Implementation of V-LOAM

Yukun Xia[1], Shuqin Xie[1], Ivan Cisneros[1], Xinjie Yao[1]

***Abstract*—For many years, the Visual-Lidar Odometry and Mapping (V-LOAM) method has been at the top of benchmarks in odometric accuracy. But there is no official public release of the codebase for this method. The goal for this project is to try to implement V-LOAM and see how our accuracy compares to the official implementation on the KITTI odometry benchmark. In particular, we focus on the visual odometry implementation, and leverage a Lidar odometry implementation (A-LOAM) for the second portion of the method. We implement this system in C++ for computational optimization reasons and use the ROS framework to organize the modules and for data loading and visualization.**

I. INTRODUCTION

A. Background

V-LOAM [1] was introduced by J. Zhang and S. Singh as an odometry and mapping method that tightly couples visual and Lidar sensor data to form an accurate and robust framework that allows 6DoF ego-motion estimation as well as a spatial representation of the environment. Visual sensors and Lidar sensors can each be used for odometry and SLAM, but they have drawbacks when used individually: common challenges and requirements of Visual Odometry (VO) are: scale uncertainty (monocular camera case), the need for sufficient lighting, frame-to-frame lighting consistency, the need for sufficient texture, the need for (mostly) static scenes [2]; and common challenges of Lidar Odometry (LO) are: low data rate (compared with VO), need for knowledge of Lidar pose, and motion distortion due to scan rates [3].

The strength of V-LOAM is in bridging over these weaknesses and allowing accurate estimation even with rapid motion and significant lighting changes. This method allows low-drift motion estimation, mapping, and localization: the visual odometry module takes high frequency estimates of the environment ensuring robustness to rapid motion, but is susceptible to drift due to lower accuracy 3D estimation ability of monocular vision, while the Lidar module refines these estimates at a lower frequency (correcting for the drift) because of the high accuracy of Lidar modules in measuring 3D geometries, and ensures robustness to lighting conditions.

This is an online odometry and mapping method that simultaneously creates a point-cloud map of the traversed environment, while estimating the motion of the camera/Lidar system through that environment. Visual odometry and Lidar odometry each provide an estimated transformation, which are then integrated to provide an overall frame to map transformation estimate. Additionally, the Lidar module has a refinement step, which uses a linear estimate of the visual drift in order to match consecutive point clouds and correct their distortion. Then these corrected point clouds are incrementally matched and merged in order to create the overall environmental map.

[1] are with the Robotics Institute at Carnegie Mellon University, Pittsburgh, PA 15213. Our code is online at https://github.com/YukunXia/VLOAM-CMU-16833

B. KITTI Dataset

As was done in the original V-LOAM paper, we evaluate the implementation on the KITTI Odometry dataset [4]. The KITTI dataset is a collection of 22 sequences of high resolution stereo RGB and Grayscale image data, as well as Velodyne laser data and the ground truth poses. The laser scanner collects 360° point data at 10 frames per second, and the cameras are set up to be triggered by the laser scanner, resulting in image data that is also captured at 10 frames per second; the image and laser data are thus synchronized. The cropped image resolution is 1382 x 512 pixels. It is important to note that the laser data was recorded using a Velodyne HDL-64E, which is a 3D scanner with a set of 64 beams over a 26.8° vertical range, which differs from the Lidar sensor that was used in the original V-LOAM paper, which has ramifications on the architecture of our implementation (more on this is explained in the II-C1 section). These data were recorded using a mobile vehicle sensor platform in the city of Karlsruhe, Germany, which provides a diverse set of real-world, challenging environments that vary in structure, lighting, length, and visual complexity.

Fig. 1: The KITTI data recording platform consisting of four cameras, a 3D Lidar, and an IMU/GPS.

The KITTI group also provides a benchmark suite and leaderboard for comparing different methods' performance to one another. They provide 10 sequences with ground truth trajectories for training and 11 sequences (11-21) without ground truth for evaluation. They accept results from methods using monocular or stereo visual odometry, laser-based SLAM or algorithms that combine visual and Lidar information. The primary metrics that are evaluated on the test set are the average translational and rotational errors (with respect to their ground truth) of the trajectories.

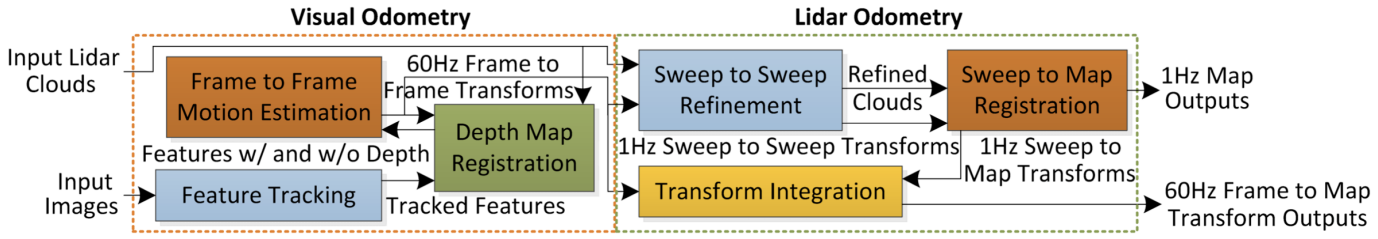


Fig. 2: An overview of the system diagram.

II. METHODS

A. System Overview

Our system diagram follows the design in the original paper [1], containing a *Visual Odometry* (VO) section and a *Lidar Odometry* (LO) section. Figure 2 illustrates an overview of the system. The visual odometry section accounts for estimating motion between two image frames. Input images are first passed to the feature tracking block to detect keypoints and features, then they are fed into the depth map registration block to obtain a depth from Lidar data or marked as unknown depth feature. The frame to frame motion estimation module estimates the motion utilizing the feature correspondence between different image frames. The Lidar odometry is responsible for estimating the sweep to sweep transformation and constructing a map from registered scan. The sweep to sweep refinement module registers incoming scans and estimate sensor motion. The sweep to map registration module maintain updates to the constructed map and further corrects the transform estimates. The transform integration module interpolating high frequency transform from previous estimations.

