Skoltech VectorMapNet: End-to-end Vectorized HD Map Learning

(Machine Learning Course, 2023)

Stepan Perminov [1] **Abdualziz Samra** [1] **Charlene Madida** [1] **Mohamed Sayed** [1]

Abstract

Self driving cars require a very precise maps defining road elements to be able to perform path planning and object avoidance algorithms. Existing methods already solve this problem but with post-processing manual annotations that makes it difficult to scale. Other recent learning methods produce dense rasterized segmentation predictions those miss including instance information of the individual elements in the map and still need heuristic processing to obtain the vector map. Due to this, we will implement an end-to-end vectorized HD map learning pipeline, termed VectorMapNet which takes onboard sensor observations and predicts a sparse set of polylines in the bird's-eye view. The main goal was to implement the method to construct an auto-generated local HD map for Skoltech self-driving car mapping and localization system and merge the generated local map patches to compare it with the a manually generated Skolkovo HD map using LiDar data. Implementation on neuscenes data using Camera only shows that the suggested method achieves strong map learning performance with about 0.848 mAP. Finally, we show the manually processed Skoltech HD-Map but the merging part of generated local map instances to compare with is still in progress.

Github repo: ML 2023 Project Team 34
Presentation file: Presentation file

[1]Skolkovo Institute of Science and Technology, Moscow, Russia. Correspondence to: Mohamed Sayed <mohamed.sayed@skoltech.ru>, Stepan Perminov <test.test@skoltech.ru>, Abdualziz Samra <test.test@skoltech.ru>, Charlene Madida <madida.charlene.skoltech.ru>.

Final Projects of the Machine Learning 2020 Course, Skoltech, Moscow, Russian Federation, 2020.

1. Introduction

Skoltech Self Driving Car team has been recently established and one of the main issues to solve is to create an HD map for the car to use for later tasks such as path planning. Unlike big companies, we can't afford to collect accurate data (images, LiDAR data, Radar data ...etc) for big cities and manually post process it to create the precise maps. For that, this paper, introducing the end to end VectorMap method, offered the best solution as an initial step for our algorithm to build on. In this report, we will explain this VectorMap method, its implementation, and the collected results. Moreover, for ground truth comparison, we created a manually processed HD map for Skoltech using LiDAR only data.

In general, Most recent methods (Tho) (Philion & Fidler) (Zhou & Krähenbühl) consider this problem as a semantic segmentation problem in bird's-eye view (BEV). However, the resulted raster maps are not the ideal option for autonomous driving because they lack instance information that's necessary to distinguish map elements (e.g. left and right boundaries). Moreover, in raster maps, nearby pixels might have contradicted semantics. Finally, and most importantly, these raster maps are incompatible with autonomous driving systems, such as the one we are using (Apollo), those use vectorized maps.

For this, without heuristic post-processing step for the raster maps that complicate the pipeline, VectorMap net method was proposed. the method represents the map as a set of polylines instead of a dense set of semantic pixels. These polylines are more closely related to the path planning and motion forecasting algorithms.

There are three key ingredients in our vectorized HD map generation pipeline, as shown in the figure below.

- A BEV feature extractor that map sensor data from sensor-view to a canonical BEV representation
- A scene-level map element detector that locates and classifies all map elements by predicting element keypoints and their class labels
- An object-level polyline generator that produces a polyline vertices sequence for each detected map element

The main contributions of this report are as follows:

- Implementing the method and validating on famous datasets like nuScenes and compare with the results provided by the main authors (Liu et al., 2022).
- Creating a manually processed local HD map for the ring road around Skoltech to be considered as ground truth.

Other goals those are still in progress:

- To Test the model with collected LiDAR only data (collected by Skoltech's self driving car) and demonstrate one instance of the map compared with the ground truth.
- To combine the generated local maps of the ring road around Skoltech to generate a global map then compare it with the previously created ground truth.

Figure 1. Singapore Holland Village.

