pose} = {\xi_{i,j}}^T \Sigma_{i,j}^{pose} \xi_{i,j}$, where $\Sigma_{i,j}^{pose}$ is a 6×6 information matrix. ICP alignment would fail under certain conditions leading to "bad" relative camera poses. To penalize the "bad" ICP alignment, we compute the information matrix adaptively according to the ICP alignment quality. Letting $\hat{e}_{i,j}$ be the ICP error for setting the relative camera pose as $T_{i,j}^{pose}$ and $\bar{e}_{i,j}$ for $T_{i,j}^{icp}$, we compute a scale factor $\zeta_{i,j} = e^{(-2.0*(\frac{\bar{e}_{i,j}}{\hat{e}_{i,j}})^2)}$ and set $\Sigma_{i,j}^{pose} = \zeta_{i,j} I_{6\times 6}$.

Point Re-projection Factor. Assume a 3D point landmark $P_i \in \mathcal{P}$ in the world frame is matched to a 2D point feature positioned p_i in the keyframe $T_j \in \mathcal{T}$. By re-projecting P_i onto the keyframe's image plane, we can obtain a re-projected position for P_i as $p_i' = \pi(P_i, T_j, \mathbb{K})$, where $\pi(\cdot)$ is the projection function and $\mathbb{K}$ is the camera's intrinsic parameter matrix. We then compute the re-projection error as $\rho_{i,j} = (p_i - p_i') \in \mathbb{R}^2$. If there exists depth prior for this point feature, we formulate $\rho_{i,j}$ as stereo-reprojection error as described in ORB-SLAM2 [7] by including the depth prior. Finally, the point re-projection error is defined as $e_{i,j}^P = {\rho_{i,j}}^T \Sigma_{i,j}^P \rho_{i,j}$, where Σ^P is the covariance matrix.

Line Re-projection Factor. A 3D line landmark $L^i \in \mathcal{L}$ in the world frame can also be re-projected to a keyframe $T_j \in \mathcal{T}$, yielding a line segment $l_i = [l_1^i \ l_2^i \ l_3^i]^T \in \mathbb{R}^3$ in the 2D image plane as described in Section II-A. Suppose a line segment l_i' with two endpoints p_i' and q_i' is matched to L^i, as illustrated in Fig. 5. We can formulate the line re-projection error vector as: $\gamma_{i,j} = [\frac{{l_i}^T p_i'}{\sqrt{{l_1^i}^2+{l_2^i}^2}} \ \frac{{l_i}^T q_i'}{\sqrt{{l_1^i}^2+{l_2^i}^2}}]^T \in \mathbb{R}^2$. If there exists depth prior for this line feature, we compute the distance between end points priors P', Q' to the 3D line landmark as $d(P')$,$d(Q')$ separately as shown in Fig. 5(left). Then the line re-projection error vector is computed as: $\gamma_{i,j} = [\frac{{l_i}^T p_i'}{\sqrt{{l_1^i}^2+{l_2^i}^2}} \ \frac{{l_i}^T q_i'}{\sqrt{{l_1^i}^2+{l_2^i}^2}} \ \frac{d(P')+d(Q')}{2}]^T \in \mathbb{R}^3$. Finally, the line re-projection error can be computed as $e_{i,j}^L = {\gamma_{i,j}}^T \Sigma_{i,j}^L \gamma_{i,j}$, where Σ^L is the covariance matrix.

In summary, our keyframe based bundle adjustment optimization seeks the minimization of the following cost function,

$$E = \sum_{\{T_i,T_j\}\subset\mathcal{T}} e_{i,j}^{pose} + \sum_{P_i\in\mathcal{P}} \sum_{T_j\in\mathcal{T}} e_{i,j}^P + \sum_{L_i\in\mathcal{L}} \sum_{T_j\in\mathcal{T}} e_{i,j}^L \quad (3)$$

In solving the non-linear optimization, we adopt a 6D vector $\xi \in \mathbb{R}^6$ as the minimum parameters for a camera pose $T(\xi) \in \mathcal{T}$, a 3D vector $\theta \in \mathbb{R}^3$ for a point landmark $P(\theta) \in \mathcal{P}$, and a 4D vector $\phi \in \mathbb{R}^4$ for a line landmark $L(\phi) \in \mathcal{L}$ [10], to make the system computationally efficient and numerically stable. We solve this point-line bundle adjustment efficiently using the *g2o* library [24].

