HyperMap: Compressed 3D Map for Monocular Camera Registration

Ming-Fang Chang[1], Joshua Mangelson[2], Michael Kaess[1], and Simon Lucey[1,3]

***Abstract*— We address the problem of image registration to a compressed 3D map. While this is most often performed by comparing LiDAR scans to the point cloud based map, it depends on an expensive LiDAR sensor at run time and the large point cloud based map creates overhead in data storage and transmission. Recently, efforts have been underway to replace the expensive LiDAR sensor with cheaper cameras and perform 2D-3D localization. In contrast to the previous work that learns relative pose by comparing projected depth and camera images, we propose HyperMap, a paradigm shift from online depth map feature extraction to offline 3D map feature computation for the 2D-3D camera registration task through end-to-end training. In the proposed pipeline, we first perform offline 3D sparse convolution to extract and compress the voxelwise hypercolumn features for the whole map. Then at run-time, we project and decode the compressed map features to the rough initial camera pose to form a virtual feature image. A Convolutional Neural Network (CNN) is then used to predict the relative pose between the camera image and the virtual feature image. In addition, we propose an efficient occlusion handling layer, specifically designed for large point clouds, to remove occluded points in projection. Our experiments on synthetic and real datasets show that, by moving the feature computation load offline and compressing, we reduced map size by $87-94\%$ while maintaining comparable or better accuracy.**

***Index Terms*— autonomous driving, map, 2D-3D registration**

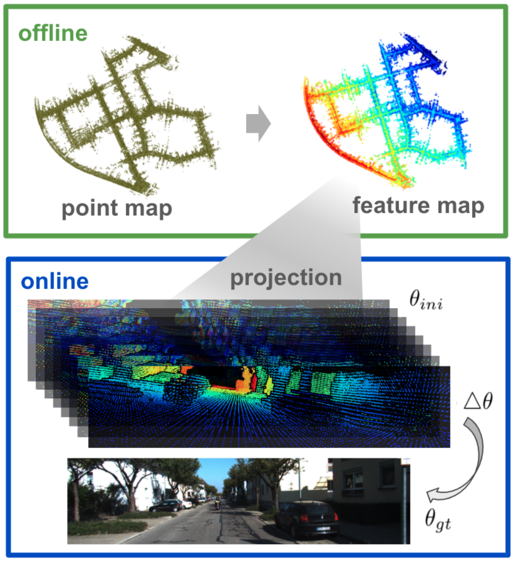


Fig. 1: Our network predicts the relative camera pose by comparing the camera image to a high-dimensional feature image created by projecting our feature map to 2D using the noisy initial pose.

I. Introduction

State-of-the-art autonomous driving systems rely on High-Definition (HD) maps for localization. By comparing online sensor measurements with the 3D information stored in the HD map (usually in the form of a dense point cloud with additional labels), autonomous vehicles are able to refine the ego pose estimation and also correct the drift caused by accumulated error. The nature of the offline map building process makes additional pre-processing and human-assisted annotations possible for HD maps, greatly increasing their utility at run-time. However, current HD map formats are not optimized for the specific task they will be used for, such as localization, leading to wasteful storage overhead.

In this paper, we propose a novel strategy to learn on-map convolutional features to compress the map and preserve the registration performance. Given a noisy initial camera pose, our method predicts the relative 6 degree-of-freedom (DoF) pose of the camera by comparing the image captured by the camera to a virtual feature image created by projecting a 3D feature map to 2D. Although the methods that perform localization by comparing LiDAR scans collected in real-time to the HD map usually outperform the camera-based methods in terms of localization error [1], LiDAR sensors are substantially more expensive than cameras. Our solution leverages the robustness of LiDAR sensors in the offline map building process and then relies solely on cameras to perform online pose registration. This design significantly reduces the online system cost while still leveraging the strengths of LiDAR sensors. In this work, we assume approximate pose is known, which is reasonable since GPS is very common in modern devices and the existing global localization or vehicle pose estimation methods can be used as our input.

The registration of 2D camera images to a 3D point-cloud map is non-trivial due to the inherent difference in the modalities. Prior works have tried to solve the problem by projecting depth information from the 3D map into 2D to form a depth image from which features are then extracted to enable comparison with the observed camera image [2]–[4]. We refer to the decision to perform projection before feature extraction as "*early projection*" in this paper. We propose instead the use of "*late projection*", a method which precomputes and compresses the 3D features on the voxelized point cloud map and then performs projection for subsequent alignment with the captured 2D RGB image. Our approach utilizes sparse 3D convolutional layers to

[1] Ming-Fang Chang, Michael Kaess, Simon Lucey are with the Carnegie Mellon University {mingfanc, kaess, slucey}@andrew.cmu.edu

[2] Joshua Mangelson is with the Brigham Young University joshua_mangelson@byu.edu

[3] Simon Lucey is also part of the Australian Institute of Machine Learning (AIML) at the The University of Adelaide

extract features from the HD point cloud map [5]–[8]. The sparse convolutional layers enable us to process large point clouds efficiently. Compared with current state-of-the-art methods [2], [3], which uses "*early projection*", our method compresses the learned features and reduces the required map voxel resolution significantly, and thus reduces $87-94\%$ of map size with comparable performance.