B. Visual Odometry

The goal of this stage is to estimate frame to frame motion from camera sensors.

1) Feature tracking: We use Shi-Tomasi detector to detect keypoints for each image and extract their ORB features [5]. To set up keypoint correspondence, we perform knn matching (k=2) based on ORB features and only keep the matches whose best-matching result is significantly better than the second-best-matching result (the score should be lower than 20% of the best-matching score). We found this leads to more robust matching result. Our feature tracking module is highly efficient; it only takes $\sim$ 30ms to process the image.

2) Depth Map Registration: We assume the system is calibrated, that is, the transformation from Lidar to camera is given. To register depth for image keypoints, we first transform Lidar point cloud into image frame, obtaining a depth map. Then we query the depth map on the keypoint's location, if a depth is available, assign it to the queried keypoint, otherwise set the keypoint to have unknown depth.

3) Frame to Frame Motion Estimation: This module estimates the motion based on keypoint correspondence between two consecutive frames. Denoting a feature i with known distance in frame k as $\mathbf{X}_i^k = \begin{bmatrix} x_i^k, & y_i^k, & z_i^k \end{bmatrix}^T$ and a feature without known distance as $\hat{\mathbf{X}}_i^k = \begin{bmatrix} \hat{x}_i^k, & \hat{y}_i^k, & \hat{z}_i^k \end{bmatrix}^T$, with $||\hat{\mathbf{X}}_i^k|| = 1$.

The goal of this module is to find a rotation $\mathbf{R}$ and a translation $\mathbf{T}$ such that

$$\mathbf{X}_i^{k+1} = \mathbf{R}\mathbf{X}_i^k + \mathbf{T} \tag{1}$$

Since we always don't know the depth for $\mathbf{X}_i^{k+1}$, we can write it as $\mathbf{X}_i^{k+1} = d_i^{k+1}\hat{\mathbf{X}}_i^{k+1}$, with d_i^{k+1} being the distance of that point.

We consider two situations here: features with known depth and features with unknown depth.

a) Features with known depth: In this case, we know the depth of $\mathbf{X}_i^k$, so we can directly use $\mathbf{X}_i^k$. Substituting $\mathbf{X}_i^{k+1} = d_i^{k+1}\hat{\mathbf{X}}_i^{k+1}$ into Eq. 1 and cancelling out d_i^{k+1}, we can obtain two equations:

$$(\hat{z}_i^{k+1}\mathbf{R}_1 - \hat{x}_i^{k+1}\mathbf{R}_3)\mathbf{X}_i^k + \hat{z}_i^{k+1}T_1 - \hat{x}_i^{k+1}T_3 = 0, \tag{2}$$

$$(\hat{z}_i^{k+1}\mathbf{R}_2 - \hat{y}_i^{k+1}\mathbf{R}_3)\mathbf{X}_i^k + \hat{z}_i^{k+1}T_2 - \hat{y}_i^{k+1}T_3 = 0. \tag{3}$$

The subscription l means the l-th row.

b) Features without known depth: In this case, neither $\mathbf{X}_i^k$ nor $\mathbf{X}_i^{k+1}$ are associated with known depth. Hence we need to substitute both $\mathbf{X}_i^k$ and $\mathbf{X}_i^{k+1}$ with their normalized form $d_i^k\hat{\mathbf{X}}_i^k$ and $d_i^{k+1}\hat{\mathbf{X}}_i^{k+1}$ in Eq. 1. Eliminating both d_i^k and d_i^{k+1}, we can obtain the following equation:

$$\begin{bmatrix} -\hat{y}_i^{k+1}T_3 + \hat{z}_i^{k+1}T_2 \\ -\hat{x}_i^{k+1}T_3 - \hat{z}_i^{k+1}T_1 \\ -\hat{x}_i^{k+1}T_2 + \hat{y}_i^{k+1}T_1 \end{bmatrix} \mathbf{R}\hat{\mathbf{X}}_i^k = 0. \tag{4}$$

To summarize, each feature with known depth yields two equations; feature with unknown depth provides one. Stacking all these equations together, we obtain a system that has six unknown variables (3 for $\mathbf{R}$ and 3 for $\mathbf{T}$). We solve this system with Levenberg-Marquardt solver and use a robust kernel (Huber loss) to reduce the effect of outliers. The obtained motion is later used by Lidar Odometry for transform integration.

C. Lidar Odometry

1) Lidar Odometry System Overview: We follow a pipeline introduced by the same authors in [3], with a different initialization using visual odometry as proposed in [6]. Note in Fig. 2, sweep to sweep refinement refers to the point cloud registration block, and the Lidar odometry block in Fig. 3. Lidar mapping accounts for sweep

to map registration. The point cloud registration block extracts edge points and planar points from each incoming point cloud data. Then in the Lidar odometry block, it removes point cloud distortion due to the rotation motion of the sensor and solves for the sweep-to-sweep transformation. Using the undistorted point cloud, it further merges into the global map by matching the same type of features as the Lidar odometry block.

