Visual Lidar Odometry and Mapping with KITTI

Ali Abdallah*, Alexander Crean†, Mohamad Farhat‡, Alexander Groh§, Steven Liu¶ and Christopher Wernette‖
University of Michigan
Ann Arbor, MI USA
Email: *alihabda@umich.edu,†amcrean@umich.edu,‡mwi@umich.edu,§grohbot@umich.edu,¶stvnliu@umich.edu,‖chriswer@umich.edu

***Abstract* — The combination of perceptually dense camera images with long-range laser scanners make accurate and robust odometry and mapping algorithms in many types of environments possible. Our goal is to integrate two open source libraries, DEMO (Depth Enhanced Monocular Odometry) and LOAM (Lidar Odometry and Mapping), into a publicly available V-LOAM (Visual-Lidar Odometry and Mapping) package. However, we ran out of time to work on DEMO. In this paper, we present a study on the open source LOAM algorithm, modify it to work with the KITTI dataset, and validate our independent results against the provided ground truth. The results of our study is publicly available on Github at [1], and videos of our test runs are available on YouTube at https://www.youtube.com/watch?v=YTCnmP5RBuE.**
While testing LOAM, we found large rotation errors when the vehicle is completing a turn. A vision based motion prior as implemented in V-LOAM would improve this error and is part of our future work.

I. Introduction

There are many companies contending to be the first to bring an autonomous fleet to full mass production. The key enablers of robust autonomous vehicles are the computational hardware, the sensing suite, the algorithms which interpret the sensing inputs and control the vehicle, and maps. Accurate and precise prior maps are critical because they greatly reduce the complexity of the autonomous vehicle problem, which allows the solution to go to market sooner.

A typical sensor suite for an autonomous vehicle includes lidars, cameras, radars, and ultrasonic sensors. Redundancy in sensing is a critical part of the robust, reliable, and safe autonomous vehicle solution. As a natural extension, perception algorithms need to incorporate multiple sensing modalities. There are advantages and disadvantages to each type of sensor and the algorithms should harness the synergistic qualities of each modality. Lidar and cameras naturally complement each other.

Cameras combined with modern computer vision techniques are able to capture the perceptually dense information contained in the visible spectrum at high frequencies. The problem with a camera only autonomous vehicle solution is that even with stereo cameras, depth is not robustly determined. Lidars can provide the robust depth determination augmenting the cameras, but due to the spinning mechanism, they are not able to appropriately sample high frequency time domain information. This paper describes a novel solution which combines high frequency camera data with the robust depth determination of a lidar to estimate pose and create precise point cloud maps.

This paper is divided into the following sections:

- Problem Formulation
- Implementation
- DEMO
- LOAM
- Results
- Challenges
- Limitations
- Conclusions and Future Directions

A. VLOAM, LOAM, and DEMO Overview

Ji Zhang's and Sanjiv Singh's work titled 'Visual-lidar Odometry and Mapping: Low-drift, Robust, and Fast' [10] constructs a real-time framework for tightly coupling camera and lidar for egomotion estimation and point cloud map generation. The V-LOAM method was developed as a combination of two previous works: LOAM: Lidar Odometry and Mapping in Real-time [9] and DEMO: Depth Enhanced Monocular Odometry [11].

LOAM contains the groundwork for lidar odometry and lidar point cloud mapping in V-LOAM. As of April 11, 2017, LOAM is the second highest ranked result on the KITTI Odometry Challenge. Ji Zhang, et. all [11], present a novel way to incorporate the sparse depth information from lidar into monocular camera based egomotion estimation (DEMO), which enables robust egomotion estimating even with high frequency vehicle motion.

The V-LOAM algorithm is designed to estimate egomotion over long distances with minimal drift without loop closure, yet it yields better results than algorithms which use loop closure. In V-LOAM, the overall process for odometry and mapping is initiated with a visual odometry method that estimates egomotion at a high frequency, followed by a lidar odometry method that refines motion estimates, and corrects drift at a low frequency. The egomotion estimation is used to undistort the lidar point cloud which is then fed into the mapping subsection which matches and registers the lidar point cloud using lidar based features. The egomotion estimation is refined again using the constructed point cloud map.

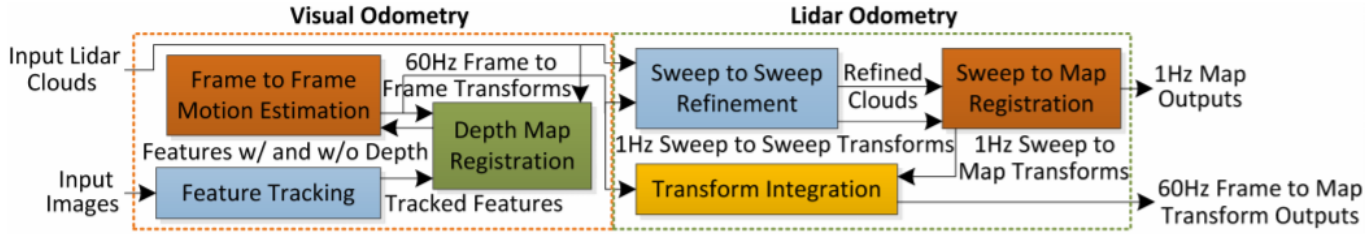

Fig. 1: Block diagram of the odometry and mapping software system [10]

V-LOAM is an algorithm which has direct applications to autonomous vehicles through the tight coupling of multiple sensing modalities. It is robust against high frequency motion, which is a flaw of lidar only odometry and mapping solutions, and is resistant to translation scaling issues, a flaw of camera only egomotion estimation solutions.

