Robust LiDAR Localization on an HD Vector Map without a Separate Localization Layer

Chi Zhang Liwen Liu Zhoupeng Xue Kun Guo Zhiwei Li

***Abstract*— Many autonomous driving applications nowadays come along with a prebuilt vector map for routing and planning purposes. In order to localize on this map, traditional LiDAR localization methods usually require a separate localization layer to function. On one hand, the separate layer occupies large storage and is not convenient to update. On the other hand, the potential of the vector map itself has not been fully exploited by existing methods. In this paper, we present a LiDAR localization system that leverages the vector map directly as the localization layer. A semantic extraction module is developed to match the heterogeneous data between LiDAR measurements and the 3D vector elements. A local map maintenance module is introduced to keep the system function robustly when there are not enough vector matches. The system adopts an optimization-based framework and infers 6-DOF poses. Experiments show that the proposed system is able to achieve centimeter accuracy robustly in both highway and urban environments, without a separate localization layer.**

I. Introduction

Accurate vehicle localization is a fundamental prerequisite for high-level autonomous driving. In many modern autonomous driving applications, the task of localization is to compute a pose on a predefined high-definition (HD) vector map so that a planning module can consume this pose and output real-time planned trajectories on this vector map. Typically, an HD vector map consists of traffic elements like lanes, road markings, traffic signs, poles, etc., represented as vectorized 3D shapes.

Current LiDAR localization techniques require a separate localization layer to function, such as a 2D reflectivity grid[1], a geometry surface map represented as Normal Distribution Transform (NDT)[2], etc. In order to localize the vehicle on the HD vector map, the vector layer and the separate LiDAR localization layer must be strictly aligned.

However, maintaining a separate localization layer comes with two disadvantages. The first is the larger map size that is required to store the sensor-dependent data in the localization layer, which does not scale well to large urban areas. The second is the inconvenience when it comes to updating the map. Whenever an old map part needs to be updated, or a new portion needs to be added, the corresponding localization layer must be regenerated. Regeneration is not only cumbersome but also is not possible in some cases if we had outsourced the map creation process to other dedicated mapping companies.

On one hand, using separate localization layers will incur some costs. On the other hand, the potential of the vector map has not been fully exploited. Motivated by this issue, we propose to leverage the vector map directly as the localization layer. However, this will be a challenging task itself since we need to associate heterogeneous data, i.e., the LiDAR point clouds and the sensor-agnostic 3D vectors.

Inspired by the recent development of deep learning, we use convolutional network[3] to extract semantics from the online LiDAR point clouds and match the extracted semantic points against the 3D vectorized elements. To deal with situations where there are not enough vector elements to match with, we also introduce a local map maintenance mechanism, so that short-term ego-motion can still be robustly tracked without the vector map. We found that vector elements such as lane lines, stop lines are very effective in constraining lateral and longitudinal positions respectively, while poles are effective in both.

The contribution of this paper is summarized as follows:

- A vector map matching module that associates heterogeneous data between LiDAR sweeps and the 3D vectors, based on the semantic extraction of lane lines, stop lines, and poles from the LiDAR sweeps.
- A fast local map maintenance module that tracks mid-term ego-motion and keeps the system function robustly when there are no sufficient vector matches.
- A complete 6-DOF LiDAR-based localization system that achieves comparable accuracy with existing methods without using a separate localization layer.

II. Related Work

In the last decades, researchers have developed a large number of LiDAR-based localization approaches. We briefly review them according to the map representation used.

A. Methods Using Separate Localization Layers

Levinson et al. [1] represent the map as a 2D grid of LiDAR's reflectiveness and performs online localization by a particle filter. The authors later extended their work in [4] by replacing the fixed grid with a probabilistic grid. Wolcott and Eustice[5] further extend the method by introducing for each grid a Gaussian mixture distribution of the height. Wan et al. [6] augments a height attribute to the grid and adopts an error state Kalman filter for localization.

Instead of using 2D grids, another type of method stores the map as a 3D point cloud accumulated from the LiDAR sweeps. Many SLAM methods [7] and [8] belong to this category. Localization is done in an ICP[9] fashion. Instead of using raw point cloud, Magnusson et al.[2] represent the geometry surfaces as a set of Normal Distribution Transforms (NDT), which increases the localization's robustness. Behley

All authors are with DeepMotion Co., Ltd. Beijing, China.

and Stachniss[10] include normal information computed from the LiDAR sweeps and represent the map as a set of oriented points, called surfel map. Chen et al.[11] introduces semantic labels to the surfel map and makes the system more robust to dynamic objects.

Despite relatively high accuracy, the above methods require large storage for their localization layers, which do not scale well to large scenes.

B. Methods Using Vector Maps

Localization against a vector-form map has been addressed by some previous works. Schreiber et al.[12] propose to use a vector map and a stereo camera for localization by matching detected lane line points with the 3D vector. Poggenhans et al.[13] segment out road markings from accumulated top view images computed from stereo cameras and then vectorize the detections to match with the vector map. Jeong et al.[14] also follow the stereo setup but represent the detection as an 8-bit top view image. Localization is performed using a corresponding bitwise particle filter. Some other visual approaches[15], [16], [17] then follow, using different matching strategies.

Biswas and Veloso[18] propose to localize an RGB-D camera on a 2D vector map consisting of 2D line segments, which correspond to the boundaries of static obstacles. During online localization, it performs a RANSAC-based plane fitting on the depth image and down-projects the plane filtered points onto 2D, to match with the 2D vector map. Ma et al.[19] feed the front view image and the projected LiDAR intensities on a bird's-eye view (BEV) to a multi-sensor convolutional network to obtain an inverse truncated distance image to a 2D lane graph in the overhead view. The distance image is then used to localize in the lane graph in the histogram filter framework. The method depends on camera input and deals mainly with highway environments where lane structures are relatively simple.