The primary contributions of this paper is as follows: we propose "*late projection*" in contrast to "*early projection*" for the 2D-3D registration task. Our late projection strategy precomputes and compresses the 3D map features offline before online projection, which we refer to as a "HyperMap" due to the use of hypercolumn features [9].. The proposed HyperMap outperforms the baseline in map size significantly while maintaining comparable or slightly better performance. Although we focus 2D-3D registration in this paper, we believe that the concept of "*late projection*" can be extended to and potentially benefit other map-related tasks.

In addition, we propose an efficient occlusion-handling layer that enables backpropagation from a projected feature image to 3D convolutional layers on a large-scale sparse point cloud map. This occlusion-handling layer is crucial to the scalability of our proposed HyperMap. We will release the 2D to 3D registration datasets used in the experiments, including from both synthetic and real-world modern autonomous driving camera and LiDAR data. Our dataset contains challenging scenarios such as weather changes, sensor noise and crowded dynamic scenes.

II. Related Works

Previous works on registration/localization range from traditional descriptor matching to deep networks. Due to the space limitation, we focus on the local registration methods in this section. In contrast to local registration, there are global registration methods that do not require an initial pose but are usually less accurate [10]–[14].

A. Local Registration (Local Localization)

When the map and an rough estimate of the camera pose are available, previous works have attempted to refine the pose using camera registration to 3D map. The prior knowledge of camera pose might be from the GPS measurements, global localization, or pose estimation results in previous time frames. Local registration, or local localization, approaches compare online sensor observations with information from the map to refine the initial pose estimate.

Leveraging classical visual odometry methods, Caselitz at el. [15] used Structure-from-Motion (SfM) to reconstruct sparse point clouds from video sequences and then performed ICP to register the SfM point cloud to the LiDAR map, which requires robust feature points and video sequences, not a single image. Kim et al. proposed to register a stereo camera to a LiDAR map using the image feature correspondences in stereo depth and projective LiDAR depth [16]. Mastin et al. proposed to use mutual information to register an aerial image to LiDAR images [17].

In autonomous driving applications, the ground plane and road markings can be especially useful. Lu et al. [18] used the chamfer distance to align the detected road markings to a sparse 3D map. Wolcott and Eustice [19] proposed to use reflectance information derived from the LiDAR ground map, containing mostly road marking information, to solve the local registration problem. Their method generated synthetic projective reflectance images and refined the initial pose by maximizing the mutual information score to align the synthetic reflectance images with a monocular camera onboard the vehicle. While the ground plane provides a distinctive set of features for alignment, methods that depend on it fail when large portions of the road are occluded or differ from the pre-built map, such as in the presence of snow or after construction [20]. This dependence on the ground-plane can be overcome by taking into account the 3D-volumetric information. In [20], Wolcott and Eustice proposed the use of Gaussian-mixture-models (GMMs) to summarize map height and reflectivity for efficient LiDAR-based localization. However, this method also depends on having a LiDAR sensor on-board the vehicle at localization time. Yu et al. used 2D-3D line correspondences for registration, which only works when line features exist [21].

Recently, CMRNet [3] and CMRNet++ [2] leveraged a CNN to solve the local 2D-3D registration problem in a way that both takes into account 3D-structure information and only requires a monocular camera at localization time. It adopted a correlation filter, which is often used in the optical flow networks [22], to regress the relative 6-DoF pose between a virtual LiDAR depth image and an RGB camera image. The experiments showed that CMRNet reached centimeter-level translation error in an unseen environment. EnforceNet [4] also used a CNN to regress pose between projected depth and RGB images. To the extent of authors' knowledge, CMRNet is the only existing method that, like our HyperMap, is not trained in testing environment, making it more generalizable to maps other than the ones on which it was trained. Thus we pick CMRNet as our baseline. Both CMRNet and EnforceNet perform feature computation after projection, we refer to this design as "*early projection*".

B. Image Compression

Clustering techniques have been used in image compression for decades. In [23], Oehler and Gray proposed to use Vector Quantization (VQ) to compress and classify medical images. Agustsson et al. [24] proposed a learned VQ to compress and decode the latent representation in an auto-encoder structure. Recently, Wei et al. applied task-orientation compression to form a 2D binary map [25].

Instead of the general 2D binary code, we perform clustering on the learned map features and only store the per-voxel centroid index as the map feature. This is more compact than the binary code used in [25]. Also, the 2D to 3D registration problem we solve is more complicated and challenging than the 2D to 2D registration setting in [25] since it requires backpropagation through projection.