In the V-LOAM paper, they do not mention using an IMU, however, it is clear from their earlier work in LOAM that using an IMU can be beneficial in further reducing high frequency motion errors that can distort point cloud registration. Using the IMU to further account for high frequency motion is another natural extension of the V-LOAM algorithm.

Our work attempts to understand and implement the V-LOAM algorithm from [10], modify the code to work with the KITTI dataset hardware, and analyze the results with the KITTI dataset. Due to time constraints and bugs associated with the DEMO subsection, we are only releasing the results for LOAM at this time.

II. Problem Formulation

The goal of this project is to understand the V-LOAM algorithm and report an independent evaluation and validation of the published results of the KITTI Vision Benchmark Suite [5]. Sensors available from the suite include an IMU, lidar, and monocular grayscale[6]. Although an IMU is not mentioned being used in the V-LOAM paper, it is used in the LOAM algorithm when available to correct for high frequency motion such as sinusoidal motion or quick turns.

III. Implementation

For the development environment, Ubuntu 16.04 was used with ROS-Kinetic. This allowed us to use RVIZ for visualization purposes. We chose to run ROS-Kinetic due to it being a recent version of ROS which supports package dependencies for our project. Our plan was to modify the open source LOAM to be KITTI compatible, modify the open source DEMO to be KITTI compatible, then integrate them into V-LOAM and release an open source V-LOAM ROS package, which is not publicly available.

We started two work streams in parallel, LOAM conversion and DEMO conversion to KITTI compatible. The open source DEMO package was built on ROS-Fuerte, which uses OpenCV2. We converted all of the OpenCV2 methods to OpenCV3 and created a ROS-Kinetic catkin package to build DEMO in ROS-Kinetic. We were able to build DEMO and get it running with Zhang's provided sample dataset. We were not able to able to get the KITTI dataset working with DEMO due some image processing pipeline issues which we believe come from the open source package kitti2bag, which we were using to convert raw KITTI data into ROS .bag files.

The open source LOAM package we started with was built for a custom sensing setup where a Hokuyo UTM-30LX lidar was attached to a 3D printed motor housing spinning from -90 to +90 degrees. We had to build the velodyne drivers into ROS package and modify the feature extraction code in the lidar odometry and mapping sections to be able to use the 360 degree HDL-64E in the KITTI dataset. We initially started working with Velodynes sample point cloud files or .pcap files to ensure the point cloud registration and feature extraction were performing correctly. Once those sections were validated, we briefly validated lidar odometry and mapping using RVIZ which produced detailed, distortion free maps.

Datasets were selected from the KITTI Vision Benchmark Suites Odometry Challenge[6] that had ground truth data available for benchmarking (sequences 00-10).

After running LOAM with a ROS bag, the point cloud output of the map is in bag format. Using the PCL tool bag2pcd turns the .bag file into a series of .pcd files. These .pcd files all have the same global reference coordinate system, but they need to be concatenated together to be viewed as a whole map. We choose to use the open source tool CloudCompare to concatenate and view the map. There is significant performance issues when running CloudCompare with PCD files, so as soon as the files are imported and concatenated, it is wise to convert to CloudCompares native .bin file format.

IV. DEMO

A. Mathematical Notation

Let C be a 3D coordinate system with its origin at the center of the camera. X points left, Z points forward, and Y points upwards.

The following will be used to represent the vehicle's pose: $X_i^k = [x_i^k, y_i^k, z_i^k]^T$

Superscript k will be used to denote timestep k, and subscript i will be used to denote visual feature i.

B. Feature Identification and Tracking

Odometry is the process of determining the position and orientation of a robot. Vision based methods are commonly

used for 6-DOF egomotion estimation problems. The generic algorithm for solving this problem is to acquire input images, apply image processing techniques to remove lens distortion, detect features in the image, track features from frame to frame, create optical flow vector field, and estimate the rotation and translation from the optical flow field while removing outliers. Typically, stereo cameras are required to obtain scale of the translation estimation, but the advent of RGB-D cameras allowed depth information to be combined with monocular vision setups to obtain scaled translation. Sources [2], [8], [3],[4] require depth information for each visual feature in order use those features for egomotion estimation. The Depth enhanced monocular visual odometry used in V-LOAM is not subject to the 100 percent depth requirement and can operate using sparse depth information generated from RGB-D cameras, stereo cameras, lidar, or some other depth sensor.

