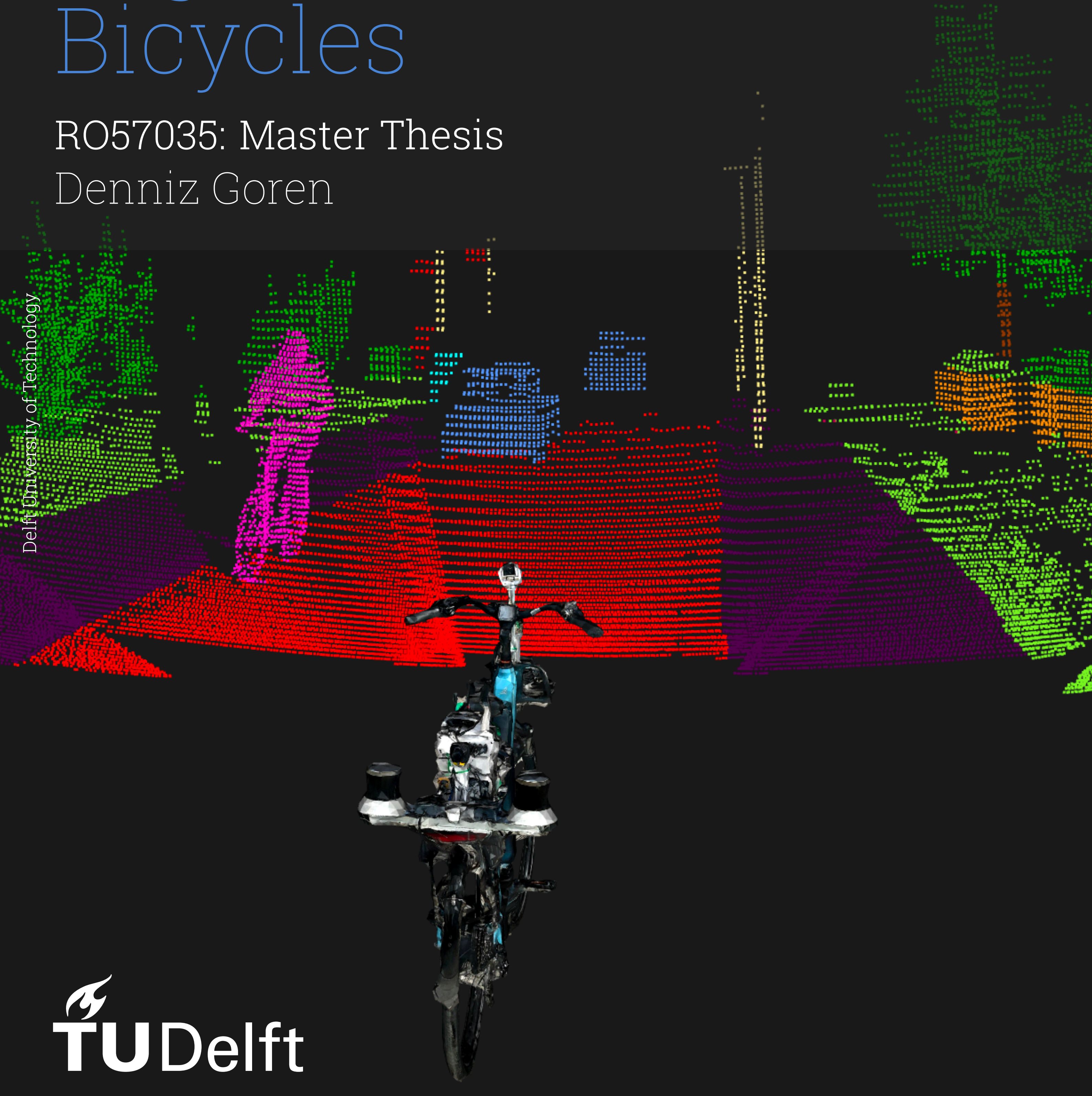
# Online LiDAR Semantic Segmentation for Bicycles

RO57035: Master Thesis

Denniz Goren



TU Delft

# Online LiDAR Semantic Segmentation for Bicycles

by

Denniz Goren

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Wednesday May 28, 2025 at 12:30 PM.