Javanmardi et al.[20] propose to localize a multi-channel LiDAR in the "abstract map", which consists of a set of vectorized 3D planar surfaces of the building walls. An NDT map is then converted from these vectors for an NDT-based localization. As the most related work to ours, we differ from Javanmardi et al. in two major aspects: 1) We match the LiDAR sweep semantically with the vector map rather than pure geometrically, where we employ a convolutional network to extract semantic labels from the LiDAR sweep before associating match candidates. It is not possible to match important elements such as lane lines that are blended with the road surface if a pure geometry matching approach is used. 2) There are real-world situations where insufficient vector matches or biased vector matches can lead to drift, we propose a fast local map maintenance module that keeps the system function robustly even under these situations.

III. Approach

Fig. 1 provides an overview of the proposed approach. The method adopts an optimization-based multi-sensor fusion framework. It consumes the online LiDAR sweeps, IMU measurements, and wheel encoder readings as inputs and outputs optimized poses.

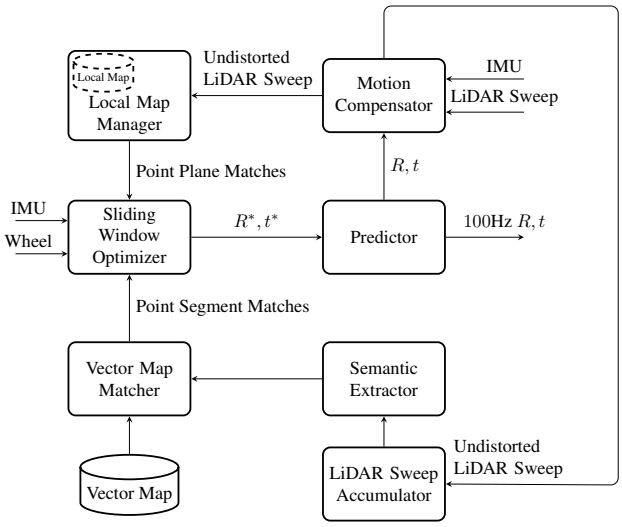


Fig. 1: An overview of the proposed localization system.

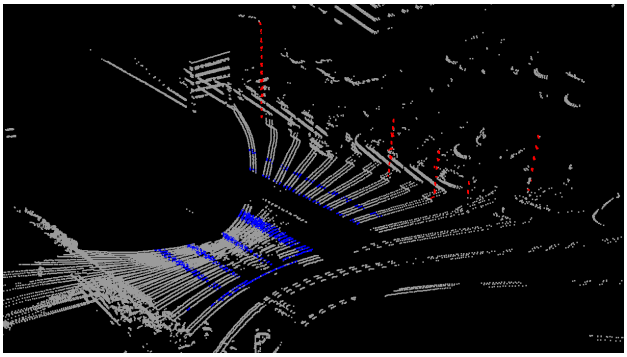


Fig. 2: Semantic extraction: four sparse VLP-16 LiDAR sweeps are concatenated into a denser LiDAR point cloud. Lane line and pole extractions are performed on the concatenated point cloud.

A. Semantic Extraction

In contrast to the traditional ICP based localization methods, the autonomous driving vector map we use here carries semantics. Some important semantics classes here, such as lane lines, cannot be directly matched by ICP since they are blended with neighboring points in the road surfaces. To deal with this issue and to better exploit the semantic info carried in the HD vector map, we design a semantic extraction module to assign semantic labels to the online LiDAR sweeps using convolutional networks. We adopt RangeNet++[3] for such task.

However, two problems prevent us from directly adopting RangeNet++. First, we are using a VLP-16 LiDAR in our experiments. The sweeps are four times sparser than the HDL-64E used in [3], which does not generate satisfying results. Second, the open datasets[21] used for training in RangeNet++ does not contain the semantic classes, i.e., lane lines, poles, we want in our localization context.

To deal with the first problem, we maintain a light-weight sweep accumulator. The accumulator assembles the

Algorithm 1 LiDAR sweep vector map matching

Input: Vector map $\mathcal{M}$; LiDAR sweep $\mathcal{P}$; Current pose $\mathbf{T}_L^W$.
Output: The set of point-to-line matches Ω.

$\Omega = \emptyset$
$\mathcal{S}$ = SemanticExtraction($\mathcal{P}$)
$\mathcal{O}$ = BuildOctree($\mathcal{S}$)
$\mathcal{L}$ = RadiusSearch($\mathcal{M}, \mathbf{T}_L^W, 50m$)
for each lane line l_i in lane line set $\mathcal{L}$ **do**
  for each one meter segment $\mathbf{c}_j^1\mathbf{c}_j^2$ in l_i **do**
    $\mathbf{f}_j$ = NearestNeighborSearch($\mathcal{O}$, $\frac{\mathbf{c}_j^1+\mathbf{c}_j^2}{2}$)
    if dist($\mathbf{f}_j, \mathbf{c}_j^1\mathbf{c}_j^2$) $< \tau$ **then**
      $\Omega = \Omega \cup (\mathbf{f}_j, \mathbf{c}_j^1\mathbf{c}_j^2)$
    end if
  end for
end for

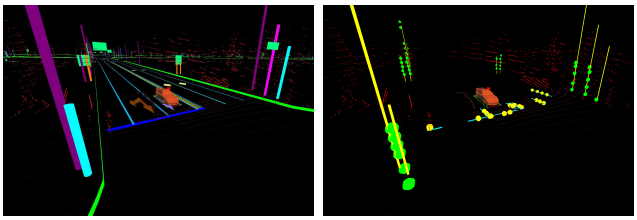
four most suitable recent sweeps into a bigger point cloud that resembles the density of an HDL-64E sweep. The four candidate sweeps are chosen from recent history. Their poses are evenly spaced so that the synthesized point cloud will have a more uniform vertical density than an ad-hoc selection of the most recent four. The four candidates are then transformed to the local frame of the most recent sweep, which is then fed into RangeNet++. Fig. 2 shows an example of the concatenated sweeps. An accumulation using more sweeps can also be used, but we found no apparent gain on the localization result.

