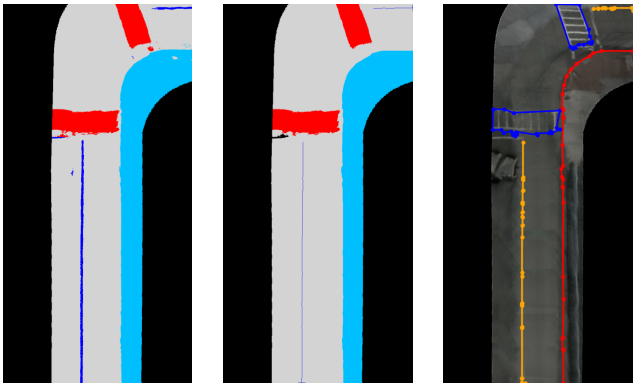
PseudoMapTrainer: Learning Online Mapping without HD Maps

Supplementary Material



(a) BEV rendering (b) Smoothed (c) Vectorization

Figure S.1. **Postprocessing and vectorization.** We remove artifacts in the raw semantic BEV rendering and further smooth the road boundary for accurate polyline and polygon extraction.

A. Postprocessing

Our postprocessing pipeline refines the initial BEV segmentation to address common artifacts introduced by the surface reconstruction and generates vectorized map elements suitable for training, as shown in Fig. S.1. In practice, we extend the BEV renderings by a small margin before postprocessing it to avoid boundary effects. After we obtain the vectorized elements, we crop them to the desired range.

A.1. Overall pipeline

The postprocessing pipeline consists of the following steps:
Removing artifacts. Due to inaccuracies in the surfel optimization, small segments can be misclassified. To reduce these artifacts, we employ a class-based connected-component labeling [39] to identify small segments enclosed by other segments. These are then reassigned to the enclosing classes to ensure semantic consistency. Small segments that are adjacent to more than one class are removed by assigning them either to the BEV mask or to one of the adjacent classes. We also remove lane-marking segments (dark blue in Fig. S.1a) that are unreasonably thick.
Extracting the road boundary. We apply morphological filtering to the outside class (light blue in Fig. S.1), resulting in a smoother road boundary, which is extracted by the border between segments of the road class and segments of the outside class.
Vectorization of line-shaped elements. We connect spatially close lane-marking fragments through dilation to become the lane dividers. We skeletonize the lane divider and road boundary segments using the Zhang-Suen algorithm [54] into line components. For Y-shaped lines, the longest path is preserved, while the other branches become new components. The lines are subsequently converted into polylines and simplified through iterative polygonal approximation based on the Ramer-Douglas-Peucker (RDP) algorithm [9]. We initialize the maximum distance threshold with $\epsilon^{(1)}$ and iteratively increase it as $\epsilon^{(t)} = \epsilon^{(1)}t$ until the simplified polyline contains no more than L points. Finally, lane dividers that are overly close and parallel to boundaries or pedestrian crossings are removed.
Vectorization of polygon-shaped elements. To extract the borders of pedestrian crossings, we first employ the Suzuki-Abe border-following algorithm [44]. Similar to line-shaped elements, we then apply the RDP algorithm to obtain a simplified yet accurate polygonal representation.

Table S.1. Pipeline ablation for the observed region in single trips. The default parameters are: 20 pixel/m, 15, and 5 cm.

AP	default	**Resol. [pixel/m]**			**Kernel size**			**Dist. step $\epsilon^{(1)}$ [cm]**		
		5	10	40	1	5	25	1	20	100
ped.	23.6	24.0	22.5	22.8	23.4	23.4	23.4	20.8	23.4	23.4
div.	6.0	3.1	5.1	6.5	3.7	4.9	6.7	4.8	4.9	5.0
bdry.	26.8	26.8	25.1	27.0	27.8	27.5	25.9	26.6	27.3	27.4
mean	18.8	18.0	17.6	18.8	18.3	18.6	18.6	17.4	18.5	18.6

A.2. Parameter ablation

For the postprocessing, there come many parameters with every filter and every algorithm we add. Thus, we mostly manually fine-tuned them based on qualitative BEV results. However, we provide an ablation for the key pipeline parameters in Tab. S.1, using pseudo labels on the same validation set as in our main experiments. We evaluate the BEV resolution, the kernel size for the morphological dilation of the lane-markings, and $\epsilon^{(1)}$, the initial distance threshold and step size, for polyline and polygon simplification. A higher resolution improves the lane dividers but slightly reduces pedestrian crossing AP, which can be explained by their different shape types. Larger dilation kernels show to significantly improve the lane preservation as fragmented lanes get connected again. We also notice that a too small $\epsilon^{(1)}$ (i.e., a too faithful approximation) harms the quality of pedestrian crossings. To ensure full reproducibility, we published the code.