| | | |
|---|---|---|
| Student number: | 4495543 | |
| Project duration: | February 26, 2024 – May 28, 2025 | |
| Thesis committee: | Dr. H. Caesar, | TU Delft, supervisor |
| | Dr. J. Kooij, | TU Delft |

TU Delft

# Online LiDAR Semantic Segmentation for Bicycles

Denniz Goren

***Abstract*— The vulnerability of cyclists, exacerbated by the rising popularity of faster e-bikes, motivates adapting automotive perception technologies for bicycle safety. This paper presents an online 3D LiDAR semantic segmentation pipeline developed using our multi-sensor 'SenseBike' research platform. To bridge the automotive-to-bicycle domain gap, we introduce the novel BikeScenes LidarSeg Dataset[1], comprising 3021 consecutive LiDAR scans around the TU Delft campus, semantically annotated for 29 dynamic and static classes. By evaluating model performance, we demonstrate that fine-tuning on our BikeScenes dataset achieves a mean Intersection-over-Union (mIoU) of 63.6%, significantly outperforming the 13.8% obtained with SemanticKITTI pre-training alone. This result underscores the necessity and effectiveness of domain-specific training. We highlight key challenges specific to bicycle-mounted, hardware-constrained perception systems and contribute the BikeScenes dataset as a resource for advancing research in cyclist-centric LiDAR segmentation.**

## I. INTRODUCTION

The inherent vulnerability of cyclists presents a persistent safety challenge, particularly in regions like the Netherlands, where cycling is vital to urban mobility. This situation is further complicated by the integration of faster e-bikes, introducing new potential risks that demand attention. While advanced driver-assistance systems (ADAS), powered by sensors and the integration of machine learning, are significantly enhancing safety in automobiles, similar technologies for bicycles are far less developed. With sensors such as LiDAR becoming more compact and affordable, the question arises: can we leverage these advancements to improve bicycle safety by providing riders, or assistive systems, with real-time awareness of their complex surroundings?

This work explores this question by developing and evaluating a real-time 3D LiDAR semantic segmentation system specifically designed for a bicycle platform. Understanding the environment – accurately identifying roads, sidewalks, vehicles, pedestrians, cyclists, and other obstacles – is a valuable foundation for any potential bicycle ADAS or enhanced rider awareness system. To facilitate this research, we utilize the SenseBike (Figure 14), a research platform based on the Boreal Holoscene X [2] and equipped with a sensor suite including three LiDAR units, an IMU, GPS, cameras, and an NVIDIA Jetson for compute (See Appendix I for details).

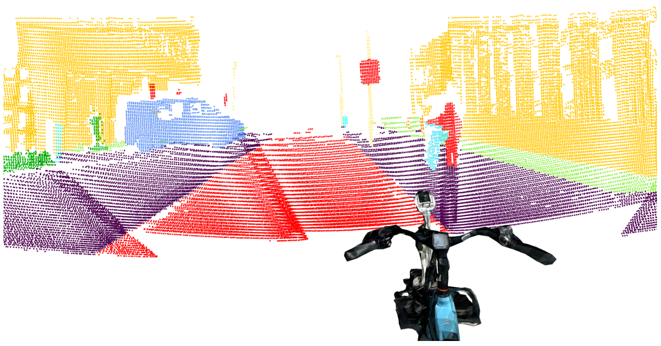


Fig. 1: Scene from the BikeScenes LidarSeg Dataset.

Implementing such a system on a bicycle presents unique challenges, including sensor calibration on a dynamic platform, correcting motion-induced distortions (skew) in LiDAR scans, and adapting perception models trained on automotive data to the distinct bicycle perspective. This paper details our approach to address these challenges, encompassing:

- Ego-motion compensation to deskew LiDAR scans using EKF-based state estimation.
- Creation of the BikeScenes LidarSeg Dataset, capturing LiDAR data from the SenseBike's perspective to bridge the domain gap with existing automotive datasets.
- Selection and evaluation of an efficient deep learning model (FRNet [27]) for semantic segmentation, balancing accuracy and suitability for resource-constrained onboard hardware.
- Integration within a ROS2 framework for online operation.