C. Depth Association

New points are added to the depth map upon receiving the depth images or point clouds. The depth map points are converted into a spherical coordinate system. Each point is represented by its radial distance, azimuth angle, and polar angle. The depth map is down-sampled to maintain constant point density over an angular interval as viewed from the optical center of the camera at frame $k-1$. This results in a denser point distribution closer to the camera and sparser as the radial distance increases. The down-sampled depth map is stored in a 2D KD-tree based on the angular coordinates. For each visual feature $i \in I$, select the three depth points closest to feature i, form a planar patch with the three depth point, project the visual feature i onto the planar patch, and assign the distance associated with the 3D projected visual feature.

Denote $\hat{X}_j^{k-1}, j \in \{1,2,3\}$ as the Euclidean coordinates of the three points in $\{C^{k-1}\}$, and recall that X_i^{k-1} is the coordinates of feature i in $\{C^{k-1}\}$. The euclidean position of feature i is computed by solving (1).

$$(X_i^{k-1} - \hat{X}_1^{k-1})(\hat{X}_1^{k-1} - \hat{X}_2^{k-1})(\hat{X}_1^{k-1} - \hat{X}_3^{k-1}) = 0 \quad (1)$$

D. Frame to Frame Motion Estimation

Setup - Let us use right superscript $k, k \in Z^+$ to indicate image frames, and I to indicate the set of visual features. For a feature $i, i \in \mathscr{I}$, that is associated with distance, its coordinates in S k are denoted as $X_i^k = [x_i^k, y_i^k, z_i^k]^T$. For a feature with unknown distance, normalized coordinates are used instead: $\bar{X}_i^k = [\bar{x}_i^k, \bar{y}_i^k, \bar{z}_i^k]^T$, where $||\bar{X}_i^k|| = 1$. The egomotion is modeled as a rigid body transformation. Let R and T be a 3x3 rotation matrix and a 3x1 translation vector describing the frame to frame motion.[11]

$$X_i^k = RX_i^{k-1} + T \quad (2)$$

Substituting the normalized coordinates into (2) gives (3).

$$z_i^k \bar{X}_i^k = RX_i^{k-1} + T \quad (3)$$

With algebraic manipulation to remove the depth z_i^k, we get (4), (5).

$$(R_1 - \bar{x}_i^k R_3)X_i^{k-1} + T_1 - \bar{x}_i^k T_3 = 0 \quad (4)$$

$$(R_2 - \bar{y}_i^k R_3)X_i^{k-1} + T_2 - \bar{y}_i^k T_3 = 0 \quad (5)$$

For a feature with unknown depth, we rewrite (2) using only normalized coordinates,

$$z_i^k \bar{X}_i^k = z_i^{k-1} R\bar{X}_i^{k-1} + T. \quad (6)$$

Using algebraic elimination to remove z_i^k and z_i^{k-1} we get,

$$[-\bar{y}_i^k T_3 + T_2, \bar{x}_i^k T_3 - T_1, -\bar{x}_i^k T_2 + \bar{y}_i^k T_1] R\bar{X}_i^{k-1} = 0. \quad (7)$$

We define θ as $[\theta_x, \theta_y, \theta_z]^T$ which are the rotation angles of camera around the x, y, z axes between $k-1$ and k. We choose to express the rotation matrix using the Rodrigues formula in (8), where $\hat{\theta}$ is the skew symmetric matrix of the θ vector [7].

$$R = e^{\hat{\theta}} = \begin{cases} I + \frac{\hat{\theta}}{||\theta||}\sin||\theta|| + \frac{\hat{\theta}^2}{||\theta||^2}(1 - \cos||\theta||) & \text{if } \theta \text{ is not a small angle} \\ I + \hat{\theta} + \frac{1}{2}\hat{\theta}^2 & \text{otherwise} \end{cases} \quad (8)$$

If there are a total of m features with known depth and n with unknown depth, then when we vertically concatenate (4) and (5) on top of (7), we obtain a nonlinear function F, with $2m+n$ rows, and a $2m+n$ length vector ϵ, the residual.[11]

$$f([T;\theta]) = \epsilon \quad (9)$$

To find the optimal translation and rotation, perform iterative optimization by Levenberg-Marquardt (LM) where J is the Jacobian of F with respect to $[T,\theta]$.[11]

$$[T;\theta]^T] \leftarrow [T;\theta]^T - (J^T J + \lambda diag(J^T J))^{-1} J^T \epsilon \quad (10)$$

V. LOAM

A. Lidar Odometry

The lidar odometry section identifies features in point clouds based on the curvature, and then estimates relative motion by finding correspondence of those features between point clouds. In [9], Zhang provides very clear explanations for feature extraction and finding feature correspondence. To avoid repetition, we focus on the implementation of Zhang's method and only briefly summarize the theory.