To deal with the second problem, we propose an automatic approach to generate the training data. We first accumulate a dense local point cloud from the sparse LiDAR sweeps. The accumulation requires a LiDAR trajectory which can be obtained by existing LiDAR odometry methods[7], [8]. Since our localizer contains a local map management module (detailed in Section III-C), we use the local map output by our localizer instead. A typical local map is shown in Fig. 4. We adopt [22] to extract pole-like objects in the dense point cloud. Similarly, we adopt [23] to extract road markings but keep only those with line shape. The line-shaped road markings mean those whose major axis's span largely exceeds its minor axis's span in Principle Component Analysis. After automatic extractions, the point cloud labels are directly mapped back to the LiDAR sweeps for training. Note that it does not matter if the trajectory contains drift since we are only concern about semantic label learning.

B. Vector Map Representation and Matching

The HD vector map represents the traffic elements as a set of vectorized 3D shapes with semantic labels. Four types of elements are exploited for matching: poles, lane lines, stop lines, and road planes.

Lane line matching. The lane line matching process produces a set of point-to-line-segment matches. First, we extract lane line points from the current LiDAR sweeps using the semantic extraction module. An octree index is then created for the lane line point set. We then divide each vector lane line into a set of one-meter line segments and finds the nearest neighbor in the octree for each midpoint in these line segments. We thus obtain a set of point-to-line segment matches, denoted as $\{\mathbf{p}_j, \mathbf{c}_j^1\mathbf{c}_j^2\}$. Algorithm 1 describes this process in detail. Note that since the octree involved is very small (e.g., 1000 points), it causes no burden to real-time performance. The **matching of poles and stop lines** follows the same procedure of lane line matching. Fig. 3 illustrates example matches between the LiDAR sweep and the HD vector map.



(a) (b)

Fig. 3: Vector map matching. (a) The LiDAR sweep overlayed on the HD vector map. The VLP-16 LiDAR is 30° slanted towards the road. (b) The point-to-segment matches between the sweep and the vector map.

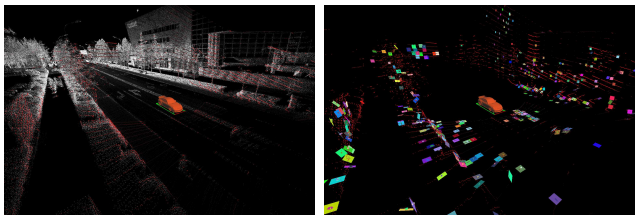
It is worth noting that instead of finding for each LiDAR sweep point a matched entities in the map, as in traditional approaches[2], [20], we are matching in the reversed direction. We found for each vector primitive a matched point in the LiDAR sweep. This allows us the create more uniform matches spatially.

Road plane matching. Road plane provides effective constraints for the vehicle's roll, pitch, and height. The vector map engine allows us to retrieve the closest pre-stored road plane given a query position. We set the query point to the virtual point ten meters ahead of the vehicle. We keep at most one inlier match (0.3m) for every 2m×2m tile in the xy-plane. We can query more road planes around the vehicle, but we found a single road plane already works well in our experiment environments. The road plane matching process produces a set of point-plane matches.

C. Local Map Matching and Maintenance

Using the vector map matches alone for localization might work for the majority of cases but could also fail when there are insufficient vector matches or when the matches are biased. To overcome this problem, we propose to maintain a local map of oriented LiDAR points to help track the vehicle's ego-motion. The local map stores geometry details in the environment that are not represented in the vector map. Therefore, the matching between the LiDAR sweeps and the local map can still keep the vehicle from drifting until good quality vector matches are observed again. A hash-based representation and a hash-based matching strategy are introduced to guarantee real-time performance.

Local map representation. The local map we maintain spans a 100m radius and consists of a set of oriented points accumulated from the recent LiDAR sweeps. Each



(a) (b)

Fig. 4: Local map matching. (a) The LiDAR sweep overlayed on the local map. The local map has a 100m radius and contains points with normals. (b) The point-to-plane matches between the sweep and the local map.

point's normal (orientation) is computed from the cross product from the adjacent LiDAR points in the LiDAR sweep. Matching with and updating such a map could be computation-intensive. For real-time performance, we store the map using a hash table $h : \mathbb{Z}^3 \mapsto \mathbb{R}^3 \times [0,1]^3 \times \mathbb{R}$. The hash table maps a voxel index (i_x, i_y, i_z) to at most one oriented point $(\mathbf{v}_s, \mathbf{n}_s, r_s)$, which stores a vertex position, a normal, and a stability scalar respectively We use a voxel size of 30cm. Since a VLP-16 LiDAR has an effective range of 0-100m, drift in the local map is limited.

Local map matching. We down-sample the LiDAR sweep spatially, keeping at most one point per $2m^3$. We then search for each downsampled point the nearest neighbor (NN) in the local map. To serve the NN query quickly, we search for occupied voxels in a two-layer neighborhood of the query voxel: $\{i_x+\Delta i_x, i_y+\Delta i_y, i_z+\Delta i_z\}$ where $\Delta i_x, \Delta i_y, \Delta i_z \in \{-2,-1,0,+1,+2\}$. The search expands outwards and stops immediately when a match is found. The whole process has a $O(1)$ time complexity with at most 27 checks. The two-layer neighborhood covers a space of $1.5m^3$, which already satisfies our matching needs. Fig. 4 shows an example of the local map and the matches found.

Local map update. After the current LiDAR pose has been optimized, we contribute the sweep into the local map. For each oriented point in the LiDAR sweep, if its corresponding voxel is empty, we create a new entry in the hash table. Otherwise we differentiate two cases:

1) If the normal of the oriented point s' is compatible with the oriented point s of the voxel, we update the point by an exponential moving average

$$\mathbf{v}_s \coloneqq (1-\gamma)\mathbf{v}_s + \gamma \mathbf{v}_{s'}$$
$$\mathbf{n}_s \coloneqq (1-\gamma)\mathbf{n}_s + \gamma \mathbf{n}_{s'}$$

2) Otherwise, we replace the old oriented point by the new one, and mark the voxel as unstable. Unstable voxels will receive lower weights in optimization.