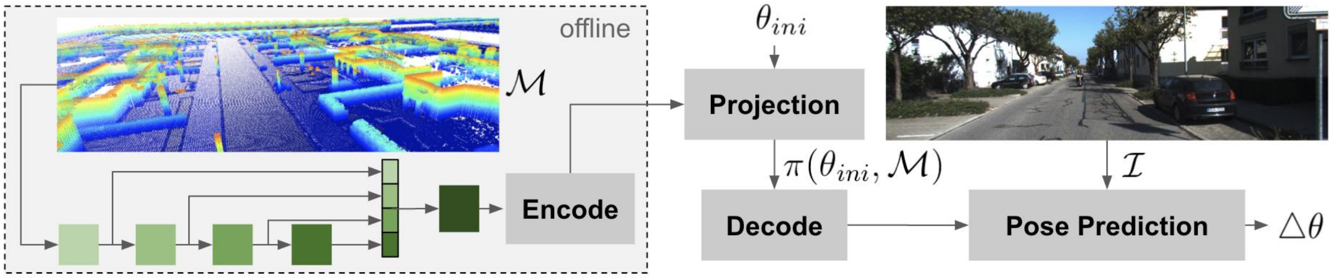


Fig. 2: System architecture. The sparse convolutional layers are shown by green boxes. We only store the compressed map.

C. Occlusion Handling in 3D Shape Learning

Occlusion handling is an important module in the differential renderers used in recent 3D shape learning methods to project 3D information to 2D. Lin et al. proposed using upsampling and max-pooling process to build a pseudo-renderer [26]. In [27] and [28], the authors used a differentiable ray tracing method with probabilistic voxel occupancy for occlusion reasoning. Sitzmann et al. proposed an occlusion-aware projection that first transformed the voxelized feature representation to the canonical view grid and then used a network to predict the per-pixel visibility [29].

In our case, the local map grid range has much larger scale than the object-level voxel grids used by the existing 3D representation learning methods. The existing methods are too expensive and memory consuming for our task. We thus propose an efficient pyramid of max-pooling layers with different kernel sizes to overcome this challenge.

III. Problem Formulation

In this paper, we propose the use of "*late projection*", i.e. the computation of 3D features directly on the map in offline fashion before projecting into the 2D space to generate the projective virtual feature image. By compressing the local statistics into the map features, our method works with much smaller maps than the raw point cloud map. We only need the compressed map to perform the online registration. The system architecture is visualized in Figure 2. The major challenges associated with "*late projection*" can be summarized as the following:

- Although we can reduce the map resolution by representing the local map statistics by features, the high-dimensional features might reversely increase the total map size and takes more time to load and process.
- Due to the sparsity of the LiDAR map, points occluded in RGB images might appear in the projective feature images. Thus, we need a differentiable occlusion handling layer for the large-scale sparse point clouds.

We use 3D feature compression and an occlusion handling layer to overcome the above challenges, as described in Section IV and Section V.

Our problem formulation is as follows. Let θ_{ini} represent the 6-DoF initial camera pose, θ_{gt} represent the ground truth pose, $\mathcal{I}$ represent the camera image and $\mathcal{M}$ represent the 3D voxelized feature map. Assuming the camera intrinsics are fixed, our goal is to estimate the relative pose $\triangle\theta$ that aligns our initial estimate θ_{ini} to θ_{gt}. We can formulate this as an estimation problem where we seek to estimate the weights ω of a network $\mathcal{G}$, that predicts $\triangle\theta$ from θ_{ini}, $\mathcal{I}$, and $\mathcal{M}$. Let $\circ$ be the pose composition operator:

$$\begin{aligned} \triangle\theta &= \mathcal{G}(\mathcal{I}, \pi(\theta_{ini}, \mathcal{M}); \omega) \\ \bar{\theta} &= \theta_{ini} \circ \triangle\theta \end{aligned} \tag{1}$$

where $\pi(\cdot)$ is the 3D to 2D perspective projection function.

IV. Map Feature Extraction

We voxelize the 3D point cloud map in high resolution to extract the 3D convolutional features and downsample. We adopt the 3D sparse convolutional filter [5]–[8] due to its great scability and efficiency. We extract the convolutional features using a set of 3D sparse convolutional layers, which only operate on the occupied voxels and are suitable for sparse point cloud data from sensors like LiDAR.

In order to capture features with different frequencies, we apply the hypercolumn [9] concept to 3D feature extraction. We use stride 2 for the first 3D convolutional block and 1 for all the other blocks. The receptive field of each layer expands as more convolutional layers are applied, and the feature dimension also increases correspondingly. Afterwards, we combine the multiple activations to form a hypercolumn feature vector for each occupied voxel to preserve both the precision of earlier layers and the capacity of later layers. The final voxel resolution was thus reduced by ratio two due to the stride 2 in the first block.

At training time, we first voxelize the whole raw point cloud map, crop the local map region using the initial pose, extract 3D features in the map coordinate frame, and then transform the cropped feature map to the camera coordinate frame. Afterwards, n layers of 3D sparse convolutional filters are applied to the voxelized local map, and the feature output from the n convolutional layers are concatenated to form a high-dimensional hypercolumn feature vector.