2. Related Work and literature overview

2.1. Semantic map learning

Because of autonomous driving, there is a lot of interest in annotating semantic maps. Lately, the semantic segmentation problem has been used to define semantic map learning (Mattyus et al., 2015)and is resolved using HD panorama, LiDAR points, and aerial photos. The crowdsourcing tags are utilized to enhance the performance of fine-grained segmentation (Xingang et al., 2017) Recent studies (Alex et al., 2019) concentrate on interpreting BEV semantics from onboard camera images and videos rather than offline data (Baran Can et al., 2020).

2.2. Lane detection

Lane detection seeks to properly distinguish lane segments from road scenes. The majority of lane detection methods combine an intricate post-processing method with pixel-level segmentation. Another area of research uses the preset proposal to speed up inference while achieving excellent accuracy. These techniques often use handcrafted components to model proposals, such as vanishing points (Seokju Lee & Kweon, 2017), polynomial curves (Wouter Van Gansbeke & Gool, 2019), line segments , and Bézier curves (Feng Li & Zhang, 2022).

2.3. Geometric data modeling

Geometric data generation is a different area of expertise closely related to VectorMapNet. These techniques often handle geometric elements as a sequence, such as the vertices of an n-gon mesh (Cha), states of sketch strokes , and basic pieces of furniture, and produce these sequences by utilizing autoregressive models. For long-range centerline maps, the directly modeling sequence is difficult, therefore HDMapGen (Lu) sees the map as a two-level hierarchy.

3. Dataset description

3.1. Data overview

We performed model training and evaluation on the nuScenes dataset (Caesar et al., 2019). The nuScenes dataset is a public large-scale dataset for autonomous driving developed by the team at Motional. Motional is making driverless vehicles a safe, reliable, and accessible reality.

The dataset contains 1000 driving scenes collected in Boston and Singapore, two cities that are known for their dense traffic and highly challenging driving situations. One location is shown in Fig. 1. The scenes of 20 second length are manually selected to show a diverse and interesting set of driving maneuvers, traffic situations and unexpected behaviors. The collected driving data has a duration of 15h.

To facilitate common computer vision tasks, such as object detection and tracking, the team annotated 23 object classes with accurate 3D bounding boxes at 2Hz over the entire dataset. Additionally they annotated object-level attributes such as visibility, activity and pose.

The dataset includes approximately 1.4M camera images, 390k LIDAR sweeps, 1.4M RADAR sweeps and 1.4M object bounding boxes in 40k keyframes.

3.2. Car setup

Two Renault Zoe cars with identical sensor layouts were used to drive in Boston and Singapore. The placement of the sensors is provided in Fig. 2. A list of sensors is provided below:

- 1x spinning LIDAR (Velodyne HDL32E);
- 6x camera (Basler acA1600-60gc)
- 5x long range RADAR sensor (Continental ARS 408-21);
- 1x IMU & GPS (Advanced Navigation Spatial).

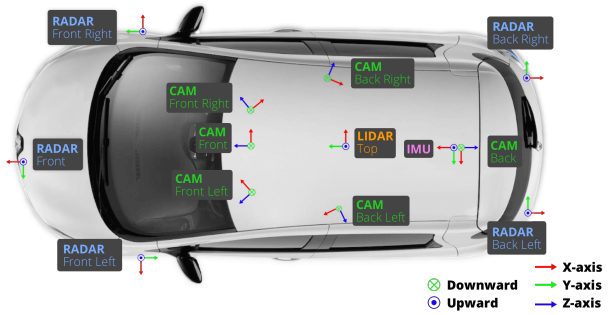

Figure 2. Sensor setup for the data collection platform.