To avoid an over-growing map, old parts of the local map are regularly retired. In our current implementation, when the vehicle has accumulated a 50m distance since the last local map clean-up, we retire the outer parts of the map that are beyond the predefined radius. This is done by a full scan of the hash table.

D. Sliding Window Optimization

We optimize the corresponding IMU states for the recent N LiDAR sweeps

$$\mathbf{x}_i = [\mathbf{p}_i^\top \ \mathbf{v}_i^\top \ \mathbf{q}_i^\top \ \mathbf{b}_{a\,i}^\top \ \mathbf{b}_{g_i}^\top]^\top \tag{1}$$

where i indexes the LiDAR frames. $\mathbf{p}$ and $\mathbf{v}$ are the IMU's position and velocity in world coordinates. $\mathbf{q}$ is the quaternion from the IMU body frame to the world frame. $\mathbf{b}_a$ and $\mathbf{g}_g$ are respectively the accelerameter and gyroscope bias. The sliding window size N ranges in $\{2,\dots,5\}$ in practice. The objective function is

$$\min_{\mathbf{x}_1,\dots,\mathbf{x}_N} \Bigg\{ \sum_{i=1}^{N} \left(\|\mathbf{r}_{\text{PPM}}(\mathbf{x}_i)\|^2 + \|\mathbf{r}_{\text{PLM}}(\mathbf{x}_i)\|^2 + \|\mathbf{r}_{\text{WS}}(\mathbf{x}_i)\|^2 \right) + \sum_{i=1}^{N-1} \|\mathbf{r}_{\text{IP}}(\mathbf{x}_i, \mathbf{x}_{i+1})\|^2_{\boldsymbol{\Sigma}} \Bigg\} \tag{2}$$

where $\mathbf{r}_\bullet$ represents a residual vector. The subscript abbreviations stand for PPM (point-plane-match), PLM (point-line-match), WS (wheel speed) and IP (IMU preintegration) respectively. This is a non-linear least squares problem, which can be solved by the Gauss-Newton method. We employ Ceres Solver[24] for the optimization.

1) The point-line-match and point-plane-match factors. As described in section III-B and III-C, the LiDAR-vector map matches and the LiDAR-local map matches are ultimately transformed into a set of point line matches and point plane matches, which can be expressed as

$$\mathbf{r}_{\text{PPM}}(\mathbf{x}_i) = \sum_j (\mathbf{f}_j^W - \mathbf{c}_j)^\top \mathbf{n}_j \tag{3}$$

$$\mathbf{r}_{\text{PLM}}(\mathbf{x}_i) = \sum_j \frac{\|(\mathbf{f}_j^W - \mathbf{c}_j^1) \times (\mathbf{f}_j^W - \mathbf{c}_j^2)\|}{\|\mathbf{c}_j^1 - \mathbf{c}_j^2\|} \tag{4}$$

where $\mathbf{f}_j^W = \mathbf{q}_i \otimes \mathbf{f}_j + \mathbf{p}_i$ denotes the sample point $\mathbf{f}_i$ detected from the LiDAR frame, transformed into world coordinates; $(\mathbf{c}_j, \mathbf{n}_j)$ represents the center and normal of a point-plane-match's plane in world frame; $(\mathbf{c}_j^1, \mathbf{c}_j^2)$ denotes the two endpoints of a point-line-match's line segment, in world coordinates. Eq. 3 and Eq. 4 represent the point-to-plane distance and the point-to-line distance respectively.

2) The wheel speed factor. The factor wants the vehicle velocity obtained respectively from the IMU and the wheel encoders to agree.

$$\mathbf{r}_{\text{WS}}(\mathbf{x}_i) = \mathbf{R}_I^V \left(\mathbf{q}_i^{-1} \otimes \mathbf{v}_i + [\omega_i - \mathbf{b}_{g_i}]_\times \mathbf{t}_V^I \right) \tag{5}$$
$$- [0, \bar{v}_i, 0]^\top \tag{6}$$

where $[0, \bar{v}_i, 0]^\top$ is the vehicle velocity observed by the wheel encoders, expressed in the vehicle frame. The right hand side of Eq. 5 is the same vehicle velocity but observed by the IMU, which is the sum of the IMU body's velocity $\mathbf{q}_i^{-1} \otimes \mathbf{v}_i$ and the lever-arm compensation $[\omega_i - \mathbf{b}_{g_i}]_\times \mathbf{t}_V^I$ induced by the angular velocity $\omega_i - \mathbf{b}_{g_i}$. $\mathbf{R}_V^I$ and $\mathbf{t}_V^I$ denote respectively the rotation and translation from frame V (vehicle) to frame I (IMU).

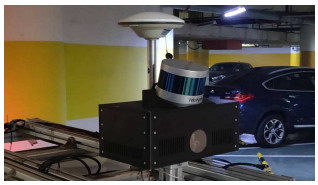
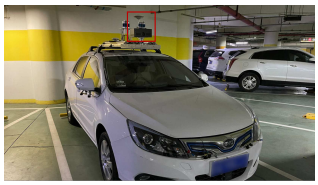



Fig. 5: The test platforms used in our experiments. The IMU is located in the box right under the VLP-16 LiDAR.