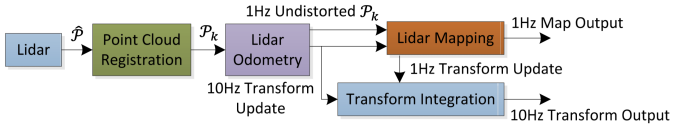


Fig. 3: System diagram of the Lidar odometry and mapping from [3]

2) Sweep to Sweep Refinement:

During feature extraction, it defines a local smoothness measurement (c) to distinguish edge points (maximum c) and planar points (minimum c). Let the coordinates of a point $i, i \in \mathcal{P}_k$, in Lidar coordinate system $\{L_k\}$ are denoted as $\boldsymbol{X}^L_{(k,i)}$, and $\mathcal{S}$ be the set of consecutive points of i returned by the sensor in the same scan.

$$c = \frac{1}{|\mathcal{S}| \cdot \left\| \boldsymbol{X}^L_{(k,i)} \right\|} \left\| \sum_{j \in \mathcal{S}, j \neq i} \left(\boldsymbol{X}^L_{(k,i)} - \boldsymbol{X}^L_{(k,j)} \right) \right\| \tag{5}$$

After downsampling feature points using some heuristics, corresponding feature points are selected. To find an edge line as correspondence from an edge point, the closest point from its consecutive scan meets the condition of forming a line. For a point $i \in$ the reprojected edge point set $\tilde{\mathcal{E}}_{k+1}$, if (j, l) is the corresponding edge line, $j, l \in$ the reprojected point cloud $\overline{\mathcal{P}}_k$, then a point to line distance for edge points is

$$d_{\mathcal{E}} = \frac{\left| \left(\tilde{\boldsymbol{X}}^L_{(k+1,i)} - \overline{\boldsymbol{X}}^L_{(k,j)} \right) \times \left(\tilde{\boldsymbol{X}}^L_{(k+1,i)} - \overline{\boldsymbol{X}}^L_{(k,l)} \right) \right|}{\left| \overline{\boldsymbol{X}}^L_{(k,j)} - \overline{\boldsymbol{X}}^L_{(k,l)} \right|} \tag{6}$$

Similarly, to find a planar patch as correspondences for a planar point, two closest points from its consecutive scan form a patch. For a point $i \in$ the reprojected planar point set $\mathcal{H}_{k+1}$, if (j, l, m) is the corresponding planar patch, $j, l, m \in$ the reprojected point cloud $\overline{\mathcal{P}}_k$, then a point to plane distance is

$$d_{\mathcal{H}} = \frac{\left| \begin{array}{c} \left(\tilde{\boldsymbol{X}}^L_{(k+1,i)} - \overline{\boldsymbol{X}}^L_{(k,j)} \right) \\ \left(\left(\overline{\boldsymbol{X}}^L_{(k,j)} - \overline{\boldsymbol{X}}^L_{(k,l)} \right) \times \left(\overline{\boldsymbol{X}}^L_{(k,j)} - \overline{\boldsymbol{X}}^L_{(k,m)} \right) \right) \end{array} \right|}{\left| \left(\overline{\boldsymbol{X}}^L_{(k,j)} - \overline{\boldsymbol{X}}^L_{(k,l)} \right) \times \left(\overline{\boldsymbol{X}}^L_{(k,j)} - \overline{\boldsymbol{X}}^L_{(k,m)} \right) \right|} \tag{7}$$

Recall the original paper leverages a 2D rotating laser scanner instead of a 3D Lidar in the formulation, thus a linear interpolation is required to recover the 6-DOF pose transform $\boldsymbol{T}^L_{k+1}$ between consecutive scans $[t_{k+1}, t]$.

$$\boldsymbol{T}^L_{(k+1,i)} = \frac{t_i - t_{k+1}}{t - t_{k+1}} \boldsymbol{T}^L_{k+1}, \forall i \in \mathcal{P}_{k+1} \tag{8}$$

While in our implementation, by using a 3d Lidar, linear interpolation is no longer need to remove distortion, Eq. 8 becomes

$$\boldsymbol{T}^L_{(k+1,i)} = \boldsymbol{T}^L_{(k+1)} \tag{9}$$

With the pose transform, a geometric relationship between the edge points set $\mathcal{E}_{k+1}$ and the reprojected edge points set $\tilde{\mathcal{E}}_{k+1}$, or the planar points set $\mathcal{H}_{k+1}$ and the reprojected planar points set $\tilde{\mathcal{H}}_{k+1}$ is

$$\boldsymbol{X}^L_{(k+1,i)} = \mathbf{R} \tilde{\boldsymbol{X}}^L_{(k+1,i)} + \boldsymbol{T}^L_{(k+1,i)}(1:3), \tag{10}$$

The Lidar motion is then estimated by applying this point transformation to obtain point to line distance for edge points and point to plane distance for planar points. Stacking the following two non-linear functions, we solve its motion using Levenberg-Marquardt method. The frame to frame transformation computed by visual odometry is used to initialize the non-linear optimization solver.

$$f_{\mathcal{E}} \left(\boldsymbol{X}^L_{(k+1,i)}, \boldsymbol{T}^L_{k+1} \right) = d_{\mathcal{E}}, i \in \mathcal{E}_{k+1} \tag{11}$$

$$f_{\mathcal{H}} \left(\boldsymbol{X}^L_{(k+1,i)}, \boldsymbol{T}^L_{k+1} \right) = d_{\mathcal{H}}, i \in \mathcal{H}_{k+1} \tag{12}$$

3) Sweep to Map Registration:

The map consists of registered point clouds from each sweep and we have higher confidence in the accumulated map than the Lidar pose estimation. This module constructs a map from distortion-free point cloud and corrects the pose estimation by enforcing another geometric relationship.

It starts by extending the sweep to sweep transformation input $\boldsymbol{T}^L_{k+1}$ between $[t_{k+1}, t_{k+2}]$. Denote $\boldsymbol{T}^W_k$ as the Lidar pose in the map coordinate $\{W\}$ at the end of sweep k, t_{k+1}. This module projects it to one more sweep from t_{k+1} to t_{k+2}, such that $\boldsymbol{T}^W_{k+1}$ is available, and projects the distortion-free point cloud during sweep $k+1$, $\overline{\mathcal{P}}_{k+1}$ into the world coordinates, $\{W\}$, denoted as $\overline{\mathcal{Q}}_{k+1}$.