Our primary contribution lies in demonstrating the feasibility of an online LiDAR semantic segmentation

[1]Publicly available in our GitHub repository.

pipeline on a bicycle platform and quantitatively evaluating the performance of a state-of-the-art model within this specific, challenging domain using the newly created BikeScenes LidarSeg Dataset.

## II. RELATED WORK

### A. Outdoor LiDAR Segmentation Datasets

Large-scale automotive datasets such as SemanticKITTI [1], nuScenes [4], and Waymo [20] are today's primary publicly available data source and benchmarks for 3D semantic segmentation. However, all these datasets are captured from sensors mounted on the roofs of cars and do not transfer well to the domain of a bicycle, given the inherent differences in perspective and positioning of these vehicles on the public roads. This domain gap also gets exacerbated when we are presented with differences in sensors: field-of-view, intensity distributions, and scan-lines and patterns. The Salzburg Bicycle LiDAR Dataset [13] was the first bicycle-centric LiDAR segmentation dataset, focusing primarily on static environmental classes such as roads, buildings, and vegetation. To evaluate and bridge the domain gap further, more data from a bicycle's perspective is required.

### B. LiDAR Segmentation Architectures

Deep learning methods dominate the task of LiDAR semantic segmentation. Common approaches process the irregular point cloud structure by operating directly on points [18, 22], using voxel representations [29, 8], projecting onto 2D range views [27, 5], or combining point and voxel information [21]. Recently, Transformer-based models have set new benchmarks by leveraging self-attention to capture global context [28, 24, 23, 11]. A key challenge, however, for online applications like the SenseBike is the trade-off between the high accuracy of state-of-the-art models [23, 11, 27] and the computational efficiency required for deployment on resource-constrained hardware.

## III. METHODOLOGY

This section details the methodology developed for our online 3D LiDAR semantic segmentation pipeline on the SenseBike research platform. Our approach involves several stages: multi-sensor calibration, LiDAR scan deskewing via ego-motion compensation, creating the novel BikeScenes dataset, neural network training for segmentation, and system integration within ROS2 for on-bike operation. Each stage is further elaborated in the following subsections.

### A. Sensor Calibration

To express every sensor's measurements in a common coordinate frame, we perform an extrinsic calibration of the LiDARs, IMU, GPS, and front camera. We first coarsely measure each sensor's translation and orientation on the bike using tape measures and plumb lines. Given that the internal beam origin of each LiDAR is unspecified, we refine the LiDAR→`base_link` transformation by groundplane segmentation and registering rear-unit scans (Helios Left and Right) to the front RoboSense M1 Plus via point-to-plane ICP. All final transforms are embedded in our URDF, which incorporates a 3D mesh of the SenseBike created using the Polycam application [16]. For an in-depth overview of the calibration procedure and the resulting positioning of the sensors, see Appendix I.

### B. Ego-Motion Compensation

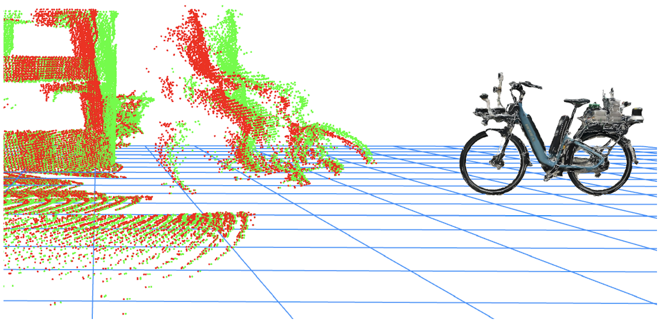


Fig. 2: Sideview of a cyclist and building before (🟥) and after (🟩) ego-motion compensation.