F. Loop Closure

Loop closure involves loop detection and loop correction based on the keyframes during motion estimation. For loop detection, we first use the DBoW [18] algorithm to train the vocabulary for the point feature (ORB descriptor) and line feature (LBD descriptor) respectively. Then every keyframe K_i is converted to a point BoW vector v_i^P and line BoW vector v_i^L. When evaluating the similarity between keyframes K_i and K_j, we define the similarity score as: $s(i,j) = e^{(1-s_p(v_i^P, v_j^P))^2} e^{(1-s_l(v_i^L, v_j^L))^2}$, with $s_p(v_i^P, v_j^P)$ is the similarity score of the point BoW vector and $s_l(v_i^L, v_j^L)$ for the line BoW vector. Once the similar keyframe candidates for a newly selected keyframe K_i are found, the loop correction is performed using a global bundle adjustment algorithm [7].

III. Experiments

Our system is implemented based on ORB-SLAM2 [7] with three threads, namely frame-to-frame odometry, keyframe based bundle adjustment, and loop closure. When performing ICP alignment, we use the Normal Distributions Transform (NDT) [26] implemented in the PCL library [27] to compute the relative camera pose between two adjacent point clouds. We test our approach on the training dataset of the public KITTI benchmark, which contains sequences 00-10 with ground truth trajectories, and analyze the accuracy and time efficiency of our approach on a computer with Intel

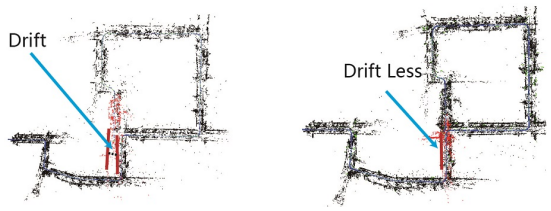

Fig. 6. The motion tracking accuracy comparison between the point-only system (Left) and our system (Right), at the glance of frame 1380 of KITTI dataset sequence 00. The results by the point-only system have obvious drifts, while our results have drift free motion tracking.

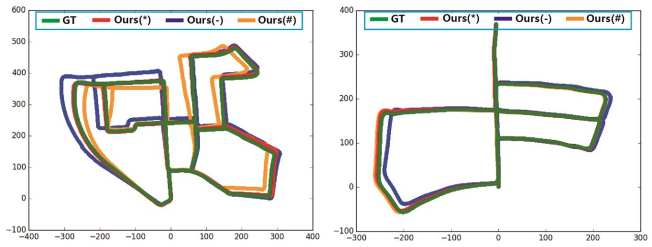

Fig. 7. Our motion tracking trajectories on KITTI training dataset sequences 00 (Left) and 05 (Right), with results by our full system (Ours(*)), the system without depth prior module (Ours(-)) and system without scale correction module (Ours(#)) comparing with the ground truth (GT) trajectory separately.

Core i7-2600 @ 2.6GHz and 8G memory in a 64-bit Linux operation system.

Parameters. For each image, we extract 1000 ORB features. Line segments with length below 50 pixels are discarded. When performing line matching in Section II-C, we set the angle threshold as $2°$, length difference ratio as 0.1, and distance threshold as 5 pixels. In the bundle adjustment step, the covariance matrix is set $\Sigma^P = 0.1 * I$ for the point re-projection factor and $\Sigma^L = 0.2 * I$ for the line re-projection factor.

Point-only versus Point-line. We first evaluate our approach on the performance of tracking features. We implemented a baseline version of our approach using point-only feature, and compared it with our full system. As shown in Fig. 6, our system achieves less drift motion tracking with higher accuracy while the point-only approach has obvious drifts. We conduct a quantitative evaluation on the whole training dataset and summarize the average rotation and translation errors obtained by dividing the segment of each trajectory by 100m, 200m, ..., 800m as done by DVL-SLAM [5], [6]. As shown in Table I, using both the point and line features achieves significantly lower translation error (0.94%) than the point-only baseline (2.16%).

Evaluation on Depth Priors and Scale Correction. We also perform evaluation on the depth priors module (Section II-B) and scale correction module (Section II-D) by comparing the accuracy with our full system. As shown in Table I, we build systems without depth priors module (Ours(-)) or scale correction module (Ours(#)) separately, and compare with our full system (Ours(*)) by the translation error on the whole KITTI training dataset. The comparison results show that the depth priors module reduces the translation error to a great extent (from 3.64% down to 0.94%). The scale correction module also plays an important role to largely reduce the translation error, with about 50% translation error reduced (from 1.95% down to 0.94%). We can conclude that the depth prior module and scale correction module (especially the depth prior module) can largely reduce the pose estimation error in our system. There are several representative motion tracking trajectories by our system with different modules on some KITTI training dataset sequences shown in Fig. 7. Please see more details result in our video attachment.