3.3. Dataset usage details

For our project we downloaded a full nuScenes dataset (v.1.0), released in March 2019. For the training, validation and test steps we used subsets of 700, 150 and 150 scenes correspondingly. In addition, to obtain ground truth data of roads, crosswalks, etc. we downloaded a nuScenes map expansion pack with 11 semantic layers (crosswalk, sidewalk, traffic lights, stop lines, lanes) and put it inside the "maps" folder of the dataset. As we used images for the training and validation, it's worth to mention that initially all dataset images had 1600×90 resolution. As for the preprocessing step, we downscaled the images to 256×256 size and modified camera intrinsics data correspondingly.

4. Algorithms and models

4.1. Overall Architecture of the Model

The first stage is BEV feature extractor, its output is a 3-way feature map with shape (200, 100, 128). the shared CNN backbone for BEV is ResNet50 (He et al., 2016). To aggregate LiDAR points into a pillar, a single layer PointNet (Qi et al., 2016) was used as a backbone with outputs of 64 dimensions. The number of element queries N_{max} in

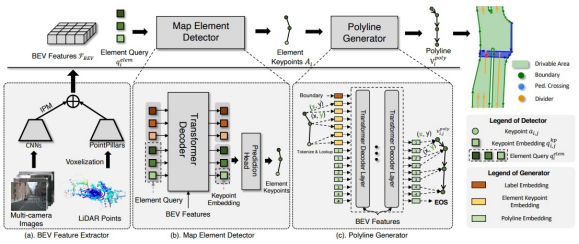

Figure 3. The network architecture of VectorMapNet. The top row is the pipeline of VectorMapNet generating polylines from raw sensor inputs. The bottom row illustrates detailed structures and inference procedures of three primary components of VectorMapNet: BEV feature extractor, map element detector, and polyline generator. Numbers in polyline embeddings indicate predicted vertex indexes. .

map element detector was set 100. The used transformer decoders in both map element detector and polyline generator consist of 6 decoder layers, with hidden embeddings of size 256. The map was space was divided into 200×100 rectangular grids of size $0.3\text{m} \times 0.3\text{m}$.

4.2. BEV feature extractor

The BEV feature extractor lifts various modality inputs into a unified feature space and aggregates these features into a canonical representation termed BEV features F_{BEV}. In this project surrounding camera images I were considered as a modality only.

We used ResNet50 to extract features from images I in the cameras' space, then used the Inverse Perspective Mapping (IPM) technique (Mallot et al., 1991) to transform these features into BEV space. Since the depth information is missing in camera images, we followed one common approach that assumed the ground is mostly planar and transformed the images to BEV via homography. Without knowing the exact height of the ground plane, this homography was not an accurate transformation. To alleviate this issue, we transformed the image features into four BEV planes with different heights: -1m, 0m, 1m, 2m. The camera BEV features $F^I_{BEV} \in R^{W \times H \times C}$ are the concatenation of these feature maps.

4.3. MAP ELEMENT DETECTOR

The objective of the map element detector is to infer element keypoints $a_{i,j}$ from the BEV features F_{BEV} after acquiring BEV features. To do this, we use a modified transformer set prediction detector (Carion et al., 2020). The positions and types of map items are represented by this detector by predicting their class labels L and element keypoints A.

Keypoint representations. In object detection problems, people use bounding box to abstract the shape of an object. Here we use k key point locations $A_i = \{a_j \in R_2 | j = 1, ..., k\}$, to represent the outline of a map element. However, defining keypoints for map elements is not straightforward since their are diverse. Authors conduct an ablation study to investigate the performance of different choices, this study indicated that bounding box(the smallest box enclosing a polyline) leads to better mAP than other two methods; Start-Middle-End (SME) and Extreme Points (left-most, right-most, top-most and bottom-most of the polyline).
Element queries. the input of the encoder is formulated as a sequence of learnable queries: $\{q_i^{\text{elem}} \in R^{k \times d} | i = 1, 2, \ldots, N_{\text{max}}\}$, where d is the hidden embeddin size, and the i-th element query is composed of k keypoints embeddings $q^{\text{kp}} : q_i^{\text{elem}} = \{q_{i,j}^{\text{kp}} \in R^d | j = 1, 2, \ldots, N_{\text{max}}\}$. **Architecture** The prediction head has two MLPs, which decodes element queries into element keypoints $a_{i,j} = \text{MLP}_{\text{kp}}(q_{i,j}^{\text{kp}})$ and their class labels $l_i = \text{MLP}_{\text{cls}}([q_{i,1}^{\text{kp}}, q_{i,2}^{\text{kp}}, \ldots, q_{i,k}^{\text{kp}}])$.