3) The IMU preintegration factor. The IMU preintegration factor wants a LiDAR sweep's starting state and ending state to agree the IMU measurements accumulated within this LiDAR sweep. Note that the ending moment of a LiDAR sweep is the starting moment of the next LiDAR sweep. The factor is expressed as

$$\mathbf{r}_{\text{IP}}(\mathbf{x}_i, \mathbf{x}_j) = \Delta\mathbf{x}_{ij} - \Delta\hat{\mathbf{x}}_{ij} \tag{7}$$

where $\Delta\mathbf{x}_{ij}$ is the IMU preintegration between frame i and frame j, defined as

$$\Delta\mathbf{x}_{ij} \triangleq \begin{bmatrix} \mathbf{R}_i^\top(\mathbf{p}_j - \mathbf{p}_i - \frac{1}{2}\mathbf{g}\Delta t^2 - \mathbf{v}_i\Delta t) \\ \mathbf{R}_i^\top(\mathbf{v}_j - \mathbf{g}\Delta t - \mathbf{v}_i) \\ \mathbf{q}_i^{-1} \otimes \mathbf{q}_j \\ \mathbf{b}_{aj} - \mathbf{b}_{ai} \\ \mathbf{b}_{gj} - \mathbf{b}_{gi} \end{bmatrix} \tag{8}$$

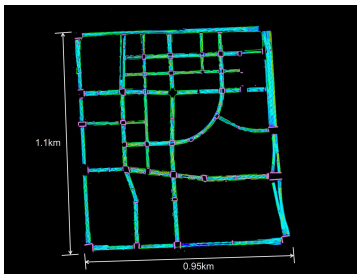
Here, $\mathbf{g}$ denotes the gravity in world frame; Δt denotes the time elapsed between the two frames. The $\Delta\hat{\mathbf{x}}_{ij}$ is also the IMU preintegration, but computed from the IMU measurements between t_i and t_j, instead of from $\mathbf{x}_i$ and $\mathbf{x}_j$. Please refer to Qin et al.[25] for a detail derivation.

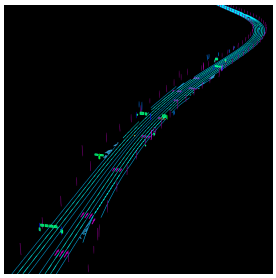
E. State predictor

The optimized poses obtained from the sliding window are at least one frame late against the latest sensor measurement. To compute real-time poses, we maintain a dedicated state predictor. The predictor internally holds an IMU *base state* obtained from the optimization and the IMU stream that arrives later than the base state. Given a pose prediction request with a timestamp, the predictor responds by integrating the cached IMU measurements starting from the base state and ending upon the input timestamp. The cached IMU stream should be kept small to avoid large noise accumulation. As such, the base state is updated immediately from the sliding window when a new optimized state is available. As a result, the predictor's base state is updated at 10Hz, while the localization results can be outputted at a higher custom rate, e.g., 100Hz.

IV. Experiments

Fig. 5 shows the test platform. The VLP-16 LiDAR is configured to spin at 10Hz. The OpenIMU and wheel encoders report measurements respectively at 100Hz and 50Hz. A NovAtel PwrPak7D-E1 INS module is used as a ground truth generation device. We use two HD vector maps for testing, as shown in Fig. 6. One is a $1.1km \times 0.9km$ size urban area. The other is a $11km$ highway. Both maps are created by a third-party mapping company using high-accuracy mobile-mapping solutions. The maps are geo-referenced in the WGS84 coordinates. We convert them into a local ENU (East-North-Up) coordinate frame for matching, where the local ENU frame's origin is set to the map data's centroid.





(a) Zhongguancun urban map (b) G7 highway map


(c) An urban image sample (d) A highway image sample

Fig. 6: The urban and highway HD vector maps in our experiments.

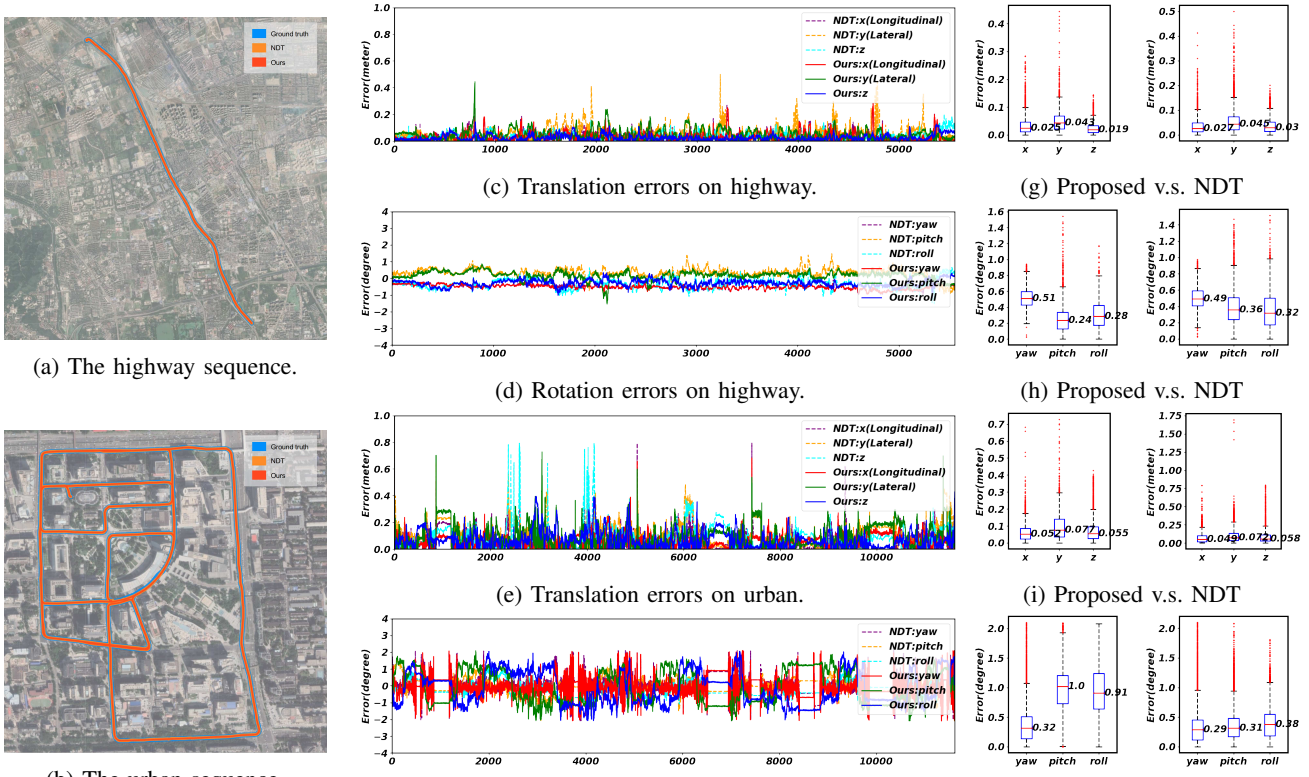
Method	2D ground reflectivity images[1]	NDT point clouds[2]	Ours
Map storage requirement	25MB	314MB	0.62MB
Mean accuracy	$0.138m$	$0.135m$	$0.153m$
Standard deviation	$\pm 0.095m$	$\pm 0.083m$	$\pm 0.097m$