A mobile-LiDAR scan is *skewed*: every beam is fired at a slightly different bicycle pose. We implement our method to compensate for this ego-motion in two stages:

1) **State estimation** – fusing IMU and GPS in an Extended Kalman Filter using the `robot_localization` package [12]) to obtain the velocities of the bicycle: $\mathbf{v}(t) = \begin{bmatrix} v_x \, v_y \, v_z \end{bmatrix}^\top$, $\boldsymbol{\omega}(t) = \begin{bmatrix} \omega_x \, \omega_y \, \omega_z \end{bmatrix}^\top$. Slow gyro drift can rotate $\mathbf{v}$ away from the true bicycle heading. We therefore keep only its scalar *forward speed* $s = \|\mathbf{v}\|$.
2) **Deskewing** – For every raw point time-stamped $t$, our ROS2 `deskew_node` applies

$$\mathbf{p}_{\text{corr}} = \mathbf{p}_{\text{raw}} - (\boldsymbol{\omega} \times \mathbf{p}_{\text{raw}})\,\Delta t - \begin{bmatrix} s & 0 & 0 \end{bmatrix}^\top \Delta t,$$
$$\Delta t = t_{\text{end}} - t.$$

where $t_{\text{end}}$ is the time-stamp carried in the point-cloud header, which is based on the last laser firing time. The translation is projected onto the bicycle's forward $x$-axis. This method assumes that the slip is minimal and a constant twist for the duration of the scan (100 ms). The corrected scan is published on `/rslidar/M1P_deskewed`.

Appendix II provides a full overview of the ego-motion compensation pipeline.

## C. Model Selection

Apart from maximising segmentation quality, the model must run within an online pipeline on our target hardware, an NVIDIA Jetson Orin NX (1.9 FP32 TFLOPS). Inference speed is therefore as important as mIoU, when it comes to selecting a model. Because the raw FPS reported in the literature depends strongly on the GPU used, we define a hardware-agnostic metric,

$$\text{normalized FPS} = \frac{\text{author-reported FPS}}{\text{GPU peak FP32 TFLOPS}}$$

which approximates how many frames a network can process per unit of compute. We compare the leading methods (Table I) on:

1) **Performance** – official test-set mIoU on SemanticKITTI and nuScenes.
2) **Efficiency** – normalized FPS.

Figure 3 shows the model performance against the normalized FPS metric for the leading segmentation models on the SemanticKITTI benchmark. We compute the metric based on author-reported FPS and hardware. As we can see, FRNet delivers the best accuracy–efficiency balance. Its mIoU is 73.3% on SemanticKITTI and 82.5% on nuScenes, only 2.2 and 0.2 points below the leading Point Transformer V3, respectively. Yet, its theoretical inference speed is 2.05 frames per TFLOPS, compared with PTV3's 0.27, a more than 7.5× difference in compute-efficiency. We therefore adopt FRNet for all subsequent experiments in Section V.

## D. ROS2 Inference Node and Performance Optimization

To deploy and manage the segmentation pipeline on the SenseBike, a dedicated ROS2 node, `FRNetROSInferenceNode`, was developed. This node subscribes to incoming deskewed LiDAR scans, executes the FRNet segmentation model detailed in Section III-C, and publishes the resulting semantically labeled point clouds. Automatic Mixed

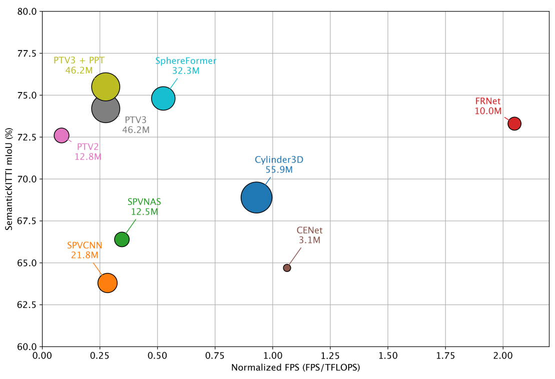


Fig. 3: mIoU on the SemanticKITTI test set versus the normalized FPS.

Precision (AMP) is employed during inference to optimize latency for online operation. A description of the ROS2 node implementation, the specifics of AMP, and an analysis of its impact on inference speed and model performance are provided in Appendix IV.