Then this module matches $\overline{\mathcal{Q}}_{k+1}$ with $\mathcal{Q}_k$ by optimizing the Lidar pose $\boldsymbol{T}^W_{k+1}$ in a similar fashion as II-C3 without the motion model. Leveraging feature points which are 10 times denser than that in II-C3, it solves the transformation by optimizing the point to plane distance of correspondences and merges $\overline{\mathcal{Q}}_{k+1}$ into the map.

D. Transform Integration

The transform integration block aggregates sweep to map pose estimations and frame to frame pose estimations to generate high frequency frame to map pose estimations. While in our case, since the camera doesn't provide a higher frequency input, we maintain the same frequency transforms along the data flow.

III. EXPERIMENTS

In this project, we mainly focused on the the visual odometry part of VLOAM and leveraged the existing implementation of LOAM (ALOAM [6]).

In visual odometry (VO), we wrote our own code, tried to replicate steps in the VLOAM paper series [1], [7], and

also tested some modifications for the purpose of optimization and learning. To feed the VO results into LOAM, we refactored the whole ALOAM into header files and classes, so that a higher level main node could easily control the data flow between each functional block. ALOAM uses separate ROS nodes for those blocks, and the communication between contiguous nodes relies on multiple ROS messages. Hence, the data communication is asynchronous and could be potentially brittle, especially when more functional blocks are added or computation resources is limited. After refactoring, functional blocks can share large data by smart pointers, and a common transformation class to handle different types of odometry and coordinates.

A. Results and Analysis

In this section, we will evaluate our implementation on the sequences in the KITTI Odometry Dataset with pose ground truth, and discuss the influence of various conditions and tricks in VO.

1) Basic Implementation:

As a baseline for further comparison, we firstly evaluated a version of VO, close to the method documented in the VLOAM paper series [1], [7]. There are several small differences. The papers use KLT tracker for feature matching, but our implementation uses Brute Force Matcher in KNN mode. KLT tracker could be more efficient given that KITTI provides a continuous video flow. However, it needs more tuning to stabilize, and we will discuss it later in III-B1. The paper also mentions the triangulation to assign depth for feature points outside the Lidar region, and it could definitely improve the robustness in some extreme cases, eg, the scenario in III-B1. The last difference is that our feature point depth comes from the weighted average of three nearest neighbors, rather than fitting a plane from those neighbors. We find the depth is much more noisy using the paper's method, and some estimated depths are negative. As shown in Fig.4 (d), at the margin of depth association, many abnormally small distances are associated, colored in red, and those blue dots in the green region means the distance is incorrectly large. However, in Fig.4 (c), our implementation provides non-negative depth all the time.

The evaluation of the basic implementation in all 10 sequences is listed in Table I. Excluding evaluation sequence 01, our basic implementation successfully reduces the LOAM translational RMSE from 1.7248% to 1.2604%, and rotational error from 0.0071 to 0.0055 deg/m. Besides, in the intermediate step of Lidar frame-to-frame motion estimation, the errors are also significantly decreased, from LO(Decoupled) to LO(Combined). In sequence 01, VO obviously fails to track the motion, causing a huge translational error. We will discuss this failure with more details in section III-B1.

2) Cost Function Design:

The whole series of papers related to VLOAM [1], [3], [7] all highly rely on nonlinear least square optimization. Specifically for VO in VLOAM, there are two sets of equations to be considered, as listed in METHODS section II-B.

In principle, VO in VLOAM can be viewed as a special type of RGB-D odometry, where there are four typical error criteria to be considered: photometric reprojection error, geometric reprojection error, 3D point-to-point distance and 3D point-to-plane distance [8]. Since VLOAM uses sparse features for image registration, the photometric error can be neglected. The VLOAM paper series use the Geometric reprojection error only. In VLOAM [1], it's mentioned that the current depth is always unknown, and in DEMO [7], it says only one of the two frames with depth should be used, and by convention, the depth is associated to the previous frame. In the KITTI Dataset, both Lidar and Cameras are running in 10Hz, so every image could be associated with a depth map. We ran all the evaluation sequences again with four cost functions to be considered, including 3D-3D, 3D-2D, 2D-3D and 2D-2D cases. In order to utilize the depth information as much as possible, the 3D-3D cost function is the most prioritized, and 2D-2D cost function is the opposite.

The evaluation results are shown in II. Comparing VO's results, adding the 3D-3D cost function significantly decreases the accuracy of VO and LO, but MO is only slightly worsened. The reason why VO is worse could be due to the fact that feature points are the corner points and so they have a large chance that the depth association has a wide range of error. The depth could come from the background or the foreground. Therefore, more outliers are introduced into the optimization solver. LO takes the VO's odometry as initialization, so the result comparison is similar.

One confusing result is that, sometimes when VO is better, LO gets worse, and then MO becomes better, and in some other cases, the opposite happens. One potential reason is that here we only evaluate the average error. If VO has larger average error and smaller deviation of that error, it could even help LO more. LO has its own optimizer, but the optimization iteration is highly limited in ALOAM due to the efficiency requirement. Therefore, as long as the error of VO is within a certain threshold, LO has a high chance of convergence. Nevertheless, this interesting phenomenon needs a closer and more rigorous investigation.

3) RANSAC vs Nonlinear Least Square Optimization:

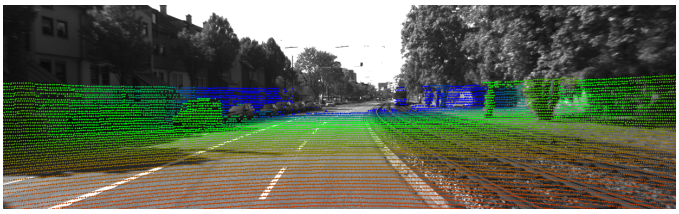
RANSAC is another common solution to robustly estimate motion from vision. Here, we use the 5-point algorithm with RANSAC to compare the frame-to-frame rotation with the basic implementation.