Algorithm 1 Frame to Frame Egomotion Estimation

procedure F2F($\bar{X}_i^k, X_i^{k-1}$ or $\bar{X}_i^{k-1}, i \in \mathscr{I}$)
  Initialize θ and T to 0
  for number of iteration: **do**
    for each feature i **do**
      if depth **then**
        $\begin{bmatrix} \text{Eq (4)} \\ \text{Eq (5)} \end{bmatrix} = \epsilon$
      end if
      if no depth **then**
        Eq (7) $= \eta$
      end if
      Compute bisquare weight for each feature based on ϵ
      Update θ and T for one iteration based on (10)
    end for
    if nonlinear optimization converges **then**
      Break
    end if
  end for
  return θ and T
end procedure

1) Feature Extraction: Point cloud features are extracted by their curvature. Let P_k represent the lidar point cloud, and let i be a point in P_k. Let S be the set of consecutive points of i returned by the lidar in the same scan. Thus, the curvature of the local surface is:

$$c = \frac{1}{|S| \cdot \|X_{(k,i)}^L\|} \| \sum_{j \in S, j \neq i} (X_{(k,i)}^L - X_{(k,j)}^L) \| \tag{11}$$

In implementation, each scan is broken down into uniform sub-regions. Each subregion provides up to 2 edge points and 4 planar points. Breaking into subregions promotes uniform spatial sampling and prevents clustering of features which can cause estimation errors. Based on the c value, a point would be classified as either an edge or a planar point where a large c corresponds to edge and a low c corresponds to plane. It is important to not exceed the maximum number of allowed features per subregion, because too many features causes confusion in motion estimation. Also, ambiguous features are not considered as illustrated in Fig 2. The criteria for an ambiguous feature is if the patch of point cloud is roughly parallel to the laser, or if the point is on the boundary of an occluded region. Its important to note that if the lidar moves and the occluded region becomes observable again, then it is no longer an ambiguous feature.

2) Feature Correspondence: To find the correspondence of the feature points, a few preconditions must be met. First, obtain a point cloud from the previous time step and projected into the current time step. Second, receive a new point cloud at the current time step. These point clouds must provide enough

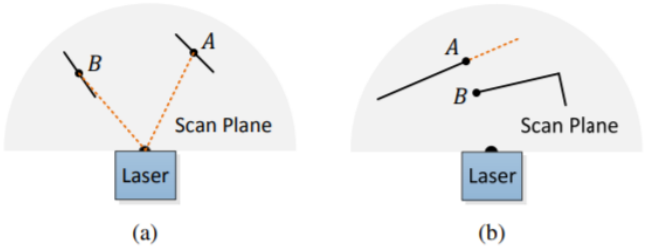

Fig. 2: Example of ambiguous features courtesy of [9]. In (a), point B is on a surface that is almost parallel to the laser beam. In (b), Point A is on the boundary of an occluded surface.

edge/plane features per (11). What remains is to search the point clouds for correspondence between a feature point and an edge line or a planar patch. It is important to realize that in the euclidean space, 2 points uniquely determine a line, and 3 non-colinear points determine a plane.

In implementation, the point clouds are stored in KD-trees with dimension of 3. Approximate nearest neighbor search is used to find corresponding edge or planar points. This method is used as a trade off between accuracy and computation time. To ensure that the points found are edge or planar points, the curvature per (11) is used as a validation gate. Finally, the distance between a point to its corresponding edge/plane is as below:

$$d_\epsilon = \frac{\left| (\tilde{X}_{(k+1,i)}^L - \bar{X}_{(k,j)}^L) \times (\tilde{X}_{(k+1,i)}^L - \bar{X}_{(k,l)}^L) \right|}{\left| \bar{X}_{(k,j)}^L - \bar{X}_{(k,l)}^L \right|} \tag{12}$$

$$d_{\mathscr{H}} = \frac{\left| \begin{array}{c} (\tilde{X}_{(k+1,i)}^L - \bar{X}_{(k,j)}^L) \\ ((\bar{X}_{(k,j)}^L - \bar{X}_{(k,l)}^L) \times (\bar{X}_{(k,j)}^L - \bar{X}_{(k,m)}^L)) \end{array} \right|}{\left| (\bar{X}_{(k,j)}^L - \bar{X}_{(k,l)}^L) \times (\bar{X}_{(k,j)}^L - \bar{X}_{(k,m)}^L) \right|} \tag{13}$$

3) Motion Estimation: Zhang proposes an optimization approach to recover the motion estimate as a 6-DOF rigid body transformation between two time steps $[t_k + 1$ and $t_k]$. Section C of [9] provides a very clear and detailed method to formulate the objective function:

$$f(T_{k+1}^L) = \mathbf{d} \tag{14}$$

where the optimal T_{k+1}^L is obtained by minimizing $\mathbf{d}$ towards 0.

In implementation, the Levenberg-Marquardt iterative method is used to solve this nonlinear least squares problem. First, construct the constraint (12) and (13). Second, assume the translational and angular velocities are constant in order to calculate the jacobian matrices with respect to the linearization point. Third, iteratively obtain the solution. To ensure computational efficiency, KD tree search is done to find feature correspondence every 5 iterations of the LM solver, and we only run the LM solver for 25 iterations. Finally, the resulting pose estimate is in the lidar local coordinates and it is transformed to the world frame.