## IV. BIKESCENES LIDARSEG DATASET

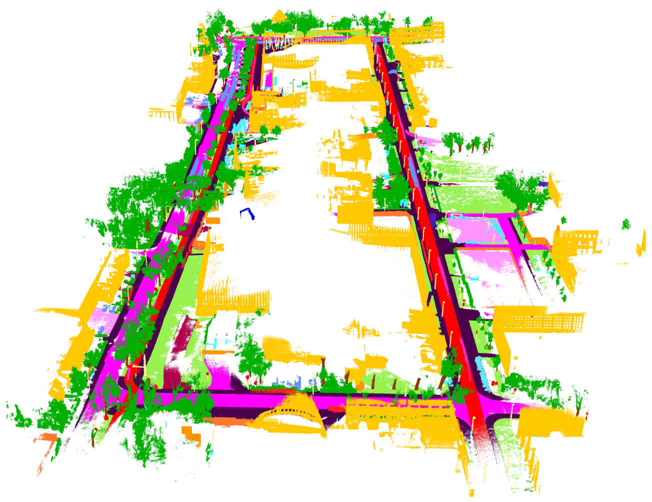


Fig. 4: Map of aggregated labeled scans.
Legend: 🟨 building; 🟪 road; ⬛ sidewalk; 🟩 vegetation; 🟥 bike-path.

Public datasets like SemanticKITTI [1] and nuScenes [4] have become cornerstone benchmarks, significantly advancing 3D automotive perception. However, their direct applicability to bicycle-centric perception is limited. These datasets are predominantly recorded using sensors typically mounted on car

TABLE I: Comparison of leading LiDAR-segmentation networks on segmentation performance, author-provided hardware specifications and inference speed, and our normalized fps metric.

| Model | Year | SemanticKITTI mIoU (%) | nuScenes mIoU (%) | Params (M) | FPS (Hz) | GPU | FP32 TFLOPS | Norm. FPS |
|---|---|---|---|---|---|---|---|---|
| Cylinder3D [29] | 2020 | 68.9 | 77.9 | 55.9 | 13.2 | RTX 2080 Ti | 13.45 | 0.93 |
| SPVCNN [21] | 2020 | 63.8 | – | 21.8 | 3.2 | GTX 1080 Ti | 11.34 | 0.28 |
| SPVNAS [21] | 2020 | 66.4 | 77.0 | 12.5 | 3.9 | GTX 1080 Ti | 11.34 | 0.35 |
| FRNet [27] | 2022 | 73.3 | 82.5 | 10.0 | 29.1 | RTX 2080 Ti | 13.45 | **2.05** |
| CENet [5] | 2022 | 64.7 | – | 3.1 | 37.8 | RTX 3090 | 35.58 | 1.06 |
| PTV2 [24] | 2022 | 72.6 | 82.6 | 12.8 | 6.9 | RTX 4090 | 82.58 | 0.08 |
| PTV3 [23] | 2023 | 74.2 | 82.7 | 46.2 | 22.7 | RTX 4090 | 82.58 | 0.27 |
| PTV3+PPT [23, 25] | 2023 | **75.5** | **83.0** | 46.2 | 22.7 | RTX 4090 | 82.58 | 0.27 |
| SphereFormer [11] | 2023 | 74.8 | 81.9 | 32.3 | 18.7 | RTX 3090 | 35.58 | 0.53 |

roofs, capturing automotive driving environments in Germany and the USA/Singapore, respectively. Consequently, their raw scans differ greatly from SenseBike's data in *sensor type and placement*, *viewing angle*, *field-of-view*, and *scene content*. To evaluate the need and benefit of in-domain data for robust LiDAR segmentation on the SenseBike, we created **BikeScenes LidarSeg**. This dataset, captured from the SenseBike's perspective, retains the SemanticKITTI class categorization to allow direct comparison.

## A. Recording

A single RoboSense M1 Plus LiDAR mounted on the front carrier of the bike was used to record a 1.3 km closed loop at 10 Hz, yielding 3021 consecutively time-stamped scans around the Delft University of Technology campus. As described in Subsection III-B, the scans were deskewed online. Additionally, images from the front-facing camera were captured at 10 Hz to aid in the labeling procedure. Table II compares the statistics of BikeScenes to other outdoor LiDAR datasets with semantic labels.

The LiDAR frames were registered offline with GLIM [9]. After the automatic alignment, we added a single manual loop-closure between the first and last scans in the GUI. GLIM then refined the whole trajectory by combining scan-to-scan matching with IMU priors. It produced a smooth and consistent sequence of poses from which we built the final aggregated pointcloud map used for multi-scan labeling. More on this procedure can be found in Appendix III.