As we can see in the Table III, the rotational error of RANSAC is on average smaller than the basic implementation, and is more robust in extreme cases, e.g. evaluation sequence 01. Another interesting test is using PnP+RANSAC to include translational motion in the comparison too, but due to the time limitation, it was not implemented.

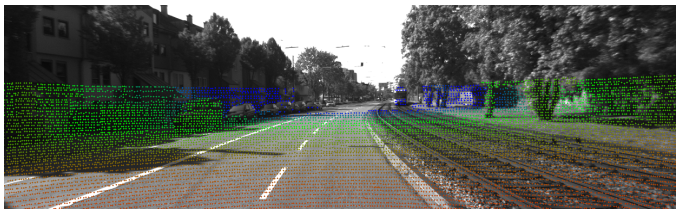
4) KLT tracker vs Feature Matching:

The original implementation uses KLT tracker to match key points, and so here we replace the brute force matcher with a KLT tracker (using OpenCV *calcOpticalFlowPyrLK* function). Our parameter settings are:

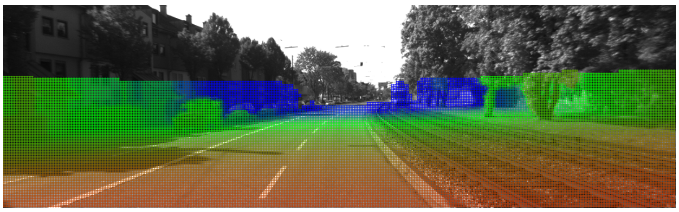
- window size = 15



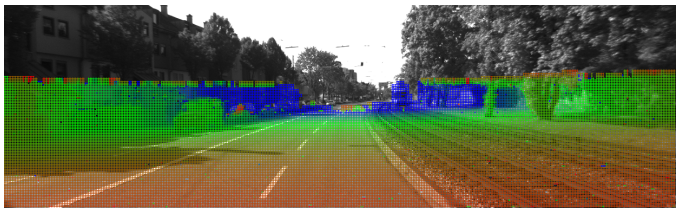
(a) Lidar point cloud projected to camera00 frame (left gray camera)



(b) Downsampled Lidar point cloud on camera00 frame with 5 pixels as the grid cell size



(c) Depth from weight average of the three nearest neighbors



(d) Depth from plane fitting of the three nearest neighbors.

Fig. 4: Image depth association from Lidar point cloud. Point color corresponds to depth: red means small distance and blue means long distance

TABLE I: Evaluation of Odometries in the Basic Implementation

Evaluation Sequence		VO	LO (C)	LO (D)	MO (C)	MO (D)	Sequence Length(s)
00	translational RMSE (%)	2.0254	5.2519	6.3766	1.0670	1.0741	4540
	rotational error (deg/m)	0.0092	0.0235	0.0275	0.0051	0.0051	
01	translational RMSE (%)	50.5147	48.3037	4.3825	45.3560	2.7186	1100
	rotational error (deg/m)	0.0578	0.0431	0.0123	0.0242	0.0069	
02	translational RMSE (%)	2.1758	5.1230	7.8746	1.5226	3.6098	4660
	rotational error (deg/m)	0.0121	0.0196	0.0289	0.0058	0.0134	
04	translational RMSE (%)	3.2902	1.8323	1.8629	0.5998	0.6051	270
	rotational error (deg/m)	0.0236	0.0096	0.0152	0.0041	0.0041	
05	translational RMSE (%)	2.3870	5.0247	5.7018	0.8550	0.8810	2760
	rotational error (deg/m)	0.0140	0.0237	0.0260	0.0048	0.0040	
06	translational RMSE (%)	2.3971	2.6512	2.8825	1.4302	1.4623	1100
	rotational error (deg/m)	0.0083	0.0132	0.0145	0.0067	0.0068	
07	translational RMSE (%)	4.1385	2.0928	2.1205	0.7953	0.7892	1100
	rotational error (deg/m)	0.0262	0.0185	0.0173	0.0070	0.0066	
08	translational RMSE (%)	2.5765	5.1730	6.3459	1.4604	1.4597	4070
	rotational error (deg/m)	0.0107	0.0223	0.0291	0.0055	0.0055	
09	translational RMSE (%)	2.3979	6.9019	8.0834	1.7278	1.7329	1590
	rotational error (deg/m)	0.0117	0.0226	0.0267	0.0062	0.0063	
10	translational RMSE (%)	2.3586	4.9008	5.7633	1.0273	1.0459	1200
	rotational error (deg/m)	0.0176	0.0238	0.0272	0.0058	0.0058	
Average (all)	translational RMSE (%)	4.7653	7.0720	6.1548	3.4267	1.7736	
	rotational error (deg/m)	0.0146	0.0225	0.0258	0.0067	0.0071	
Average (without 01)	translational RMSE (%)	2.4016	4.9416	6.2464	**1.2604**	**1.7248**	
	rotational error (deg/m)	0.0124	0.0214	0.0265	**0.0055**	**0.0071**	

Note: VO = Visual Odometry. LO = frame to frame estimation of Lidar Odometry. MO = frame to map estimation of Lidar Odometry. (C) = Combining VO and LO, i.e. VLOAM mode. (D) = Decoupling VO and LO, i.e. VO&LOAM mode.

- maximal level = 2
- termination criteria = 10 steps
- termination epsilon = 0.03

The results of the KLT version are shown in Fig. 6. We chose two representative time frames (4.03s and 11.52s) to compare the KLT version against the basic implementation. As we can see, in the KLT version, there are more matches in total, fewer large distance mismatches, but more small distance mismatches. In Fig. 6(b), the optical flow near the top and bottom right margin is quite messy. In Fig. 6(d), lots of mismatches exist on the right side tree. Therefore, it's harder for the robust kernel in the optimizer to distinguish outliers in the KLT version, and the large errors in the odometry support this analysis.