Comparison with Existing Methods. We compare our approach with DEMO [3], DVL-SLAM [5], [6] and LIMO [4], all of which also have an optimization back-end like ours. Other approaches such as V-LOAM [2] are only visual odometry systems without optimization back-end , and are thus not chosen for comparison. Note that LIMO [4] needs extra semantic label information to identify moving objects such as cars and people. To obtain such semantic label information, it often needs huge computational resources such as Nvidia TitanX Pascal. In contrast, our approach, DEMO and DVL-SLAM do not require such semantic label information and can efficiently run in a CPU-only platform. Our evaluation thus focuses on the comparison between our method, DEMO, and DVL-SLAM. The average translation errors by the compared methods are shown in Table I. Since the semantic label data required by LIMO is available for sequences 00,01,04, we show the results by LIMO for these sequences only.

TABLE I

TRANSLATION ERROR [%] ON THE WHOLE KITTI TRAINING DATASET

Seq.	DEMO	DVL-SLAM	Ours (*)	Ours (-)	Ours (#)	Point-Only	LIMO
0	1.05	**0.93**	0.99	3.23	4.14	2.81	1.12
1	1.87	1.47	1.87	9.40	5.03	3.21	**0.91**
2	**0.93**	1.11	1.38	4.96	3.81	2.10	-
3	0.99	0.92	**0.65**	1.06	0.67	1.67	-
4	1.23	0.67	**0.42**	0.89	0.66	1.78	0.53
5	1.04	0.82	**0.72**	2.94	0.88	1.52	-
6	0.96	0.92	**0.61**	3.28	0.85	1.30	-
7	1.16	1.26	**0.56**	3.53	0.74	2.07	-
8	**1.24**	1.32	1.27	3.71	1.83	3.61	-
9	1.17	**0.66**	1.06	4.41	1.82	1.73	-
10	1.14	**0.70**	0.83	2.63	0.99	1.92	-
avg	1.16	0.98	0.94	3.64	1.95	2.16	0.93

In the whole KITTI training dataset, we achieve the average translation error 0.94% and rotation error $0.0036\frac{deg}{m}$, which are lower than DEMO (translation error 1.14%, rotation error $0.0049\frac{deg}{m}$ in the paper) and DVL-SLAM (translation error 0.98%, rotation error is about $0.0040\frac{deg}{m}$ in the paper). According to the original LIMO paper [4], their reported average translation error is 0.93% and rotation error is $0.0026\frac{deg}{m}$. Though our translation error is slightly higher than LIMO's, we achieve lower translation error in sequences 00 and 04 as shown in Table I. It shows that

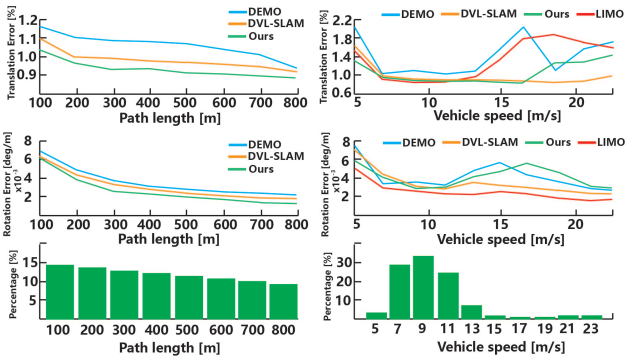

Fig. 8. The error analysis performed on the KITTI training dataset between DEMO, DVL-SLAM, LIMO and Ours. Note that LIMO only provided error curves according to vehicle speed (Right), so we only collect DEMO, DVL-SLAM and Ours error curves in the path length analysis (Left).

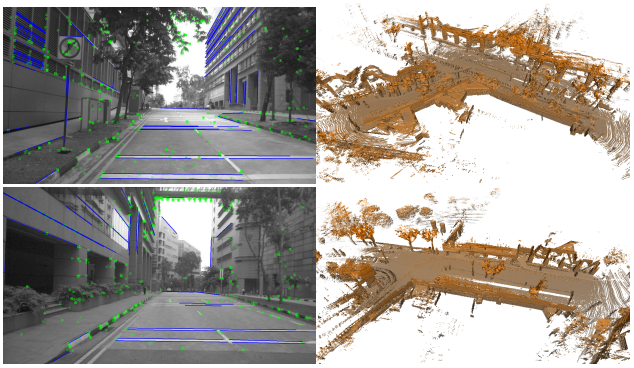

Fig. 9. The test results of scene0007(top row) and scene0024(bottom row) in NuScenes Part I dataset using our full system, including the point-line detection results (left) and final point cloud reconstruction results using the estimated camera poses (right).

our approach as a purely geometric approach can achieve comparable (and sometimes even better) accuracy to LIMO, which requires extra semantic label information.