## B. Labeling Procedure

We labeled the sequence using the SemanticKITTI Labeling Tool, preserving the original 28 classes (22 unique + 6 moving/static versions) and adding a *bike-path* class. Static objects were labeled in the aggregated map, while dynamic objects (pedestrians, cyclists, cars) were labeled *per scan* to ensure the highest data quality for these sparse and fine-scale classes, whose structure is represented by only a small number of points. The labeled map consisting of all the scans combined is shown in Figure 24. The class distribution of BikeScenes is provided in Figure 5.

In some scans of the RobosSense M1 Plus sensor, the effects of *ghosting* can be observed. This phenomenon occurs when highly reflective surfaces mirror the objects in front of them, making them appear as real objects behind them. As these *ghost points* did not directly come from real objects, they are labeled as outliers. Figure 23 in Appendix III shows this phenomenon.

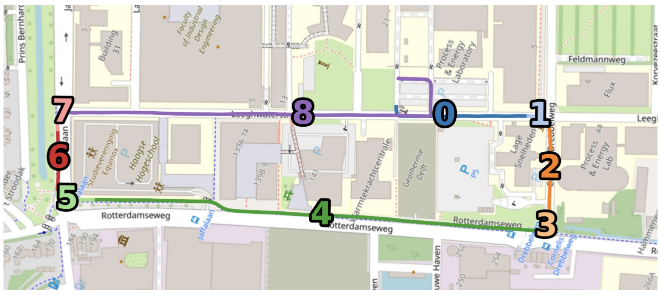


Fig. 6: GPS trajectory of the dataset recording and the subsequence categorization.

## C. Subsequences - Train/Val/Test Split

We created a train, validation, and test set by dividing the sequence into 9 subsequences, as can be seen in Figure 6. For our train set, we use the long consecutive segments (0, 4, 8), and for our test set, the two shorter straight segments (2, 6). The corners (1, 3, 5, 7) are used as the validation set for the evaluation of

TABLE II: Comparison of outdoor LiDAR semantic-segmentation datasets.

| **Dataset** | **Platform** | **Countries** | **Sequences** | **Ann. Frames** | **Classes** | **LiDAR** |
|---|---|---|---|---|---|---|
| SemanticKITTI [1] | Car | Germany | 22 | 43,552 | 28 | Velodyne HDL-64E |
| nuScenes [4] | Car | USA, Singapore | 1,000 | 40,000 | 32 | Velodyne HDL-32E |
| Waymo [20] | Car | USA | 1,150 | 230,000 | 23 | Undisclosed 64-beam rotating |
| SemanticPOSS [15] | Car | China | 1 | 2,988 | 14 | Hesai Pandora |
| PandaSet [26] | Car | USA | 30 | 6,080 | 37 | Pandar64 + PandarGT |
| SBLD [13] | Bicycle | Austria | 17 | 9,486 | 10 | 5 x Livox Horizon |
| **BikeScenes** | Bicycle | the Netherlands | 1 | 3,021 | 29 | RoboSense M1 Plus |

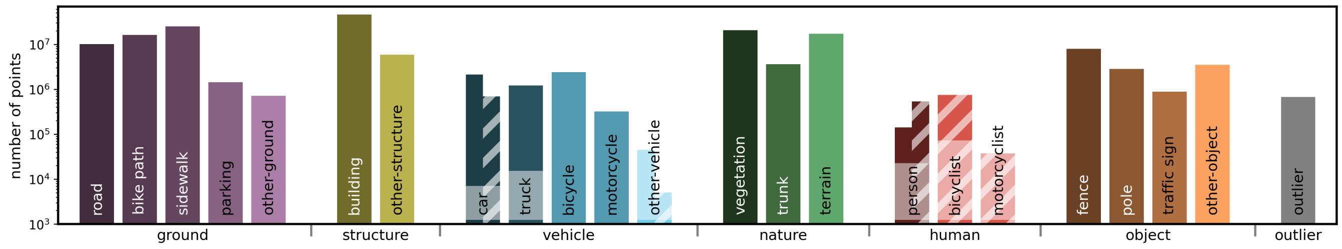


Fig. 5: Class distribution of the BikeScenes dataset. The number of points for dynamic classes is divided between non-moving (solid bars) and moving objects (hatched bars).

our training strategies. An overview of the number of points per class for the train, validation, and test set is provided in Table VI in Appendix III.