B. Lidar Mapping

The lidar mapping section matches and registers the undistorted point cloud $\bar{P}_{k+1}$ to the map while fusing the lidar pose generated by the mapping section and the lidar pose provided by the odometry section. The intuition here is that the curvature based feature extraction is not exactly a highly accurate method. Observation noise is inevitable, and we have yet to account for it. Zhang proposes a method to optimize the accuracy of the lidar odometry pose estimate by making the assumption that observations at the current time have more error than the previous time. In other words, we have higher confidence in the constructed map than the lidar odometry pose estimate. Thus, we can use the constructed map to correct the trajectory.

1) Odometry Correction: For a sweep between $[t_k, t_{k+1}]$, let us define the lidar point cloud as $\bar{P}_k$ and the pose estimate as $T_k^L(t_{k+1})$. Let Q_{k-1} represent the accumulated map at t_k, and the lidar pose converted to the map frame as $T_{k1-1}^W(t_k)$. Based on $T_k^L(t_{k+1})$, we can project $T_{k-1}^W(t_k)$ to $\bar{T}_k^W(t_{k+1})$ and also coordinate transform $\bar{P}_k$ into the world frame to get $\bar{Q}_k$. Then, the optimal $T_k^W(t_{k+1})$ registers the point clouds Q_{k-1} and $\bar{Q}_k$. Fig 3 provides an illustration of this process.

In implementation, Q_{k-1} is stored in 10m cubic areas in order to find feature correspondence. If there is an overlap between the points in the cube and the point cloud $\bar{Q}_k$ in the current frame, then we extract and save them in a KD tree. After we find the point in the map Q_{k-1}, we search its neighborhood with a width of 10cm in the vicinity of the feature point. This width is a design choice, which Zhangs implementation chose to be 10cm x 10cm x 5cm. We then find out which cube the current pose estimate $\bar{T}_k^W(t_{k+1})$ belongs to. After finding that cube, we look for the corresponding registration point in a 5x5x5 neighborhood of the cube. Thus, we have obtained the map feature points that are in the neighborhood of the current lidar pose. This is a great thrift on computation time, since we only need to process a few map feature points in these neighboring cubes instead of the the whole map. Also of note is that the point cloud is downsampled to ensure smoothness and also to help computational time.

2) Map Construction: A high resolution map is required to correct the trajectory. Thus, we repeat the same idea as presented in the odometry section, but now with 10 times more point cloud features to register a point cloud into the accumulated map. In this case, the map is simply all the previously registered point clouds.

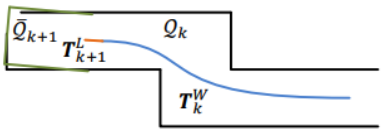

Fig. 3: Illustration of the map registration process, courtesy of [9].

At this point, we have map feature points near the current lidar pose and the point cloud feature points of the current frame. To match them together, KD tree search is used to find the nearest 5 points around a feature point. Zhangs method computes a point cloud covariance matrix and finds its Eigendecomposition. If the five points are distributed on a straight line, the eigenvalue of the covariance matrix contains one element significantly larger than the other two, and the eigenvector associated with the eigenvalue represents the direction of the straight line. If these five points are distributed on a plane, there is a significantly smaller element of the eigenvalue of the covariance matrix, and the eigenvector associated with this eigenvalue represents the normal direction of the plane in which it lies. This method allows us to once again formulate the same optimization problem as presented in the lidar Odometry section, solved by the Levenberg-Marquardt iteration. The final result is a corrected lidar pose estimate and an accumulated map array with the registered point clouds.

3) Real-Time Considerations: In order to maintain real-time operation, the mapping node runs at 1 Hz, a much lower frequency than the odometry node, and it is called only once per sweep. The reason for this is that we need to extract 10 times more features to obtain the high resolution map. Thus, we integrate the odometry pose estimate with the optimized pose estimate by simply using one or the other. In short, if there is an optimized pose estimate, use it. If not, use the odometry pose estimate. This allows the mapping section to also output a pose estimate at 10 Hz.

VI. Results

A. Odometry Results

Results of the LOAM algorithm odometry can be seen in Fig 5. The figure shows the pose of the vehicle for Sequence 08 from the residential KITTI Raw Data site [6]. The data file describes a seven minute long drive and about 3 km of translation, featuring seventeen turns and three separate loop closures. We found that our implementation of LOAM had slightly higher translation and rotation error than the claims in the paper and those found on the KITTI Vision Benchmark Suite [5].