Note that the performance could be highly dependent on the parameter setting, a "fair" comparison relies on more comprehensive studies.

5) Mapping Frequency:

One extra factor of the overall VLOAM performance could also depend on the running frequency of the mapping module. Table IV demonstrates that, with higher computational cost, a higher frequency VLOAM always has smaller translational and rotational error. In the block diagram of the VLOAM paper, the mapping module runs at 1Hz, but according to the

TABLE II: Comparison Between Basic Implementation and the Implementation with Four Cost Functions

Evaluation Seq		VO (Basic)	VO (All)	LO (C-Basic)	LO (C-All)	MO (C-Basic)	MO (C-All)	Sequence Length
0	translational RMSE (%)	2.0254	4.4182	5.2519	4.9637	1.0670	1.1126	4540
	rotational error (deg/m)	0.0092	0.0222	0.0235	0.0217	0.0051	0.0052	
1	translational RMSE (%)	50.5147	42.4450	48.3037	38.7857	45.3560	34.0033	1100
	rotational error (deg/m)	0.0578	0.0442	0.0431	0.0277	0.0242	0.0174	
2	translational RMSE (%)	2.1758	6.0832	5.1230	4.1166	1.5226	1.5431	4660
	rotational error (deg/m)	0.0121	0.0242	0.0196	0.0151	0.0058	0.0057	
4	translational RMSE (%)	3.2902	10.7834	1.8323	4.8278	0.5998	0.7675	270
	rotational error (deg/m)	0.0236	0.0855	0.0096	0.0213	0.0041	0.0040	
5	translational RMSE (%)	2.3870	5.1730	5.0247	4.0327	0.8550	0.8512	2760
	rotational error (deg/m)	0.0140	0.0262	0.0237	0.0179	0.0048	0.0048	
6	translational RMSE (%)	2.3971	3.6164	2.6512	1.8332	1.4302	1.4207	1100
	rotational error (deg/m)	0.0083	0.0203	0.0132	0.0091	0.0067	0.0066	
7	translational RMSE (%)	4.1385	4.5304	2.0928	2.0959	0.7953	0.7916	1100
	rotational error (deg/m)	0.0262	0.0298	0.0185	4.5304	0.0070	0.0066	
8	translational RMSE (%)	2.5765	6.0618	5.1730	4.3320	1.4604	1.4228	4070
	rotational error (deg/m)	0.0107	0.0265	0.0223	0.0265	0.0055	0.0054	
9	translational RMSE (%)	2.3979	6.3650	6.9019	5.4819	1.7278	1.7408	1590
	rotational error (deg/m)	0.0117	0.0242	0.0226	0.0176	0.0062	0.0062	
10	translational RMSE (%)	2.3586	7.8724	4.9008	3.6932	1.0273	1.0124	1200
	rotational error (deg/m)	0.0176	0.0342	0.0238	0.0169	0.0058	0.0057	
Average	translational RMSE (%)	4.7653	7.3910	7.0720	5.8918	3.4267	2.8767	
	rotational error (deg/m)	0.0146	0.0268	0.0225	0.2414	0.0067	0.0062	
Average (exclude 1)	translational RMSE (%)	**2.4016**	**5.5799**	4.9416	4.1923	1.2604	1.2685	
	rotational error (deg/m)	**0.0124**	**0.0259**	0.0214	0.2524	0.0055	0.0055	

Note: (Basic) means the data is from the basic implementation, and (All) means it considers all cost functions.

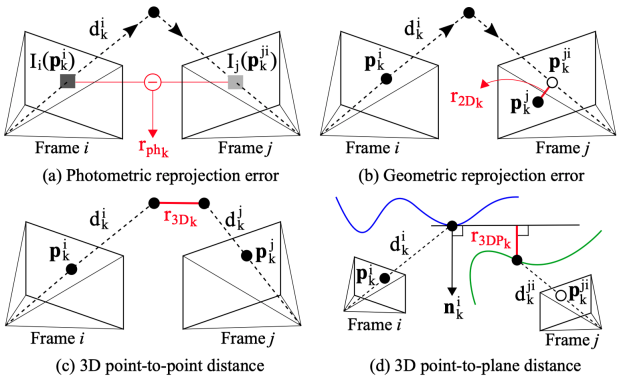


Fig. 5: Four common types of error criteria for RGB-D SLAM [8]

TABLE III: Comparison of Rotational Error Between Basic Implementation and the RANSAC Version

Evaluation Seq		VO (RANSAC)	VO (Basic)
0	rotational error (deg/m)	0.0144	0.0092
1	rotational error (deg/m)	0.0110	0.0578
2	rotational error (deg/m)	0.0159	0.0121
4	rotational error (deg/m)	0.0096	0.0236
5	rotational error (deg/m)	0.0160	0.0140
6	rotational error (deg/m)	0.0147	0.0083
7	rotational error (deg/m)	0.0251	0.0262
8	rotational error (deg/m)	0.0116	0.0107
9	rotational error (deg/m)	0.0117	0.0117
10	rotational error (deg/m)	0.0120	0.0176
Average (all)	rotational error (deg/m)	0.0139	0.0146

experiment, it's recommended to run faster if running time permits.