For a voxel v_i in the map, the corresponding hypercolumn feature vector is first compressed to a lower dimension feature $f_i \in \mathbb{R}^m$ (dimension 72 → 16) using another 3D sparse convolutional layer. After trained end-to-end, we apply K-means algorithm to all the f_i in the map to obtain k centroids, and compute the cluster index d_i of each voxel (As shown in Figure 3b, we use $k = 16$ in our experiments, so we only need 4 bits to represent the centroid index, $d_i \in 0, 1, 2, ..., 15$). We then project the cluster index d_i to form a 2D virtual feature image, and recover the original feature f_i from d_i using the corresponding K-means centroids. Notice that map feature projection required retrieving the map feature data from the storage and thus projecting d_i is cheaper and faster than projecting f_i due to its small size.

In the map projection step, we project the voxel grids to form a depth map and concatenate the depth map to f_i as an additional channel, so the final projective virtual feature image has $m + 1$-dimensions. This feature precomputation step reduces the required voxel resolution while preventing the performance drop. We use kernel size 3 for all the 3D sparse convolutional layers.

V. Occlusion Handling

The compressed map features are projected and decoded to form a virtual feature image using the camera intrinsics and the given initial pose. However, because of the nature of sparse point clouds, the occluded points may appear in the virtual feature image if not handled. To remove the occluded points, we design a maxpooling pyramid inspired by the point cloud occlusion filtering described in [3]. Our occlusion handling layer is very efficient and is suitable for large-scale point clouds since it only utilizes the max pooling layers.

We use the voxel size to approximate the occupied neighborhood of the map points in 3D space, and projection of the occupied neighborhood should only contain the projections of the map points that are closer to the camera than the voxel center (with smaller depth value). This means that if a projective map point has some nearby pixels with smaller depth, it is likely that this voxel is occluded. We use efficient maxpooling filters to simulate the 2D occupied neighborhood. In order to apply the maxpooling layers, we first make the projective depth negative and set the empty pixels to the maximum negative depth value. The pixels with smaller depth values originally would be larger after this transformation, and thus will be kept after the maxpooling operation. Afterwards, we recover the original image by setting the empty pixels back to zero and inverting the sign of the depth map. The output, maxpooled depth map, is noted as $M_r(\mathbf{p})$ with kernel size r at pixel position $\mathbf{p}$.

Let $D(\mathbf{p})$ and $R(\mathbf{p})$ be the depth map and its corresponding occlusion filter kernel size map, and f be the focal length. The map of the occlusion filter kernel size (in pixel) can be computed from the fixed voxel size in the map:

$$R(\mathbf{p}) = \frac{\text{voxel size} \times f}{D(\mathbf{p})} \tag{2}$$

Afterwards, we find the pyramid level with smallest $M_r(\mathbf{p})$ among all levels for each pixel, denoted as:

$$r_{min} = \arg\min_{r} \ M_r(\mathbf{p}). \tag{3}$$

If r_{min} is larger than the $R(\mathbf{p})$ at this pixel, it means that this pixel is occluded by a nearby pixel with smaller depth value and the corresponding feature value should be set to zero. Let $F(\mathbf{p})$ be the virtual feature image. The final virtual feature image is computed by:

$$F_{final}(\mathbf{p}) = \begin{cases} 0, & \text{if } \arg\min_r \ M_r(\mathbf{p}) - R(\mathbf{p}) > \delta \\ F(\mathbf{p}), & \text{otherwise} \end{cases} \tag{4}$$

We choose $\delta = 0.5$ so the occlusion filter is only effective when the occluded points are far away from the visible point. If several layers in $M_r(\mathbf{p})$ have the same pixel value, which happens when all the maxpooling layer outputs are dominated by a close-by nearer point, we pick the smallest r among them. Results are shown in Figure 3a.

VI. Camera Pose Prediction

Given the virtual feature image and the RGB camera image, we regress the relative camera pose $\triangle\theta$ using a CNN following [3]. We use the image feature extraction branch of PWCNet [22] for RGB feature extraction and simply replace the dimension of the first convolutional block in the depth feature extraction branch with our projective map feature dimension. A correlation filter is then used to match the features from the RGB image and the virtual map feature image. Several fully connected layers are used to predict the translation in xyz directions and the quaternion for rotation to represent 6-DoF camera pose. We add one additional tanh layer as an output layer to constrain the range of predicted translation and rotation. For training, we use Smooth L1 loss and quaternion angular distance loss as proposed in [3].

VII. Experiments

In this section, we describe the evaluations on CARLA synthetic dataset [30], KITTI Odometry dataset [31], and Argoverse Tracking dataset [32]. We choose CMRNet [3] as our baseline and compare with it in $0.1m$, $0.2m$ and $0.4m$ voxel resolutions. We use $n = 4$, $m = 16$, $k = 16$, and a five-level occlusion pyramid (maxpooling kernel $r = 3, 5, 11, 15, 23$) in all experiments (the details of the parameters are in Section IV).