TABLE I: Map size and localization accuracy comparison. The map sizes are calculated from the original urban map data corresponding to Fig. 6a. The accuracy values of [1] is taken from their original publication, while [2]'s accuracy values are evaluated from our re-implementation.

A. Quantitative Evaluation

We evaluate our method on two data sequences recorded within the highway map and the urban map respectively. The highway sequence is 10km long and consists of 10 minutes of data. The urban sequence is 9km long, and consists of 20 minutes of data. The urban sequence includes a considerable number of turns, traffic light waitings, and dynamic objects. We use the NovAtel trajectories the ground truth in the evaluations. Fig. 7 shows the quantitative evaluation results. In the figures, x and y refer to the translation error in the longitudinal and lateral directions of the vehicle, while z is the translation error in global height. In most cases, our translation errors are within 20cm in both longitudinal and lateral directions, and our rotation errors are within 2°. The localizer shows larger average errors and jittering in the urban sequence than in the highway sequence. We suspect this is caused by the more complex environments in the urban sequence, which results in larger noises for the matches found for optimization.

We also compare our method with NDT[2], an existing



(a) The highway sequence.

(b) The urban sequence.

(c) Translation errors on highway.

(d) Rotation errors on highway.

(e) Translation errors on urban.

(f) Rotation errors on urban.

(g) Proposed v.s. NDT

(h) Proposed v.s. NDT

(i) Proposed v.s. NDT

(j) Proposed v.s. NDT

Fig. 7: Localization accuracy evaluation on the highway sequence and the urban sequence. We compare our trajectories to NDT[2], and use the RTK-enabled trajectories from the NovAtel INS module as ground truth. For the majority of cases, our translation errors are within 20cm, and our rotation errors are within 2°. Overall the proposed method achieves comparable accuracy with NDT that uses a separate point cloud layer stored as 3D normal distance transforms.

LiDAR-based method that uses a separate point cloud (normalized distance transforms) layer. We have reimplemented [2] and have incorporated the IMU information using our optimization framework in Section III-D. Our accuracies are slightly worse than but are comparable to that of NDT[2], whose results are also shown in Fig. 7. Table I also provides a comparison on the map size versus accuracy over three LiDAR-based methods, including a Levinson et al.[1], NDT, and ours. The map size is calculated from the original map data used to create the urban vector map in Fig. 6a. We use a $10 \times 10cm^2$ grid and $10 \times 10 \times 10cm^3$ voxel for [1] and NDT respectively. Note that we have not implemented Levinson et al.[1] but merely compute a map size estimate corresponding to its map representation. The accuracies are summarized from the reported values in their original publication. In summary, the proposed approach is able to achieve a comparable accuracy with traditional layer-based methods, albeit using a much smaller vector map.

B. Ablation Study

We perform ablation studies to access the contributions from different components in our system. The modules to disable include the pole matches, the lane line matches, the road plane matches, and the local map matches. The performances on the highway sequence and the urban sequence are evaluated separately.

Pole matches	Lane line matches	Local map matches	Road plane matches	Behaviors	Drift location
×	✓	✓	✓	no drift	-
✓	×	✓	✓	no drift	-
✓	✓	×	✓	no drift	-
✓	×	×	✓	no drift	-
×	✓	×	✓	drift quickly	1%
×	×	✓	✓	drift slowly in xy	20%
✓	×	✓	×	drift slowly in height	30%

TABLE II: Ablation study on the urban sequence.

Pole matches	Lane line matches	Local map matches	Road plane matches	Behaviors	Drift location
×	✓	✓	✓	longitudinal drift	10%
✓	×	✓	✓	no drift	-
✓	✓	×	✓	no drift	-
✓	×	×	✓	drift quickly	15%
×	✓	×	✓	drift quickly	starting
×	×	✓	✓	drift quickly	starting

TABLE III: Ablation study on the highway sequence.

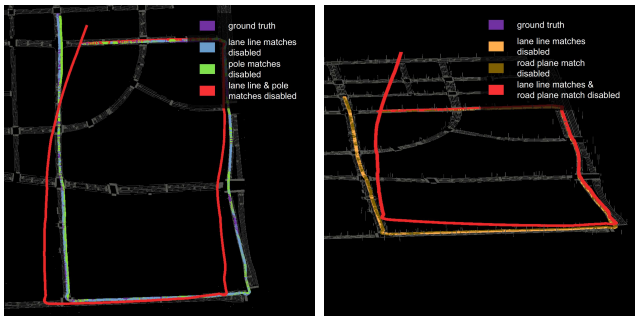
Table II summarizes the ablation study outcomes of the urban sequence. We found that when only one of the components is disabled, the localizer can still function robustly. However, problems occur when two or more modules are disabled. When lane lines and poles are both disabled, the system slowly drifted on the xy-plane, as depicted by the red trajectory in Fig. 8a. The drift is expected since there is no map to compare against, so that the localizer has degenerated to a LiDAR-intertial odometry. In another case, when both lane lines and road planes are disabled, the system slowly drifted in height, as depicted by the red trajectory in Fig. 8b. The drift in height finally leads into a situation that we can no longer find pole matches, which subsequently results in drift on the xy-plane.