In order to plot pose and calculate error, we downloaded the KITTI Odometry Ground Truth Poses [5] and wrote the results of our odometry from ROS to a text file. Post processing was done via a Matlab script that can be found on our Github repository [1]. We implemented a similar algorithm to the KITTI benchmarking evaluation based on the equations below in order to compare our results accordingly[5]:

$$E_{rot}(F) = \frac{1}{|F|} \sum_{(i,j)\in F} \angle[(\hat{p}_j \ominus \hat{p}_i) \ominus (p_j \ominus p_i)] \tag{15}$$

$$E_{trans}(F) = \frac{1}{|F|} \sum_{(i,j)\in F} \|(\hat{p}_j \ominus \hat{p}_i) \ominus (p_j \ominus p_i)\|_2 \tag{16}$$

$E_{rot}(F)$ and $E_{trans}(F)$ are the rotation and translation error respectively, and $\hat{p}$ and p are the results from the algorithm and results from ground truth respectively. The features are denoted by F and the current measurement and previous measurement are denoted by j and i respectively. The equations are general representations of taking the difference between vector positions and angles and then normalizing them about the features. In doing so, we can see from Table I, there is a small translation error difference with what was posted on KITTI benchmarking. However, the rotation error was substantially worse than what was benchmarked on KITTI. When plotting the errors and position plots iteratively in Fig 4 we see that the spikes in rotation error are mostly occurring during turns. This is likely due to the constant velocity assumption between lidar sweeps and the lidar sweep frequency is too low for fully sample the quick turns.

This can be confirmed by contrasting the error during during straightaways. In addition, the data sets that are being compared are not the same since the KITTI website shows results for Sequence 11-21, while our results are referencing Sequence 08. We were not able to compare the results on those same sequences since the KITTI website does not provide ground truth results for those sequences. Another contributing factor to both the translation and rotation error difference can be due to the divergence that seems to occur after running large datasets on a computer with suboptimal CPU specifcations. We found that we needed to run the data files at slower rates to reduce the divergence. The algorithm does not have global initialization, only local, which leads to having an initial rotation and translation error, which can be seen in Fig 5. We corrected for these errors with our Matlab script, which better shows the error in the LOAM algorithm.

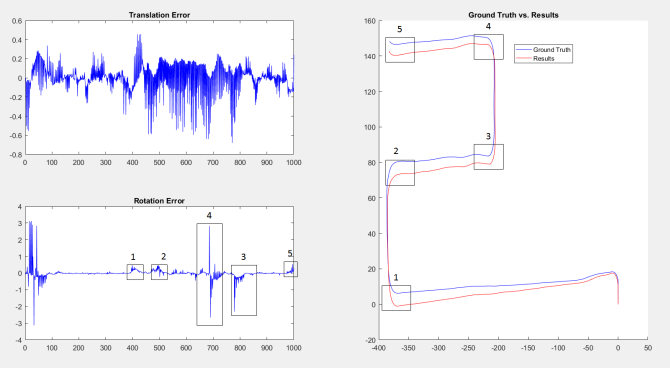

Fig. 4: Sequence 08 run showing the errors through time of the first few turns.

B. Mapping Output

Fig 5 shows the results of our LOAM algorithm in orange, with ground truth from integrated IMU data in blue. On the x-axis is the z-axis in the LOAM algorithm (longitudinal motion) and on the y-axis is the x-axis from the LOAM algorithm (lateral motion). We ignored vertical motion for plotting purposes. The overall map of the lidar point cloud is shown in Fig 6, with two zoomed in images in Fig 7 and Fig 8. The point clouds turn out very good for mapping the overall environment, but there are some issues with seeing streaks of the same car between frames as shown in Fig 7. When zooming in to a scene with stationary vehicles as shown in Fig 8, it is evident that the map is very detailed and accurate. A detailed video of the full map can be found on YouTube at **https://www.youtube.com/watch?v=YTCnmP5RBuE**.

TABLE I: LOAM Error Results

LOAM Implementation	Translation Percent Error (%)	Rotation Error ($^{deg}/_{m}$)	Absolute Position Error (m)
LOAM - Our Sequence 08 Results	0.81	0.732	12.24
LOAM - KITTI Dataset Results [9]	0.61	0.0014	N/A

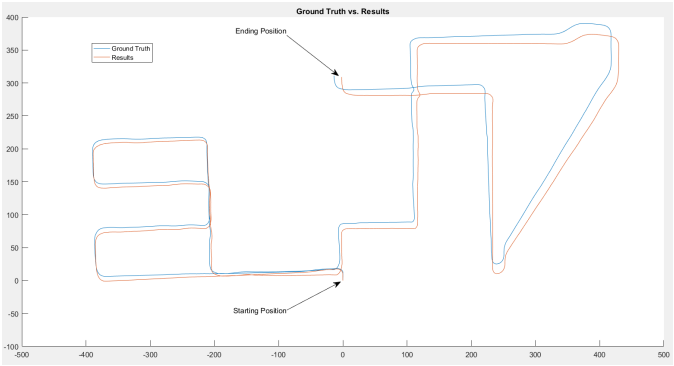

Fig. 5: Sequence 08 run of ground truth and algorithm results' position over time.