A. Data Preparation

For the CARLA dataset, we used the official data collector to collect single camera sequences with ground truth poses. We collected seven sequences for the training set (14755 frames) and two sequences for the validation (4228 frames). The validation set contains weather conditions that do not exist in training set, shown in Figure 5. The point cloud map (Town01) was downloaded from the official repository.

As for the KITTI Odometry dataset, we used the LiDAR maps, the ground truth poses and the initial poses for validation set provided by the authors of CMRNet [3] We used the sequences 03, 04, 05, 07, 08 and 09 in the KITTI Odometry dataset as the training set (10581 frames) and the randomly downsampled sequence 00 as the validation set (1500 frames). We excluded sequence 06 due to the artifacts in the generated map. The validation map does not overlap with training maps except for only 200 frames. We used SLAM poses from [3] as the ground truth since the KITTI

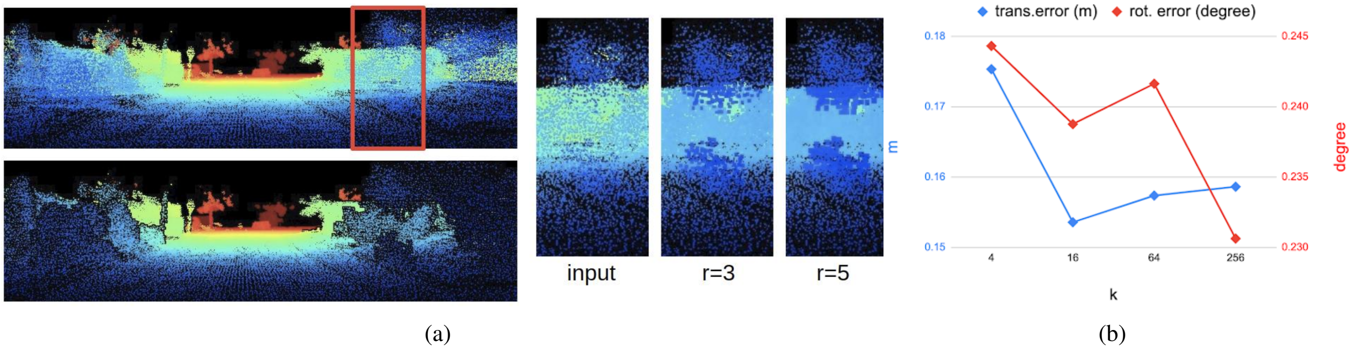


Fig. 3: (a) Occlusion handling. We use a max pooling pyramid to implement an occlusion handling layer, described in Section V. The above figure shows the effect of applying a maxpooling kernel with several different sizes, where the occluded pixels are set to zeros (shown in black), and the input and output of the occlusion handling layer. Larger depth values are shown in red and small depth values are shown in dark blue. (b) Accuracy plot of using different number of centroids in K-means with CARLA dataset. We observed no obvious benefit of using more than 16 centroids.

ground truth poses are noisy. As mentioned in [3], the ground truth poses in KITTI dataset caused map inconsistency in loop closures, so we used the ground truth poses provided by CMRNet authors as well, which was optimized by loop-closure SLAM method as described in [3], [15].

We built the Argoverse maps by accumulating the LiDAR scans using the provided ground truth poses. We uniformly downsampled the original train and validation splits as the training set (9328 frames from 85 logs), and also downsampled the original test split as the test set (1599 frames from 24 logs). We removed log 3373 and 7d37 from the training set because the ego vehicle was surrounded by large buses.

We downsampled the maps using voxel resolutions $0.1m$, $0.2m$, and $0.4m$ for the baseline experiments, and used the $0.2m$ resolution as the input of our HyperMap. The final HyperMap resolution is $0.4m$. To simulate erroneous initial pose, we added translation noise within [-2m, +2m] in xyz directions, and rotation noise of [-10°, +10°] about xyz axes applied in xyz order following [3]. The initial poses were generated online in training time and fixed in test time.

B. Implementation Details

Aiming for a fair comparison, we integrated the CMRNet into our pipeline, so that the only difference in the experiments was the network itself. We added a scaled tanh layer to the CMRNet implementation at output to leveraging the prior knowledge of the known noise range, as we did in our HyperMap. We split the training process into two stages. First, we cropped the local map around initial camera pose with radius 50m and voxelized it and then applied the 3D sparse convolutional layer to the local voxelized map to extract map features. Afterwards, the extracted map feature was projected to form a virtual feature image for pose prediction and initial training. Second, after the map feature is well-trained, we applied the pretrained sparse convolutional layers to the whole voxelized map to get the map features f_i, using K-means to get the centroid index d_i for each voxel, and only store the d_i in the map for the map size comparison. Afterwards, we fixed the map features, only refining the pose prediction network until convergence. The refinement step helped to compensate the compression error induced by K-means. Given the scale of our maps and the efficiency of the sparse convolutional networks, we were able to process the whole map offline on our lab server without splitting it into submaps for the experiments. This approach is scalable to larger maps with a divide-and-conquer approach since the convolutional filters are translationally invariant and the receptive fields are limited.