We perform error analysis according to the path length and vehicle speed as did in DVL-SLAM, and the comparison curves between DEMO, DVL-SLAM, LIMO and ours are shown in Fig. 8. When the vehicle speed is faster, our system leads to the lower accuracy. This explains why our method performs relatively worse on sequence 01, which is a very challenging case with very fast vehicle speed. Please see more detail results of our method on the whole KITTI training dataset in the video attachment.

Evaluation on NuScenes dataset. We evaluate our approach on the NuScenes dataset [28], which is a public large-scale dataset of urban enviroment with plenty of structural line features. To evaluate the accuracy of our lidar-visual odometry without the effect of loop closure, we use the NuScenes Part I subset dataset with 80 test scenes, each of which only has one way trajectory without any loop pathes. Since the trajectory path of every test scenes is not too long ($<$ 200m), we compute the average rotation and translation errors by dividing the trajectory into segments like DVL-SLAM but using 5m, 10m, 50m, 100m, 150m, 200m segment length. In summary, our average translation error is 2.4% and rotation error is $0.0007\frac{deg}{m}$. In Fig. 9, we show the point-line feature detection intermediate results (left) and point cloud reconstruction results using the estimated camera poses (right) of two example test scenes of NuScenes Part I dataset.

Time and Complexity. The feature extraction costs about 80ms, and the scale correction optimization takes about 30ms. The most time consuming part lies in the ICP alignment step, which costs about 300ms each time. Since the ICP alignment is only performed between keyframes, and one keyframe is selected every 3-4 frames, in average our system achieves about 5Hz processing rate in CPU-only platform, the same as LIMO which is achieved with the essential GPU platform.

Discussion and Limitation. In this paper, we don't claim the point-line loop closure as our main contribution, since this BoW based loop closure technique is ordinarily used in systems like DEMO, DVL-SLAM and LIMO. The vehicle odometry sensor data could also be fused into our system by formulating the relative pose measured from the vehicle odometry as factors in the point-line bundle adjustment, leading to a more accurate LiDAR-Visual-Odometry system. However, we only forcus on fusing more structural prior information into the LiDAR-Visual SLAM solution and leave it for the future work. Besides, our system didn't consider the factors raised by errors from the calibration parameters and approximative synchronization of the LiDAR-Camera sensors yet, which is also an interesting problem for a robust LiDAR-Camera SLAM system valuable for a future exploration. One of the drawbacks of our approach lies in the tracking robustness. When the camera moves in large rotations or in fast speed, our tracking system might get failed, leading to less accurate motion estimation. Another drawback of our approach lies in the moving objects, which is beyond the power of purely geometric methods like ours. In the future we would like to incorporate semantic information for a more accurate lidar-visual odometry approach.

IV. Conclusion

In this paper we presented an accurate and efficient lidar-monocular visual odometry approach using both the point and line features. By leveraging more structural information, we show that our approach is more accurate than the state-of-the-art purely geometric techniques, and achieve comparable accuracy with systems using extra semantic information such as LIMO. We hope that our work could inspire follow-up works to explore other types of structural prior information, such as plane prior, parallel, orthogonal or co-plane rules, for a more accurate LiDAR-Camera sensor fusion system.

V. Acknowledgement

Thanks for the anonymous reviewers' detail comments on this paper. This work was supported by the Natural Science Foundation of China (Grant No.: 61521002, 61863031, 61902210, 61862053) and the China Postdoctoral Science Foundation (Grant No.: 2019M660646).

REFERENCES

[1] S. Yang, B. Li, M. Liu, Y.-K. Lai, L. Kobbelt, and S.-M. Hu, "Heterofusion: Dense scene reconstruction integrating multi-sensors," *IEEE Trans. Vis. Comput. Graph.(TVCG)*, 2020 to appear.

[2] J. Zhang and S. Singh, "Visual-lidar odometry and mapping: low-drift, robust, and fast," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 2174–2181.