4.4. POLYLINE GENERATOR

The objective of the polyline generator is to produce a comprehensive geometrical shape of the map elements from the label and keypoints. In particular, polyline generator simulates a distribution across each polyline's vertices based on the initial layout's BEV characteristics and element keypoints. Specifically, polyline generator models a distribution $p(V_i^{\text{poly}} | a_i, l_i, \mathcal{F}_{\text{BEV}}^f)$ over the vertices of each polyline, conditioned on the initial layout (i.e., element keypoints and class labels) and BEV features. This distribution was decomposed into a product of conditional distributions related to each vertex, and then modeled using autoregressive network.

Architecture. The chosen autoregressive network is a vanilla transformer (Vaswani et al., 2017). The query inputs of the transformer decoder are set of tokens that each consist of polyline's keypoint coordinates and class label. Then that sequence of vertex tokens are fed into the transformer iteratively, BEV features are integrated with cross-attention, then tokens are decoded as polyline vertices. This generator can generate all polylines in parallel. Authors used an addition of three learned embeddings as the embedding of each vertex token as in PolyGen (Nash et al., 2020). These embeddings are: "*Coordinate Embedding*, indicating whether the token represents x or y coordinate; *Position Embedding*, representing which vertex the token belongs to; *Value Embedding*, expressing the token's quantized coordinate value."

5. Experiments

5.1. Training Setting (Single GPU)

We train all our models on 1 RTX3060 GPU for 36 epochs with a total batch size of 32. We use AdamW (Loshchilov & Hutter) with the same parameters of the original work. We made some adjustments to fit our hardware, we set both parameters 'samples_per_gpu' and 'worker_per_gpu' to 1 instead of 5 and 8 respectively. We trained image data only as the code provided by the authors did not consider LiDAR data.

5.2. Training Setting (Multiple GPUs)

We tried to train the model on *Zhores* supercomputer at Skoltech, we faced some issues related to run distributed training procedure.

5.3. Evaluation Metrics

We used Chamfer distance [see 8.1] to determine if our predictions match the ground truth or not. In other words, we consider the predicted map element to match the ground truth element if Chamfer distance between the is less than some threshold. In our experiments, we fixed this threshold to 1.5.

6. Results

6.1. Method Implementation on Nuscenes

In order to evaluate performance of the models, the following metrics were calculated:

- Recall;
- Average Precision;
- Mean Average Precision.

Evaluation results for the weights provided by authors is provided in Table. 1. The obtained Mean Average Precision value was 0.848.

Table 1. Evaluation results for author weights.

CLASS	RECALL	AVERAGE PRECISION
PEDESTRIAN CROSSING	0.974	0.956
DIVIDER	0.939	0.890
CONTOURS	0.809	0.699

Evaluation results for the weights trained from the scratch up to Epoch 36 is provided in Table. 2. The obtained Mean Average Precision value was 0.454.

Table 2. Evaluation results for weights trained from the scratch.