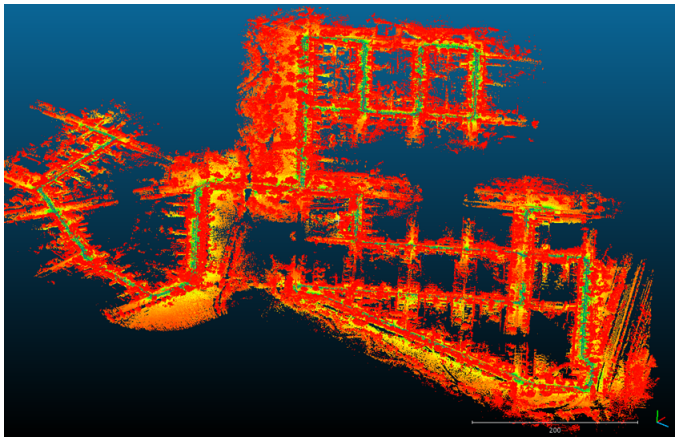

Fig. 6: Sequence 08 run of 3D point cloud of full map.

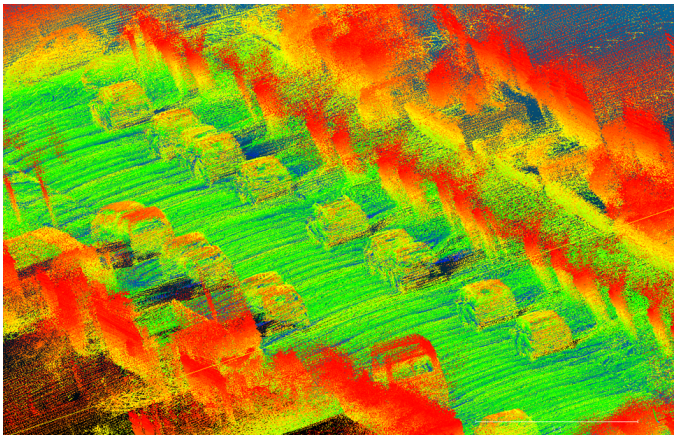

Fig. 7: Sequence 08 run of 3D point cloud zoomed in on road with cars.

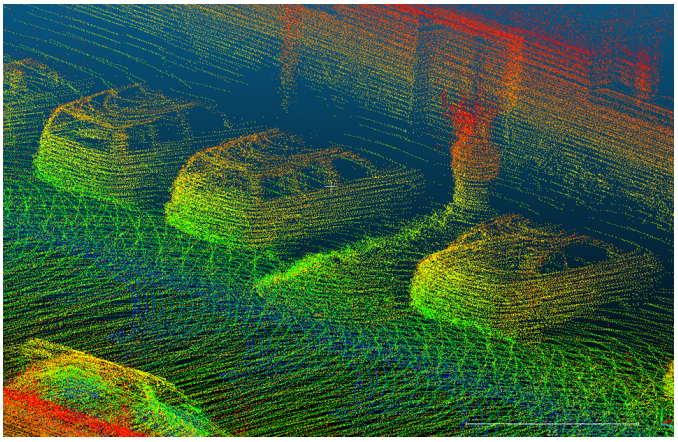

Fig. 8: Sequence 08 run of 3D point cloud zoomed in on parking spaces with cars in them.

VII. Challenges

A. OpenCV and ROS

Our original intent was to create our own version of the V-LOAM algorithm based on the open source implementations of DEMO and LOAM. However, we require extra modules from the OpenCV contrib library which were not included in ROS-kinetic by default. Building from source, resolving build errors, and releasing third party packages into ROS required a time overhead that we were not prepared for. In the end we could not resolve all errors with OpenCV and decided that our remaining time would be better spent on delivering a fully functioning LOAM.

A considerable amount of time was also spent understanding the ROS environment. Between understanding the subscriber-publisher protocol and converting raw KITTI data into ROS .bag files, it took longer than expected to simply create a ROS node that subscribed to KITTI data and published it for visualization. In addition, homogeneous transformations needed to be understood at a deeper level in order to visualize data in the correct frame of reference in RVIZ.

B. Computer Performance Issues

Virtual Machines were used to quickly get up and running in a Ubuntu development environment. However, unanticipated issues such as incompatible ROS versions, low disk space due to large data size, and suboptimal CPU performance all contributed to time spent on setup instead of development. We also wanted to visualize the maps after playback of a .bag file in order to validate the mapping portion of the algorithm. However, large amounts of memory are required to visualize a map that is 9GB large. In addition, large amounts of disk space are needed to store and work with the complete KITTI odometry dataset. For example, the largest sequence, Sequence 08 was 29GB of data once unzipped and converted to a .bag file. We didn't anticipate this when originally creating our virtual machines, so some members of the group had to create new virtual machines with more disk space set aside.

C. Initialization Assumptions

Finally, when trying to assess performance, we noticed extremely high translation and rotational errors. Upon plotting the results, we noticed large offsets between our trajectory and the ground truth. An investigation into the provided KITTI odometry devkit identified the solution, which is that the ground truth is only provided for sections of the raw data. We simply needed to remove extra data in order to have a fair comparison with the ground truth. After correcting for this, we obtained more satisfactory results as seen in the results section.