Table III summarizes the ablation study outcomes of the highway sequence. The system is able to produce good trajectories when only the lane lines or only the local map are disabled. However, when only the poles are disabled, the system demonstrates slow longitudinal drift. The amount of drift accumulates as time goes by. Fig. 9a visualizes two longitudinal drifts at two respective locations during the course. The visualization compares the arrows' positions in the vector map to those observed from the local map. At the earlier location, the local map lags behind by one-fourth of the arrow, while at the later localization, the local map lags behind by a full arrow. When only the poles are enabled, the system fails at 10% during the course. It is worth noting that this pole-only setting fails in the highway sequence but succeeds in the urban sequence. The reason could be that poles in highways are much sparser than those in urban scenes, which leads to weaker spatial constraints in the highway scene. However, when the local map is turned on for the pole-only setting, the localizer performs robustly. This showcases the contribution of the local map maintenance module.

We also perform a simulated experiment to visualize the local map's contribution more intuitively. We deliberately remove a road segment from the vector map and test whether the system can reliably navigate over the removed segment with or without the local map. At the same time, the pole matches are deliberately turned off as well to avoid generating potential matches outside the radius of the removed road segment. Fig. 10 illustrates the experiment results. The removed segment is 100m long. The red trajectory represents the result without the local map, which shows visible lateral drift. Luckily, the vehicle is able to get back into the correct lane after turning right at the crossroad since the turn converts the lateral drift into a longitudinal drift. On the other hand, with the local map enabled, the green trajectory overlays correctly with the ground truth.

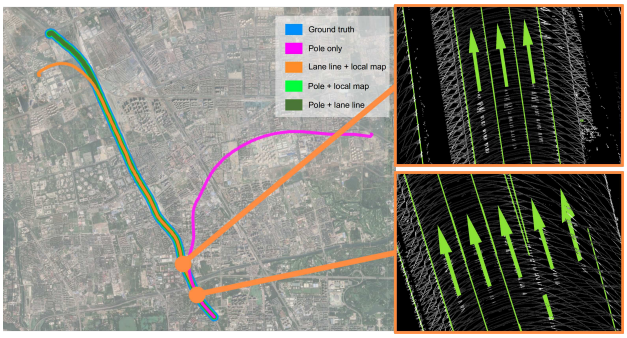
C. Runtime

The system's runtime consists of four major types of operations, namely, semantic extraction, vector map matching, local map matching, and pose optimization. Fig. 11 shows the time cost composition through the course of a sequence. The whole system runs in real-time on a PC with an 8-Core i7 CPU, a 1080Ti GPU, and 32GB RAM. Here, real-time means 10Hz for the back-end optimization thread and 100Hz for the front-end prediction thread. Note that the semantic extraction module consumes the majority of the running time. Since semantic extraction is uploaded to GPU, it can be pipelined with the rest of the tasks running on CPU. Therefore, the system's frame rate is determined by lower one between the two devices: $\text{FPS}_{\text{SYSTEM}} = \min(\text{FPS}_{\text{CPU\_TASK}}, \text{FPS}_{\text{GPU\_TASK}})$. The price of pipelining is a one-frame delay for the freshest optimization result. The state predictor in Section III-E is explicitly designed to cope with this situation. The delay only affects the predictor's base state, which is by design, not the

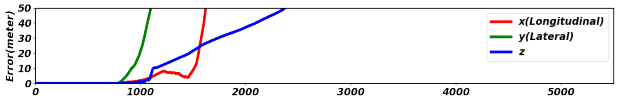


(a) (b)

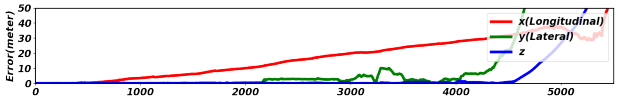
Fig. 8: Ablation study on the urban sequence. Disabling only one component does not result in drift. (a) Disabling both lane lines and poles results in drift on the xy-plane. (b) Disabling both lane lines and road planes result in height drift, which subsequently leads to xy-drift because poles can no longer be matched.



(a) Trajectories from different ablation settings.



(b) Error curves for the pole-only setting.



(c) Error curves for the lane line plus local map setting.

Fig. 9: Ablation study on highway sequence. (a) Activating poles and local map works robustly. Activiting poles and lane lines also work robustly. (b) Unlike the urban scenario, using poles alone without the local map in highway result in drift. (c) Using both lane lines and local map result in slow longitudinal drift.

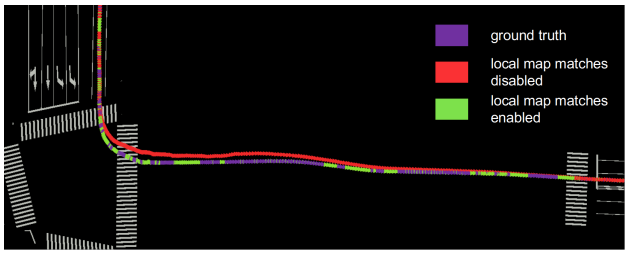


Fig. 10: Accessment of the local map's contribution. A 100m road segment is deliberately removed from the vector map to simulate the situation of insufficient vector matches. The trajectory shows no visible drift when the local map is enabled but a visible lateral drift when the local map is disabled.