CLASS	RECALL	AVERAGE PRECISION
PEDESTRIAN CROSSING	0.961	0.679
DIVIDER	0.717	0.458
CONTOURS	0.496	0.226

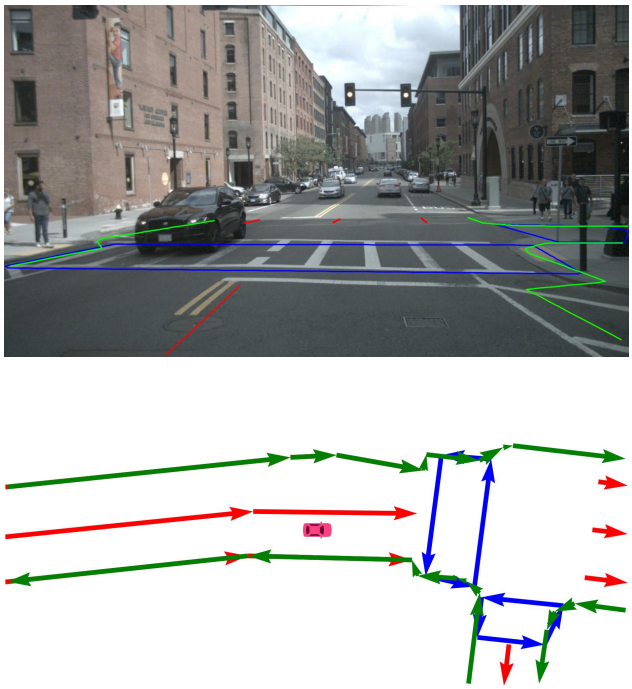

Figure 4. Road data extracted from the dataset: [above] Front camera view; [below] Bird eye view. Pedestrian crosswalk boundaries (blue vectors), road contours (green vectors), dividers (red vectors).

Ground Truth data extracted from the nuScenes dataset is shown in Fig. 4. Results of predicting corresponding classes are shown in Fig. 5.

6.2. Created Skoltech HD map ground truth

For this task, LiDAR data were collected using our Skoltech car with velodyne (Vel) 3D liDARs, and flat liDARs installed onboard. The point cloud was then processed using Apollo (Apo) architecture V7.0 with manual processing. The resulting ground truth map is shown as in figure(6).

7. Conclusion

Unlike previous works, VectorMapNet uses polylines as the bases to represent the vectorized HD map elements. We believe that this end to end model will provide a new perspective to tackle the HD semantic map learning problem. It

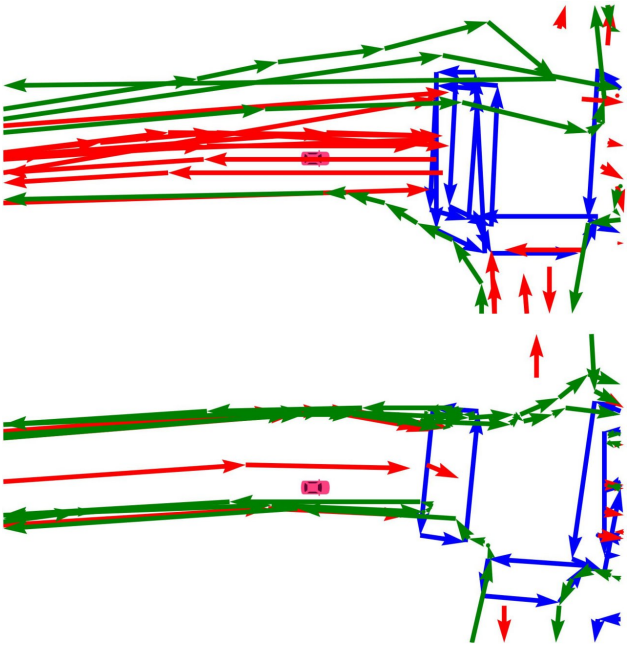

Figure 5. [above] sample of map detection for our experiment (36 epochs of training). [below] sample of map detection using authors' checkpoint (110 epochs)