We implemented all the models in PyTorch. All the models are trained and timed on an Intel(R) Xeon(R) CPU E5-2660 v4 @ 2.00GHz machine with GeForce GTX TITAN Xp GPU. We train all the models using learning rate 10^{-4} and batch size 40 with Adam optimizer. The occlusion handling layer takes about $1ms$ and the pose prediction takes about $14ms$ on our machine for KITTI odometry dataset.

C. Performance

We observed that our HyperMap has comparable or better accuracy in both the synthetic and real-world datasets, especially in translation, and much smaller map size than the baseline. Our method, as shown in Table I, outperforms the $0.4m$ baseline significantly and even outperforms the $0.1m$ baseline in the Argoverse Tracking dataset where the baseline map size is more than seven times larger. We adopt a sparse representation to store the map. The map size is computed as the total storage required to store all the 3-dimensional indice (2-byte integer coordinates) of the occupied voxels (map points) and the corresponding voxel features (4-bit for the 16 K-means centroid indices). The baseline map only contains the voxel indices and no feature. Although increasing the voxel size reduces the map size effectively, the corresponding baseline performance drops, as shown in Table I. We used $bytes/m^2$ as the map size unit since the maps in our scenario are usually very flat. Our method compressed the local statistics into voxel features, generated feature maps with voxel size $0.4m$, and reached comparable performance with the baselines using smaller voxel sizes, at the cost of small storage overhead for storing the features.

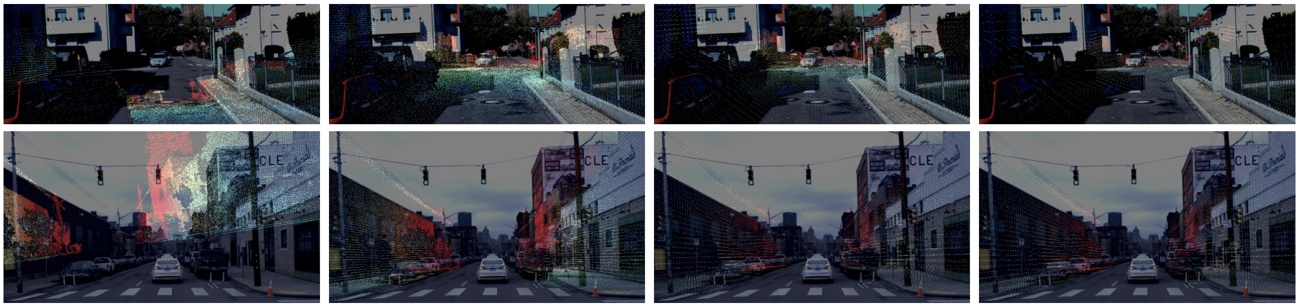
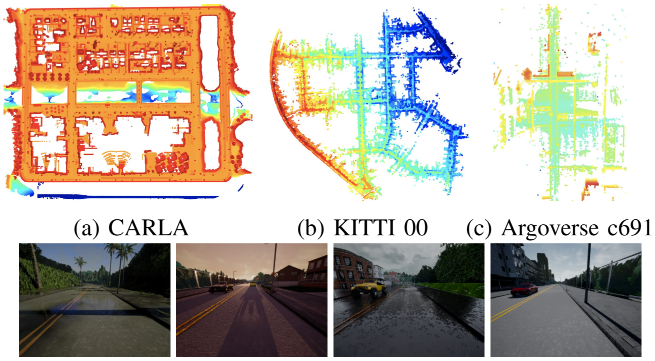


Fig. 4: Visualizations of the projected depth maps from KITTI Odometry dataset (upper row) and Argoverse Tracking dataset (lower row) (further distance is represented by red, closer distance is represented by blue) From left to right are initial pose, baseline with $0.1m$ voxel size, baseline with $0.2m$ voxel size and our HyperMap result. Best viewed in PDF file.



(a) CARLA (b) KITTI 00 (c) Argoverse c691

(d) after rain (e) sunset (f) rain (g) sunny

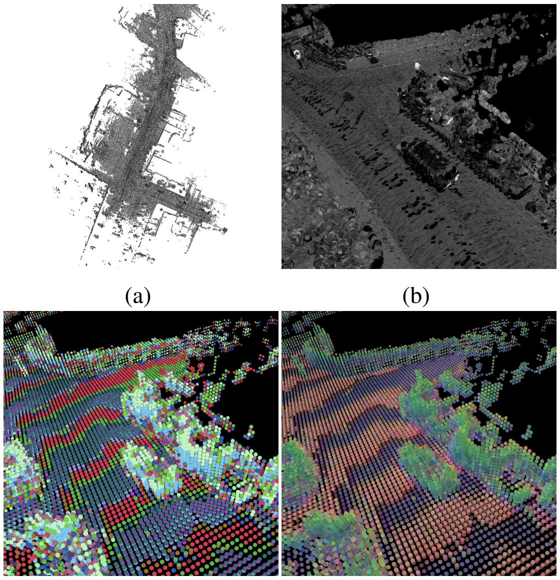
Fig. 5: Visualization of the maps and the CARLA weathers.