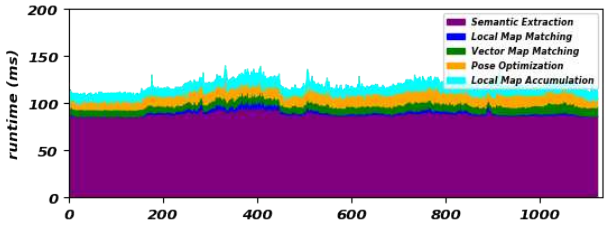


Fig. 11: The localizer's runtime decomposition.

freshest, while the real-time state is computed by integrating the IMU measurements from the base state to the now state. The time cost of the optimization corresponds to a sliding window size $N = 2$. We found no obvious performance gain using a larger sliding window size in our experiments.

V. Conclusion

We have presented a LiDAR-based localization solution on an HD vector map without using a separate localization layer. Experiments show that the proposed method is able to achieve the same level of accuracy compared to existing separate layer-based approaches while requiring orders of magnitude less map storage. In future work, we need to better exploit dashed segments for longitudinal positioning so that the system can deal more robustly with highway scenarios even without pole-like objects.

References

[1] J. Levinson, M. Montemerlo, and S. Thrun, "Map-based precision vehicle localization in urban environments.," in *Robotics: science and systems*, vol. 4, p. 1, Citeseer, 2007.

[2] M. Magnusson, A. Lilienthal, and T. Duckett, "Scan registration for autonomous mining vehicles using 3d-ndt," *Journal of Field Robotics*, vol. 24, no. 10, pp. 803–827, 2007.

[3] A. Milioto, I. Vizzo, J. Behley, and C. Stachniss, "Rangenet++: Fast and accurate lidar semantic segmentation," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4213–4220, IEEE, 2019.

[4] J. Levinson and S. Thrun, "Robust vehicle localization in urban environments using probabilistic maps," in *2010 IEEE International Conference on Robotics and Automation*, pp. 4372–4378, IEEE, 2010.

[5] R. W. Wolcott and R. M. Eustice, "Fast lidar localization using multiresolution gaussian mixture maps," in *2015 IEEE international conference on robotics and automation (ICRA)*, pp. 2814–2821, IEEE, 2015.

[6] G. Wan, X. Yang, R. Cai, H. Li, Y. Zhou, H. Wang, and S. Song, "Robust and precise vehicle localization based on multi-sensor fusion in diverse city scenes," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4670–4677, IEEE, 2018.

[7] J. Zhang and S. Singh, "Loam: Lidar odometry and mapping in real-time.," in *Robotics: Science and Systems*, vol. 2, 2014.

[8] H. Ye, Y. Chen, and M. Liu, "Tightly coupled 3d lidar inertial odometry and mapping," in *2019 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2019.

[9] P. J. Besl and N. D. McKay, "Method for registration of 3-d shapes," in *Sensor fusion IV: control paradigms and data structures*, vol. 1611, pp. 586–606, International Society for Optics and Photonics, 1992.

[10] J. Behley and C. Stachniss, "Efficient surfel-based slam using 3d laser range data in urban environments.," in *Robotics: Science and Systems*, vol. 2018, 2018.

[11] X. Chen, A. Milioto, E. Palazzolo, P. Giguere, J. Behley, and C. Stachniss, "Suma++: Efficient lidar-based semantic slam," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4530–4537, IEEE, 2019.

[12] M. Schreiber, C. Knöppel, and U. Franke, "Laneloc: Lane marking based localization using highly accurate maps," in *2013 IEEE Intelligent Vehicles Symposium (IV)*, pp. 449–454, IEEE, 2013.

[13] F. Poggenhans, N. O. Salscheider, and C. Stiller, "Precise localization in high-definition road maps for urban regions," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2167–2174, IEEE, 2018.

[14] J. Jeong, Y. Cho, and A. Kim, "Hdmi-loc: Exploiting high definition map image for precise localization via bitwise particle filter," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6310–6317, 2020.

[15] A. Ranganathan, D. Ilstrup, and T. Wu, "Light-weight localization for vehicles using road markings," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 921–927, IEEE, 2013.

[16] Y. Lu, J. Huang, Y.-T. Chen, and B. Heisele, "Monocular localization in urban environments using road markings," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, pp. 468–474, IEEE, 2017.

[17] Z. Xiao, D. Yang, T. Wen, K. Jiang, and R. Yan, "Monocular localization with vector hd map (mlvhm): A low-cost method for commercial ivs," *Sensors*, vol. 20, no. 7, p. 1870, 2020.

[18] J. Biswas and M. M. Veloso, "Localization and navigation of the cobots over long-term deployments," *The International Journal of Robotics Research*, vol. 32, no. 14, pp. 1679–1694, 2013.

[19] W.-C. Ma, I. Tartavull, I. A. Bârsan, S. Wang, M. Bai, G. Mattyus, N. Homayounfar, S. K. Lakshmikanth, A. Pokrovsky, and R. Urtasun, "Exploiting sparse semantic hd maps for self-driving vehicle localization," *arXiv preprint arXiv:1908.03274*, 2019.

[20] E. Javanmardi, Y. Gu, M. Javanmardi, and S. Kamijo, "Autonomous vehicle self-localization based on abstract map and multi-channel lidar in urban area," *IATSS research*, vol. 43, no. 1, pp. 1–13, 2019.

[21] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, C. Stachniss, and J. Gall, "Semantickitti: A dataset for semantic scene understanding of lidar sequences," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9297–9307, 2019.

[22] Y. Yu, J. Li, H. Guan, C. Wang, and J. Yu, "Semiautomated extraction of street light poles from mobile lidar point-clouds," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 53, no. 3, pp. 1374–1386, 2014.

[23] Y. Yu, J. Li, H. Guan, F. Jia, and C. Wang, "Learning hierarchical features for automated extraction of road markings from 3-d mobile lidar point clouds," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 8, no. 2, pp. 709–726, 2014.

[24] S. Agarwal, K. Mierle, and Others, "Ceres solver." `http://ceres-solver.org`.

[25] T. Qin, P. Li, and S. Shen, "Vins-mono: A robust and versatile monocular visual-inertial state estimator," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 1004–1020, 2018.