TABLE IV: Comparison of the Final Performance Between Different Mapping Frequency

Evaluation Seq		MO (D-Basic)	MO (D-2Hz)
0	translational RMSE (%)	1.0741	1.4572
	rotational error (deg/m)	0.0051	0.0067
1	translational RMSE (%)	2.7186	2.9931
	rotational error (deg/m)	0.0069	0.0071
2	translational RMSE (%)	3.6098	4.0875
	rotational error (deg/m)	0.0134	0.0150
4	translational RMSE (%)	0.6051	1.6504
	rotational error (deg/m)	0.0041	0.0054
5	translational RMSE (%)	0.8810	1.1574
	rotational error (deg/m)	0.0040	0.0055
6	translational RMSE (%)	1.4623	1.5906
	rotational error (deg/m)	0.0068	0.0069
7	translational RMSE (%)	0.7892	0.9842
	rotational error (deg/m)	0.0066	0.0075
8	translational RMSE (%)	1.4597	1.5474
	rotational error (deg/m)	0.0055	0.0065
9	translational RMSE (%)	1.7329	2.1406
	rotational error (deg/m)	0.0063	0.0084
10	translational RMSE (%)	1.0459	1.4032
	rotational error (deg/m)	0.0058	0.0075
Average (all)	translational RMSE (%)	**1.7736**	**2.0908**
	rotational error (deg/m)	**0.0071**	**0.0085**

Note: (Basic) runs mapping solver at 10Hz.

B. *Case Study for Failures*

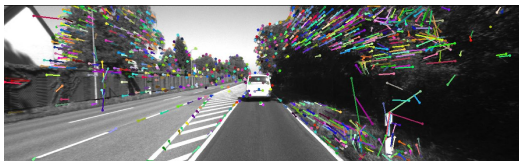
As shown in the previous sections, we faced two main failures within the given 10 sequences in KITTI Dataset.

1) Failure Case 1: Highway in Sequence 01:

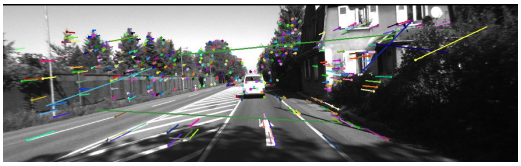
This sequence is particularly difficult for visual odometry due to the following factors:

1) On highway, most regions in an image are featureless, either sky or uniform ground, and thus there are far fewer feature points. As shown Fig 7(a). Only a small portion of feature point matches are correct, and the corners of white marks are note fully detected. At this

(a) Feature point matches of basic implementation in Sequence 04 at ∼4.03s

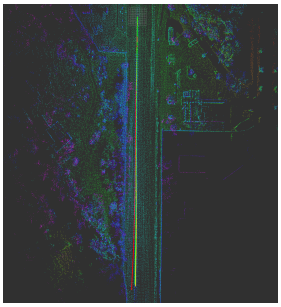


(b) Optical Flow of KLT implementation in Sequence 04 at ∼4.03s

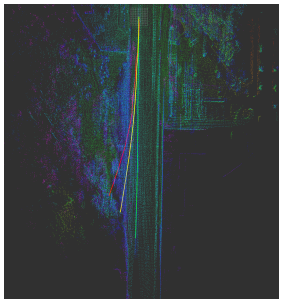


(c) Feature point matches of basic implementation in Sequence 04 at ∼11.52s

(d) Optical Flow of KLT implementation in Sequence 04 at ∼11.52s



(e) Three odometries of Basic implementation in Sequence 04 at ∼11.52s



(f) Three odometries of KLT version in Sequence 04 at ∼11.52s

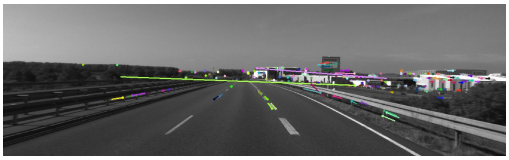
Fig. 6: Comparison between Brute Force Matcher and KLT Trakcer. In (e) and (f), red, yellow and green curves respectively represent VO, LO and MO. In the following figures, odometries will always be represented by those colors.

frame, LO estimates the front direction speed is ∼26m/s, while VO estimates it to be ∼4.4m/s.

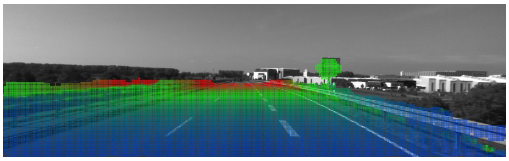
2) The vehicle speed is particularly high so that feature points have larger shifts, and easily move out of the field of view.
3) The camera frequency is only 10Hz.
4) The landmarks around the vehicle, e.g. corners of the guard rail, look very similar to each other. With the high vehicle speed, both naive matching or tracking will be likely to fail.
5) Other landmarks are at a distance, so their shifts in the image plane are small and potentially noisy.
6) Combining the previous two factors, Lidar point cloud covers a relatively smaller region of the image, e.g. less than 50% of the camera field of view here in Fig.7(b).
7) VO in LOAM heavily relies on feature points with depth to measure linear motion.
8) In contrast to Lidar with 360° horizontal field of view, the camera can only see a narrow front scene.
9) (ix) Dynamic objects occupy too much area. In Fig 8(a-b), another car moves fast toward the forward direction. Subsequently, the visual odometry estimates that the ego vehicle is moving backwards.

2) Failure case 2: Waiting for traffic at a crossroad in Sequence 07:

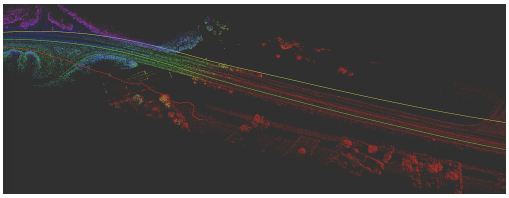
This failure case is similar to the last factor in the failure



(a) Feature point matches

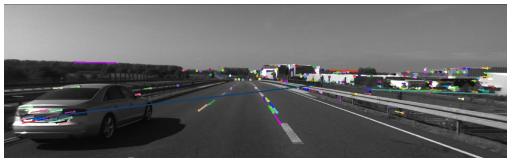


(b) Depth map

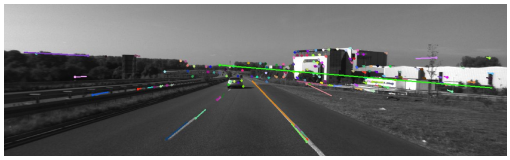


(c) Odometries

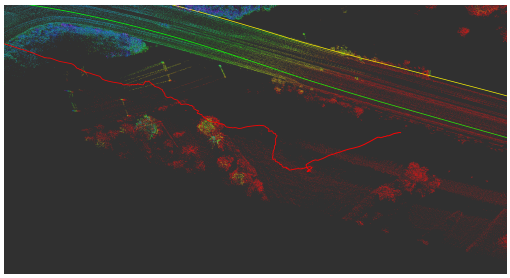
Fig. 7: Failure case 1 example 1.