DATASET	MODEL	VOXEL SIZE(M)	TRANS(M)	ROT(°)	MAP SIZE ($bytes/m^2$)
CARLA	CMRNet + tanh	0.1	0.16	0.30	524.2
	CMRNet + tanh	0.2	0.18	0.29	193.5
	CMRNet + tanh	**0.4**	0.24	**0.24**	**52.5**
	HyperMap	**0.4**	**0.15**	**0.24**	56.7
KITTI	CMRNet + tanh	0.1	**0.45**	**1.35**	1471.2
	CMRNet + tanh	0.2	0.47	1.40	330.3
	CMRNet + tanh	**0.4**	0.63	1.97	**75.7**
	HyperMap	**0.4**	0.48	1.42	81.8
Argoverse	CMRNet + tanh	0.1	0.60	1.35	717.1
	CMRNet + tanh	0.2	0.61	0.95	282.7
	CMRNet + tanh	**0.4**	0.65	0.95	**89.0**
	HyperMap	**0.4**	**0.58**	**0.93**	96.0

TABLE I: Quantitative comparisons.Overall, we reduced the total map size by $87 - 94\%$ with comparable accuracy.

VIII. Discussion and Conclusion

In this work, we demonstrated the valuable potential of the proposed offline map feature preprocessing. It is possible to apply additional compression methods on top of the raw formats to further compress the maps. The map compression advantages from our method would still be valid in this case. We have so far demonstrated that the proposed approach is effective with voxel-based downsampling. It would be interesting to explore different point cloud downsampling methods, such as uniform downsampling or selective downsampling in the future. In addition, although not the focus of this paper, it is possible to further the performance of HyperMap using an iterative approach [2], [3]. In addition to the methods used in this paper, other types of features, such as semantic labels, loop-closure features, and other 3D representations (like PointNet [33], FCGF [7]), can be added to the offline process and potentially improve performance. The advantage of the lowered voxel resolution is obvious in reducing map storage and the online query and processing time. Furthermore, the benefits enabled by the "*late projection*" paradigm opens the door to many possibilities in algorithm and system design, such as iterative optimization, high-speed localization, and cheaper on-board computers. We look forward to investigating other applications and methods that take advantage of the "*late projection*" way of thinking.



(a) (b)

(c) (d)

Fig. 6: Visualizations of (a)(b) the raw point cloud map, (c) the K-means centroids stored on the map, and (d) the reconstructed map features from the K-means centroids and the stored indices in Hypermap. Notice that the LiDAR sweep pattern from the point cloud map generation process is also captured by the feature extractor.

Acknowledgements

This work was supported by the CMU Argo AI Center for Autonomous Vehicle Research. We thank the authors of CMRNet [3] for providing the maps and the ground truth poses of the KITTI Odometry Dataset. We also thank our labmates for the valuable suggestions to improve this paper.

References

[1] S. Kuutti, S. Fallah, K. Katsaros, M. Dianati, F. Mccullough, and A. Mouzakitis, “A survey of the state-of-the-art localization techniques and their potentials for autonomous vehicle applications,” *IEEE Internet of Things Journal (IoT-J)*, vol. 5, no. 2, pp. 829–846, Mar. 2018.

[2] D. Cattaneo, D. G. Sorrenti, and A. Valada, “CMRNet++: Map and camera agnostic monocular visual localization in lidar maps,” in *Proc. IEEE Intl. Conf. on Robotics and Automation (ICRA)*, May 2020.

[3] D. Cattaneo, M. Vaghi, A. L. Ballardini, S. Fontana, D. G. Sorrenti, and W. Burgard, “CMRNet: Camera to lidar-map registration,” in *IEEE Intelligent Transportation Systems Conference (ITSC)*, Oct. 2019, pp. 1283–1289.

[4] Y. Chen and G. Wang, “EnforceNet: Monocular camera localization in large scale indoor sparse lidar point cloud,” arXiv, Tech. Rep., Jul. 2019.

[5] D. Retinskiy, “Submanifold sparse convolutional networks,” Tech. Rep., Jun. 2017.

[6] Y. Zhou and O. Tuzel, “VoxelNet: End-to-end learning for point cloud based 3d object detection,” in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2018, pp. 4490–4499.

[7] C. Choy, J. Park, and V. Koltun, “Fully convolutional geometric features,” in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2019.

[8] C. Choy, W. Dong, and V. Koltun, “Deep global registration,” in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2020, pp. 2511–2520.