VIII. Algorithm Limitations

The paper we investigated did not incorporate loop closure because the authors wanted to minimize drift by strictly using odometry. We followed the same type of implementation in order to compare our results to their results from the KITTI dataset website. As a result, a key limitation to this algorithm is the lack of drift correction with loop closure. As such, this isn't considered a true SLAM algorithm, but the rotation and translation error has been shown to be very small in the results section and in the KITTI dataset benchmarking. The algorithm can be extended to incorporate loop closure to overcome this limitation and achieve better results.

One of the assumptions in the paper by Zhang and Singh indicate they use a constant velocity motion model between each lidar sweep in the algorithm. This is not an optimal assumption as vehicles tend to accelerate and decelerate substantially during a drive, especially during a turn. As shown in the results, this constant velocity assumption causes the rotation error to jump to significant values during a turn. In addition, motion distortion in the point cloud is amplified during turns, which is another source of error in LOAM.

Point cloud registration is feature based and more sparse than traditional brute force methods such as iterative closest point. As a result, it saves computation time, but is not exhaustive and may miss features. The subregions that are made to counteract issues with missing features is set up to have a uniform amount of features throughout the frame

causing each subregion to have a certain number of features. While this is easily achieved in urban areas that are feature rich, the algorithm may have trouble in rural or dessert areas that have less features. These areas can cause the odometry to be less accurate since the algorithm performs a better motion estimation with more features in the frame.

When strictly using lidar for odometry there is a hardware limitation as the lidar rotates at a fixed rate. In the case of Velodyne HDL-64E, it rotates at 600 RPM which results in collecting data at 10Hz. Other sensors such as cameras can observe the environment at 60 Hz and can therefore miss less changes in the environment between frames. This becomes more of an issue when traveling at high speeds with the lidar. In addition, a supplement to the algorithm input in an IMU, which in this case costs $35,000. Using a camera as a substitute for this IMU would greatly reduce the cost and allow this type of algorithm to enter a production environment. The lidar data is composed of 2.2 million data points per second. This can be seen in many of the KITTI datasets which reach about 30 GB for a seven minute, 3 km drive. The algorithms map output also does not get rid of many points so the detailed map turns out to be about 20 GB. This may not be a significant concern as the algorithm can be run in real time and cheap hard drives can be used to collect the data.

With regards to mapping, the algorithm does not reject moving/stationary objects in the frame. This can cause the map to have streaks of moving objects and stationary objects that are not part of the environment such as parked vehicles, throughout the map. Without a rejection of certain features, the implementation of loop closure could also suffer since it may not identify enough similarly seen features if many features are part of the map that shouldn't be.

IX. Conclusion and Future Directions

The team was successfully able to modify an opensource implementation of LOAM, built for a Hokuyo 180 degree lidar, and to work with the Velodyne HDL64E with the KITTI data set. We intended to do the same for DEMO but were not successful in the time allotted. Our plan is to get DEMO working with KITTI dataset and then integrate DEMO and LOAM into VLOAM. Then release it as the only open source V-LOAM for ROS.

There is a small translation error difference in our LOAM, compared to that of what is posted on KITTI LOAM results. Rotational error difference was significantly larger due to linearization of motion during cornering. We believe vision based motion prior which is implemented in the V-LOAM algorithm would improve the rotation error due to higher frequency and multimodal sampling of motion.

Camera data combined with robust depth information from lidar can be used to create precise odometry and point cloud maps. For the foreseeable future, the autonomous vehicle sensor suite will contain lidars, radars, and cameras. The ultimate goal is to implement a safe autonomous vehicle at scale. Advancements in mapping algorithms will allow autonomous vehicles to become safer and more robust faster.

References

[1] Team 18. 568-final-project. https://github.com/stevenliu216/568-Final-Project, 2017.

[2] P. Henry M. Krainin D. Maturana D. Fox A. Huang, A. Baschrach and N. Roy. Visual odometry and mapping for autonomous flight using an rgb-d camera. In *Int. Symposium on Robotics Research (ISRR)*, 2011.

[3] J. Sturm C. Kerl and D. Cremers. Robust odometry estimation for rgb-d cameras. In *IEEE International Conference on Robotics and Automation*, 2013.

[4] L. Zhao A. Alempijevic G. Hu, S. Huang and G. Dissanayake. A robust rgb-d slam algorithm. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct 2012.

[5] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

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

[7] R. Murray and S. Sastry. *A mathematical introduction to robotic manipulation*. CRC Press, 1994.

[8] E. Herbst X. Ren P. Henry, M. Krainin and D. Fox. Rgb-d mapping: Using kinect-style depth cameras for dense 3d modeling of indoor environments. *The International Journal of Robotics Research*, 31(5):647–663, 2012.

[9] Ji Zhang and Sanjiv Singh. Loam: Lidar odometry and mapping in real-time. In *Robotics: Science and Systems Conference*, July 2014.

[10] Ji Zhang and Sanjiv Singh. Visual-lidar odometry and mapping: Low-drift, robust, and fast. *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, May 2015.

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