[3] J. Zhang, M. Kaess, and S. Singh, "A real-time method for depth enhanced visual odometry," *Auton. Robots*, vol. 41, no. 1, pp. 31–43, 2017.

[4] J. Gräter, A. Wilczynski, and M. Lauer, "LIMO: lidar-monocular visual odometry," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 7872–7879.

[5] Y.-S. Shin, Y. S. Park, and A. Kim, "Dvl-slam: sparse depth enhanced direct visual-lidar slam," *Autonomous Robots*, 2019.

[6] Y. Shin, Y. S. Park, and A. Kim, "Direct visual SLAM using sparse depth for camera-lidar system," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 1–8.

[7] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: an open-source SLAM system for monocular, stereo, and RGB-D cameras," *IEEE Trans. Robotics (TRO)*, vol. 33, no. 5, pp. 1255–1262, 2017.

[8] J. Engel, V. Koltun, and D. Cremers, "Direct sparse odometry," *IEEE Trans. Pattern Anal. Mach. Intell. (PAMI)*, vol. 40, no. 3, pp. 611–625, 2018.

[9] A. Bartoli and P. F. Sturm, "The 3d line motion matrix and alignment of line reconstructions," *International Journal of Computer Vision (IJCV)*, vol. 57, no. 3, pp. 159–178, 2004.

[10] A. Bartoli and P. Sturm, "Structure-from-motion using lines: Representation, triangulation, and bundle adjustment," *Computer Vision and Image Understanding (CVIU)*, vol. 100, no. 3, pp. 416–441, 2005.

[11] G. Zhang, J. H. Lee, J. Lim, and I. H. Suh, "Building a 3-d line-based map using stereo SLAM," *IEEE Trans. Robotics (TRO)*, vol. 31, no. 6, pp. 1364–1377, 2015.

[12] X. Zuo, X. Xie, Y. Liu, and G. Huang, "Robust visual SLAM with point and line features," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 1775–1782.

[13] P. Smith, I. D. Reid, and A. J. Davison, "Real-time monocular SLAM with straight lines," in *Proceedings of the British Machine Vision Conference 2006 (BMVC)*, 2006, pp. 17–26.

[14] A. P. Gee and W. W. Mayol-Cuevas, "Real-time model-based SLAM using line segments," in *Advances in Visual Computing, Second International Symposium*, 2006, pp. 354–363.

[15] G. Klein and D. W. Murray, "Improving the agility of keyframe-based SLAM," in *ECCV*, 2008, pp. 802–815.

[16] Y. Zhao and P. A. Vela, "Good line cutting: Towards accurate pose tracking of line-assisted VO/VSLAM," in *ECCV*, 2018, pp. 527–543.

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

[18] D. Gálvez-López and J. D. Tardós, "Bags of binary words for fast place recognition in image sequences," *IEEE Transactions on Robotics (TRO)*, vol. 28, no. 5, pp. 1188–1197, October 2012.

[19] R. G. von Gioi, J. Jakubowicz, J. Morel, and G. Randall, "LSD: A fast line segment detector with a false detection control," *IEEE Trans. Pattern Anal. Mach. Intell. (PAMI)*, vol. 32, no. 4, pp. 722–732, 2010.

[20] L. Zhang and R. Koch, "An efficient and robust line segment matching approach based on LBD descriptor and pairwise geometric consistency," *J. Visual Communication and Image Representation*, vol. 24, no. 7, pp. 794–805, 2013.

[21] K. Tateno, F. Tombari, and N. Navab, "Real-time and scalable incremental segmentation on dense SLAM," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 4465–4472.

[22] W. M. Häußler, "A local convergence analysis for the gauss-newton and levenberg-morrison-marquardt algorithms," *Computing*, vol. 31, no. 3, pp. 231–244, 1983.

[23] A. Vakhitov, J. Funke, and F. Moreno-Noguer, "Accurate and linear time pose estimation from points and lines," in *ECCV*, 2016, pp. 583–599.

[24] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "G^2o: A general framework for graph optimization," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2011, pp. 3607–3613.

[25] H. Strasdat, "Local accuracy and global consistency for efficient SLAM," Ph.D. dissertation, Imperial College London, UK, 2012.

[26] P. Biber and W. Straßer, "The normal distributions transform: a new approach to laser scan matching," in *2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2003, pp. 2743–2748.

[27] P. org., "Pcl library," http://pointclouds.org/.

[28] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuscenes: A multimodal dataset for autonomous driving," *arXiv preprint arXiv:1903.11027*, 2019.