[9] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik, “Hypercolumns for object segmentation and fine-grained localization,” in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2015, pp. 447–456.

[10] T. Sattler, B. Leibe, and L. Kobbelt, “Fast image-based localization using direct 2d-to-3d matching,” in *Proc. Intl. Conf. on Computer Vision (ICCV)*, Nov. 2011.

[11] A. Kendall, M. Grimes, and R. Cipolla, “PoseNet: A convolutional network for real-time 6-dof camera relocalization,” in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2015.

[12] L. Svarm, O. Enqvist, F. Kahl, and M. Oskarsson, “City-scale localization for cameras with known vertical direction,” *IEEE Trans. on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 39, no. 7, Jul. 2017.

[13] A. Gawel, C. D. Don, R. Siegwart, J. Nieto, and C. Cadena, “X-View : Graph-based semantic multi-view localization,” *IEEE Robotics and Automation Letters (RA-L)*, vol. 3, no. 3, Jul. 2018.

[14] D. Cattaneo, M. Vaghi, S. Fontana, A. L. Ballardini, and D. G. Sorrenti, “Global visual localization in lidar-maps through shared 2d-3d embedding space,” in *Proc. IEEE Intl. Conf. on Robotics and Automation (ICRA)*, May 2020.

[15] T. Caselitz, B. Steder, M. Ruhnke, and W. Burgard, “Monocular camera localization in 3d lidar maps,” in *Proc. IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Oct. 2016, pp. 1926–1931.

[16] H. Kim, C. D. Correa, and N. Max, “Automatic registration of lidar and optical imagery using depth map stereo,” in *IEEE Intl. Conf. on Computational Photography (ICCP)*, May 2014.

[17] A. Mastin, J. Kepner, and J. Fisher, “Automatic registration of lidar and optical images of urban scenes,” in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*. Institute of Electrical and Electronics Engineers (IEEE), Jun. 2009.

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

[19] R. W. Wolcott and R. M. Eustice, “Visual localization within lidar maps for automated urban driving,” in *Proc. IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Sep. 2014, pp. 176–183.

[20] ——, “Fast lidar localization using multiresolution Gaussian mixture maps,” in *Proc. IEEE Intl. Conf. on Robotics and Automation (ICRA)*, no. June, Jun. 2015, pp. 2814–2821.

[21] H. Yu, W. Zhen, W. Yang, J. Zhang, and S. Scherer, “Monocular camera localization in prior lidar maps with 2d-3d line correspondences,” in *Proc. IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Oct. 2020.

[22] D. Sun, X. Yang, M. Y. Liu, and J. Kautz, “PWC-Net: CNNs for optical flow using pyramid, warping, and cost volume,” in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2018, pp. 8934–8943.

[23] K. L. Oehler and R. M. Gray, “Combining image compression and classification using vector quantization,” *IEEE Trans. on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 17, no. 5, pp. 461–473, 1995.

[24] E. Agustsson, F. Mentzer, M. Tschannen, L. Cavigelli, R. Timofte, L. Benini, and L. Van Gool, “Soft-to-hard vector quantization for end-to-end learning compressible representations,” in *Proc. Conf. on Neural Information Processing Systems (NeurIPS)*, Dec. 2017, pp. 1142–1152.

[25] X. Wei, I. A. Barsan, S. Wang, J. Martinez, and R. Urtasun, “Learning to localize through compressed binary maps,” in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 10 308–10 316.

[26] C. H. Lin, C. Kong, and S. Lucey, “Learning efficient point cloud generation for dense 3d object reconstruction,” in *Proc. AAAI Conf. on Artificial Intelligence (AAAI)*, Aug. 2018, pp. 7114–7121.

[27] S. Tulsiani, T. Zhou, A. A. Efros, and J. Malik, “Multi-view supervision for single-view reconstruction via differentiable ray consistency,” in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2017, pp. 209–217.

[28] E. Insafutdinov and A. Dosovitskiy, “Unsupervised learning of shape and pose with differentiable point clouds,” in *Proc. Conf. on Neural Information Processing Systems (NeurIPS)*, Dec. 2018.

[29] V. Sitzmann, J. Thies, F. Heide, M. Nießner, G. Wetzstein, and M. Zollhöfer, “DeepVoxels: Learning persistent 3d feature embeddings,” in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2019.

[30] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, “CARLA: An open urban driving simulator,” in *Proc. Conf. on Robot Learning (CoRL)*, Nov. 2017.

[31] A. Geiger, P. Lenz, and R. Urtasun, “Are we ready for autonomous driving? the KITTI vision benchmark suite,” in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2012.

[32] M. F. Chang, J. Lambert, P. Sangkloy, J. Singh, S. Bak, A. Hartnett, D. Wang, P. Carr, S. Lucey, D. Ramanan, and J. Hays, “Argoverse: 3d tracking and forecasting with rich maps,” in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2019.

[33] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, “PointNet: Deep learning on point sets for 3d classification and segmentation,” in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2017.