## V. EXPERIMENTS AND RESULTS

### A. Class Remapping

For the following experiments, the 29 classes of the BikeScenes dataset are remapped to the 19 SemanticKITTI evaluation classes, additionally merging the *bike-path* class into *road*. This alignment enables comparison across domains. Unless otherwise specified, all training and testing use this 19-class subset.

### B. Evaluation Metrics

For our experiments, we use the standard intersection-over-union (IoU) metric for semantic segmentation to evaluate the performance of the different models and configurations quantitatively.

For a given class $i$, IoU is defined as:

$$IoU_i = \frac{TP_i}{TP_i + FP_i + FN_i}$$

Where $TP_i$ and $FP_i$ are the true and false positives, respectively, and $FN_i$ the false negatives for class $i$. The mean over all the evaluated classes is often calculated and presented as $mIoU$.

### C. Domain Gap Analysis and Initial Training Strategy

This initial experiment has two primary objectives: first, to evaluate the domain gap between the SemanticKITTI and the BikeScenes dataset, and second, to compare the effectiveness of training a model on BikeScenes *from scratch* (i.e. with random weight initialization) versus *fine-tuning* existing weights pre-trained on SemanticKITTI. To obtain these insights, we evaluate the performance of the FRNet model on the BikeScenes validation set under three conditions:

- **Exp. 1a:** Pre-trained (KITTI), no further training.
- **Exp. 1b:** From scratch, LR 0.01, 50k iterations.
- **Exp. 1c:** Fine-tuning, LR 0.01, 50k iterations.

For strategies **1b** and **1c**, we adopted the reported training configuration by the authors of FRNet for their SemanticKITTI training: 50k iterations, a batch size of 2, and the One-Cycle learning rate scheduler [19] with a peak learning rate (LR) of 0.01. The One-Cycle policy varies the LR between a lower and upper bound, starting low, climbing to the peak, then gradually decaying. The brief large-LR phase lets the weights take larger update steps and evaluate a broader range of parameters before settling, typically yielding faster convergence (flattening of the loss) and better generalization.

As can be observed in Table III, directly testing the SemanticKITTI pre-trained model on the BikeScenes validation set (**Exp. 1a**) yielded a low mIoU of 12.2%, indicating a significant domain gap. Both training strategies on BikeScenes data substantially improved performance. Training from scratch for 50k iterations (**Exp. 1b**) achieved an mIoU of 53.3%. Fine-tuning the pre-trained model for the same duration and LR (**Exp. 1c**) resulted in a higher mIoU of 55.6%, suggesting that leveraging features learned from the larger SemanticKITTI dataset provides a beneficial starting point. However, during the fine-tuning process of **Exp. 1c**, we observed that the training loss briefly increased, suggesting that the 0.01 peak LR, while useful for exploration for random weight initialization, may be too high once the weights are near a good solution. This observation motivated the subsequent experiment to investigate a lower learning rate.

## *D. Learning Rate*

Based on the brief training instability observed with an LR of 0.01 during fine-tuning (**Exp. 1c**), this experiment tests whether a much lower peak LR of $3{\times}10^{-4}$ improves performance and training stability for both from-scratch and fine-tuning strategies. The loss graph showed that it was decreasing while the LR was below roughly $10^{-3}$, but started increasing as the LR climbed toward 0.01, motivating the choice of this smaller value. The experimental setups are identical to **Exp. 1b** and **Exp. 1c** respectively, with the only difference being the reduction in peak learning rate:

- **Exp. 2a:** From scratch, LR $3{\times}10^{-4}$, 50k iterations.
- **Exp. 2b:** Fine-tuning, LR $3{\times}10^{-4}$, 50k iterations.

As can be observed by comparing the results in Table III, the fine-tuning strategy benefited from the reduction in learning rate, achieving an mIoU of 57.8% compared to 55.6% with the higher LR (**Exp. 1c**). The lower learning rate also resulted in a more stable loss curve during fine-tuning, as shown in Figure 7. Conversely, for the training from scratch strategy, the lower learning rate had an adverse effect on performance, with an mIoU of 52.7%, which is lower compared to the mIoU of 53.3% in **Exp. 1b**.