(a) Side vehicle shows up



(b) Side vehicle is at a distance



(c) Odometries after (b)

Fig. 8: Failure case 1 example 2.

case 1. Many relatively fast moving vehicles appear and occlude the static background. In Fig.9(a-b), a truck moves to the right and VO (in red) bends left, and when a car moves to the left, VO turns to the right side. As more and more large or nearby vehicles pass by, VO gets more distorted. However, the ego vehicle is actually static at that point.

3) Potential Improvements:

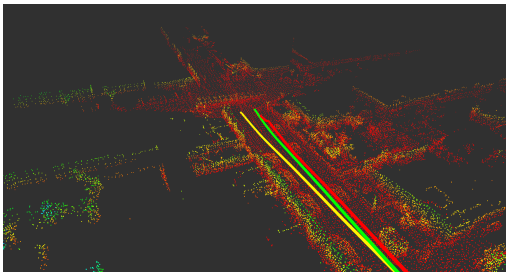
One potentially good method to have better feature matching/tracking is to include motion prediction or motion prior from other sensors. For example, Wu [9] used smooth motion constraint to help KLT tracking the correct target position in KITTI sequence 01. In [10], IMU results help remove the rotational motion for tracking.

Built upon that, another important trick could be detecting and eliminating outliers. For dynamic objects, especially those fast moving vehicles, we could use deep models to detect their bounding boxes, or simply track the groups of outliers and keep ignoring them, until they slow down.

Additionally, adding bundle adjustment and triangulation as mentioned in VLOAM are definitely beneficial.

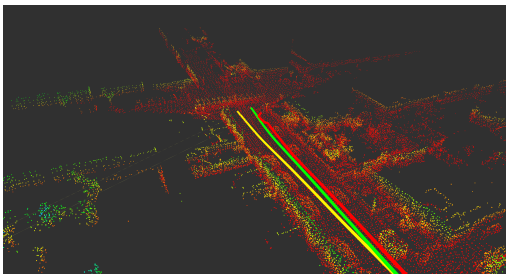
Lastly, we can try out different parameters, feature detectors and descriptors and compare. For example, our observation is that using SIFT for both detection and matching in sequence 01 will lead to slightly better trajectories than ShiTomasi+ORB or ORB+ORB. ORB+ORB detects the white marks better than ShiTomasi+ORB, but has more problems on normal conditions. The whole procedure relies on tuning experience, and is both time consuming and tedious, but still is a way for improvement.

(a) Feature point matching



(b) Odometries

(c) Feature point matching



(d) Odometries

Fig. 9: Failure case 2. In (a) and (b), a truck is moving right, and in (c) and (d) a car is moving left.

IV. CONCLUSIONS

In summary, we finished a minimal version of VLOAM in this project. We found that, compared with Lidar Odometry, Visual Odometry generally has higher accuracy but has lower robustness, or at least, takes more effort to deal with those difficult edge cases. Given a stable visual odometry, simply adding it as the initialization of liar odometry will reduce the average pose estimation errors, as pointed out in the VLOAM paper [1]. In addition to trying replicating the original implementation, we also tested other variants of cost function design, robust estimation framework, feature matching/tracking methods. Finally, we revisited the failure cases in our visual odometry and discussed the potential solutions. From those cases, we learned that a stable visual odometry could take a long time to implement, test and optimize.

In the future, methods in III-B3 are worth trial and beyond ALOAM, loop closure and pose graph optimization can be introduced. If a higher frequency camera is provided, we could

consider removing the motion distortion in point cloud. Lastly, implementing LOAM from scratch should be considered to truly understand the ideas and challenges inside LOAM.

ACKNOWLEDGMENT

A special thanks to Professor Kaess for teaching us this semester. And a big thanks to our TAs Ceci Morales, Jay Patrikar, and Wei Dong for their tireless help.

REFERENCES

[1] Ji Zhang and Sanjiv Singh. Visual-lidar odometry and mapping: Low drift, robust, and fast. In *IEEE International Conference on Robotics and Automation(ICRA)*, Seattle, WA, May 2015.

[2] Davide Scaramuzza and Friedrich Fraundorfer. Visual odometry [tutorial]. *IEEE Robot. Autom. Mag.*, 18(4):80–92, December 2011.

[3] Ji Zhang and S. Singh. Loam: Lidar odometry and mapping in real-time. In *Robotics: Science and Systems*, 2014.

[4] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[5] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. ICCV '11, page 2564–2571, USA, 2011. IEEE Computer Society.

[6] Tong Qin and Shaozu Cao. Advanced implementation of loam.

[7] Ji Zhang, M. Kaess, and S. Singh. Real-time depth enhanced monocular odometry. *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4973–4980, 2014.

[8] Javier Civera and Seong Hun Lee. *RGB-D Odometry and SLAM*, pages 117–144. Springer International Publishing, Cham, 2019.

[9] Meiqing Wu, Siew-Kei Lam, and Thambipillai Srikanthan. A framework for fast and robust visual odometry. *IEEE Transactions on Intelligent Transportation Systems*, 18(12):3433–3448, 2017.

[10] Myung Hwangbo, Jun-Sik Kim, and Takeo Kanade. Inertial-aided klt feature tracking for a moving camera. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1909–1916. IEEE, 2009.