B. Linear Program Formulation

We perform both one-to-one and one-to-many assignment to optimally match elements between predictions and fragmented pseudo-labels. Thereby, we formulate a binary in-

teger linear program with the following constraints: each pseudo-label element is assigned exactly once, and each prediction is assigned at most once. Our objective is to minimize the total matching cost.

Let the binary variable

$$x_{ij} \in \{0,1\}, \quad \forall i \in Q_{\text{ind}},\ j \in G_{\text{ind}}, \tag{8}$$

denotes the direct one-to-one assignment between the predicted element q_i and the pseudo-label element g_j, and

$$y_{iJ} \in \{0,1\}, \quad \forall i \in Q_{\text{ind}},\ J \in \mathcal{J}, \tag{9}$$

denotes a one-to-many assignment between the predicted element q_i and a set of pseudo-label elements $\{g_j\}_{j \in J}$. We enforce that every pseudo-label element should be assigned exactly once by

$$\sum_{i \in Q_{\text{ind}}} \left(x_{ij_G} + \sum_{J \in \mathcal{J} | j_G \in J} y_{iJ} \right) = 1, \quad \forall j_G \in G_{\text{ind}}, \tag{10}$$

and that every prediction should be assigned not more than once by

$$x_i + y_i \leq 1, \quad \forall i \in Q_{\text{ind}} \tag{11}$$

with $x_i = \sum_{j \in G_{\text{ind}}} x_{ij}$ as the one-to-one flag and $y_i = \sum_{J \in \mathcal{J}} y_{iJ}$ as the one-to-many flag. The overall objective is to minimize the total cost:

$$\min_{\{x_{ij}\},\{y_{iJ}\}} \sum_{i \in Q_{\text{ind}}} \left(\sum_{j \in G_{\text{ind}}} c_{\text{o2o}}(q_i, g_j)\, x_{ij} + \sum_{J \in \mathcal{J}} c_{\text{o2m}}(S^i, J)\, y_{iJ} \right) \tag{12}$$

yielding an optimal matching denoted as x^*_{ij}, y^*_{iJ}.

C. Centerlines

In addition to the evaluated map classes, centerlines are imaginary lines that run along the middle of driving lanes and serve as crucial references for planning. However, since we cannot infer these lines from our BEV segmentation, we suggest two potential approaches for future work: deriving centerlines from ego trajectories or extracting them from parallel polylines of existing map elements. Both methods have limitations, such as ego trajectories reflecting overtaking maneuvers and parallel polylines introducing ambiguities at intersections. A promising strategy might be combining these approaches for mutual verification. Fig. S.2 provides a preliminary example of centerlines generated from multiple ego trajectories.

D. Qualitative Results

Further qualitative results of our pseudo-labels compared to the ground-truth map are shown in Fig. S.3.