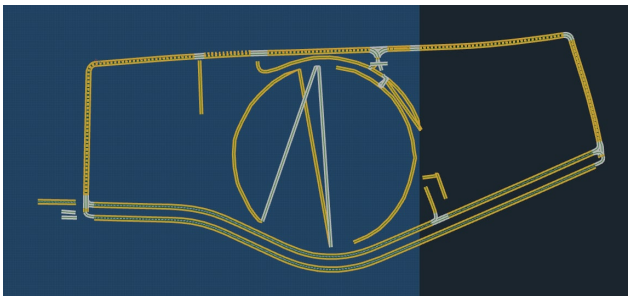

Figure 6. Skoltech HD Map (Ground Truth)

works by decomposing the learning routine into a detection and generation problem. Overall, the results show that it can generate coherent and complex geometries for the chosen map elements. Further work is still in progress to verify the method on collected real world data and extra future work is needed to smooth and merge the produced map instances to produce a global map.

References

URL https://velodynelidar.com/products/.

Alex, H. L., Sourabh, V., Holger, C., Lubing, Z., Jiong, Y., and Oscar, B. Pointpillars: Fast encoders for object detection from point clouds. *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

Baran Can, Y., Liniger, A., Unal, O., Danda, P., and Gool, L. V. Understanding bird's-eye view semantic hd-maps using an onboard monocular camera. *arXiv preprint arXiv:2012.03040*, 2020.

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

Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., and Zagoruyko, S. End-to-end object detection with transformers. In Vedaldi, A., Bischof, H., Brox, T., and Frahm, J.-M. (eds.), *Computer Vision – ECCV 2020*, pp. 213–229, Cham, 2020. Springer International Publishing. ISBN 978-3-030-58452-8.

Feng Li, Hao Zhang, S. L. J. G. L. M. N. and Zhang, L. Accelerate detr training by introducing query denoising. *arXiv preprint arXiv:2203.01305*, 2022.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016. doi: 10.1109/CVPR.2016.90.

Liu, Y., Yuantian, Y., Wang, Y., Wang, Y., and Zhao, H. Vectormapnet: End-to-end vectorized hd map learning. *arXiv preprint arXiv:2206.08920*, 2022.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. URL https://github.com/loshchil/AdamW-and-SGDW.

Mallot, H. A., Bülthoff, H. H., Little, J., and Bohrer, S. Inverse perspective mapping simplifies optical flow computation and obstacle detection. *Biological cybernetics*, 64(3):177–185, 1991.

Mattyus, G., Urtasun, R., and Fidler, S. Deeproadmapper: Extracting road topology from aerial images. *arXiv preprint arXiv:1511.02641*, 2015.

Nash, C., Ganin, Y., Eslami, S. M. A., and Battaglia, P. PolyGen: An autoregressive generative model of 3D meshes. In III, H. D. and Singh, A. (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 7220–7229. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/nash20a.html.

Philion, J. and Fidler, S. Lift, splat, shoot: Encoding images from arbitrary camera rigs by implicitly unprojecting to 3d.

Qi, C. R., Su, H., Mo, K., and Guibas, L. J. Pointnet: Deep learning on point sets for 3d classification and segmentation. 12 2016. URL http://arxiv.org/abs/1612.00593.

Seokju Lee, Junsik Kim, J. S. Y. S. S. O. B. N. K. T. L. H. S. H. S.-H. H. and Kweon, I. S. Vpgnet: Vanishing point guided network for lane and road marking detection and recognition. *Proceedings of the IEEE international conference on computer vision*, 2017.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Wouter Van Gansbeke, Bert De Brabandere, D. N. M. P. and Gool, L. V. Endto-end lane detection through differentiable least-squares fitting. *IEEE/CVF International Conference on Computer Vision Workshops,*, 2019.

Xingang, P., Jianping, S., Ping, L., Xiaogang, W., and Xiaoou, T. Spatial as deep: Spatial cnn for traffic scene understanding. *arXiv:1712.06080*, 2017.

Zhou, B. and Krähenbühl, P. Cross-view transformers for real-time map-view semantic segmentation.