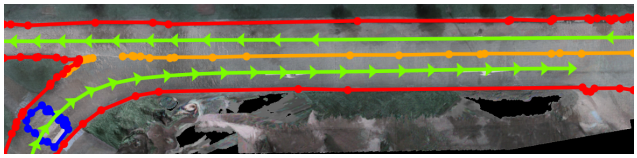
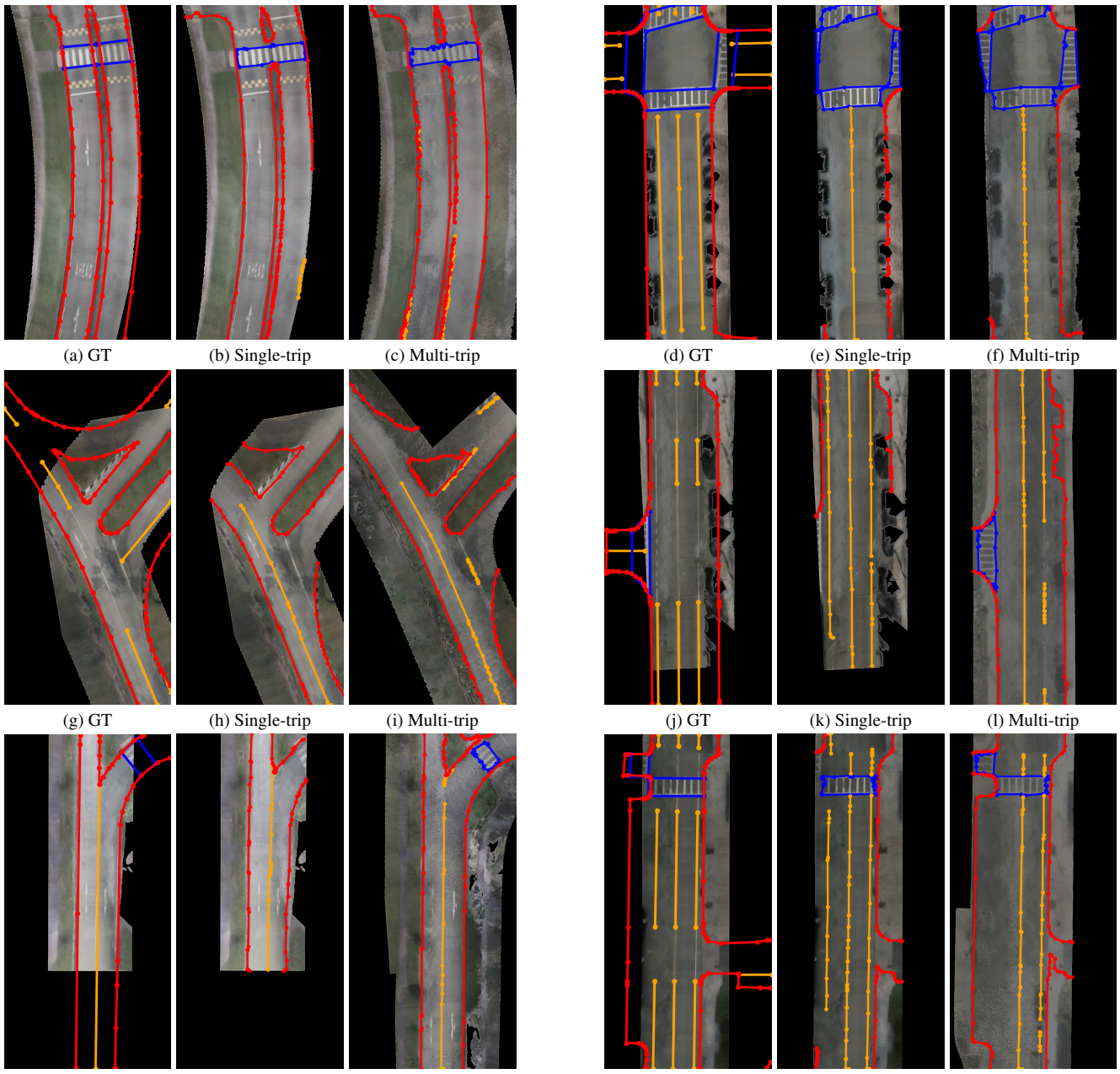


Figure S.2. Centerlines derived by multi-trip ego trajectories.

E. Pseudo-Label Generation Details

We train the Mask2Former [6] segmentation model on 1x NVIDIA A100 GPU with a Swin-L transformer [27] with 200 queries as its backbone and pre-trained on ImageNet-21k [40]. For the surface optimization, we also use 1x A100 GPU. For combining data from multiple trips, we limit the maximum number of trips to 50 to reduce runtime and avoid memory peaks. We group trips based on the minimum distance of their ego trajectories and deliberately exclude combinations of trips from the training set with those from the validation set. For the meshgrid expansion, we follow [10] and choose an offset of $r = 7$ m along the ego trajectory.



(a) GT (b) Single-trip (c) Multi-trip (d) GT (e) Single-trip (f) Multi-trip

(g) GT (h) Single-trip (i) Multi-trip (j) GT (k) Single-trip (l) Multi-trip

(m) GT (n) Single-trip (o) Multi-trip (p) GT (q) Single-trip (r) Multi-trip

Figure S.3. **Additional qualitative results.** We plot the lane dividers (orange), road boundaries (red), and pedestrian crossings (blue) for pseudo-labels and ground truth (GT).