8. Appendix

8.1. Chamfer Distance

Chamfer distance is used to measure the similarity between two unordered sets of points, it considers the mutual distance between each permutation of points of the two sets. It can be calculated using the formula:

$$D_{\textbf{Chamfer}}(S_1, S_2) = \frac{1}{2}\Big(\frac{1}{|S_1|}\sum_{p \in S_1} \min_{q \in S_2} \|p, q\|_2 + \frac{1}{|S_1|}\sum_{p \in S_2} \min_{q \in S_1} \|p, q\|_2\Big)$$

A. Team member's contributions

Explicitly stated contributions of each team member to the final project.

Stepan Perminov (25% of work)

- Conducted an evaluation of the paper results provided by authors.
- Trained the model on the nuScenes image dataset from the scratch and evaluated it.
- Recorded a lidar dataset from the Skoltech SDC.
- Tried to train the model on the recorded lidar datset but did not succeed.
- Prepared the GitHub Repo
- Prepared Sections 1.1 and 3 of this report

Abdualziz Samra (25% of work)

- Run reproduction experiments on Zhores.
- Edit Section 4 and 5 , and 8.1 of this report.

Charlene Madida (25% of work)

- Edited sections 1 and 2 of the report
- Theoritical research

Mohamed Sayed (25% of work)

- Theoretical Research
- Prepared the Github Repo
- Created the PowerPoint presentation
- Trained and evaluate dthe model only using the neuscenes miniset data.
- Still working on the merging algorithm (to merge two instances of the map) probably will take more time.
- Proofreading of the report.

B. Reproducibility checklist

Answer the questions of following reproducibility checklist. If necessary, you may leave a comment.

1. A ready code was used in this project, e.g. for replication project the code from the corresponding paper was used.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   General comment: If the answer is **yes**, students must explicitly clarify to which extent (e.g. which percentage of your code did you write on your own?) and which code was used.

   Students' comment: A forked repo of the original authors' code was used for the main part. For the local Skoltech HD ground truth generation, we used apollo software V7.0. Moreover, the code for the merging part is still in progress (not included yet in the repo)

2. A clear description of the mathematical setting, algorithm, and/or model is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: You can find the main needed theory part in the algorithms section 4.

3. A link to a downloadable source code, with specification of all dependencies, including external libraries is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: A link to the GitHub repo with detailed implementation steps was provided.

4. A complete description of the data collection process, including sample size, is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: Detailed dataset description as well as its collection process are provided in Section 3.

5. A link to a downloadable version of the dataset or simulation environment is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: The link to the downloadable version of the dataset is provided in Section 3.3.

6. An explanation of any data that were excluded, description of any pre-processing step are included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: The image dataset preprocessing step is described in Section 3.3.

7. An explanation of how samples were allocated for training, validation and testing is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: Dataset distribution among training, validation and test subsets is provided in Section 3.3.

8. The range of hyper-parameters considered, method to select the best hyper-parameter configuration, and specification of all hyper-parameters used to generate results are included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: We adapt most of values of hyper-parameters from the original work. When changes are made, we mentioned that in relevant sections like 5.1 5.3

9. The exact number of evaluation runs is included.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: Number of evaluation epochs is provided in Section 6.1.

10. A description of how experiments have been conducted is included.

    ☑ Yes.
    ☐ No.
    ☐ Not applicable.

    Students' comment: in 5,

11. A clear definition of the specific measure or statistics used to report results is included in the report.

- [x] Yes.
- [ ] No.
- [ ] Not applicable.

Students' comment: mAP based on thresholded Chamfer distance was described in 5.3 and 8.1.

12. Clearly defined error bars are included in the report.

- [ ] Yes.
- [ ] No.
- [x] Not applicable.

Students' comment: In the task of map element detection, error margins are defined in the evaluation metric as a threshold [see 5.3].

13. A description of the computing infrastructure used is included in the report.

- [x] Yes.
- [ ] No.
- [ ] Not applicable.

Students' comment: The two experiments of training the model using single GPU and multiple GPUs (failed) are described in 5.1 and 5.2.