## *E. Impact of Training Iterations*

This experiment investigates the impact of fewer training iterations using the more promising lower learning rate of ($3{\times}10^{-4}$) for both from-scratch training and fine-tuning. The aim is to identify an optimal training length and observe if shorter training durations can achieve comparable or better results, potentially avoiding overfitting or saving computational resources. We repeat the training configurations from experiment 2, but for 10k and 30k iterations:

- **Exp. 3a:** From scratch, LR $3{\times}10^{-4}$, 10k iterations.
- **Exp. 3b:** Fine-tuning, LR $3{\times}10^{-4}$, 10k iterations.
- **Exp. 3c:** From scratch, LR $3{\times}10^{-4}$, 30k iterations.
- **Exp. 3d:** Fine-tuning, LR $3{\times}10^{-4}$, 30k iterations.

The results in Table III indicate the following trends:
For the fine-tuning strategy with $3{\times}10^{-4}$:

- 10k iterations (**Exp. 3b**) achieved 55.9% mIoU.
- 30k iterations (**Exp. 3d**) achieved 58.6% mIoU.
- 50k iterations (**Exp. 2b**) achieved 57.8% mIoU.

For the training from scratch strategy with $3{\times}10^{-4}$:

- 10k iterations (**Exp. 3a**) achieved 46.2% mIoU.
- 30k iterations (**Exp. 3c**) achieved 51.5% mIoU.
- 50k iterations (**Exp. 2a**) achieved 52.7% mIoU.

Training for 30k iterations yielded the highest mIoU for this learning rate for fine-tuning, **Exp. 3d** outperformed shorter training and slightly exceeded the performance at 50k iterations, suggesting 30k is a better duration. For from-scratch training with the lower LR, 50k iterations (**Exp. 2a**, 52.7% mIoU) performed best on the validation set, though 30k iterations (Exp. 3c, 51.5% mIoU) was also an improvement over 10k.

Based on these results, the best overall configuration identified from Experiments 1-3 on the validation set is fine-tuning the pre-trained FRNet model for 30k iterations with a peak learning rate of $3{\times}10^{-4}$ (**Exp. 3d**), achieving an mIoU of 58.6%. This optimized configuration is used in later analyses, unless stated otherwise.

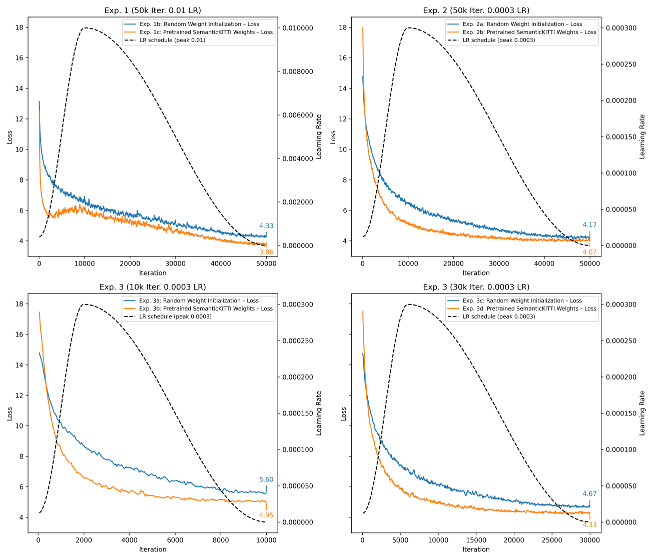


Fig. 7: The loss and the learning rate over the iterations of the training cycles of our experiments.

TABLE III: Class-wise IoU (%) on the BikeScenes validation set (19-class mapping). Best scores are in **bold**. A dash "–" marks classes absent in the validation set. mIoU is averaged over the 17 present classes.

| **Exp.** | **Strategy** | **LR** | **Iter.** | mIoU (%) | Car | Bicycle | Motorcycle | Truck | Other-Veh. | Person | Bicyclist | Motorcyclist | Road | Parking | Sidewalk | Other-Gnd. | Building | Fence | Vegetation | Trunk | Terrain | Pole | Traffic-Sign |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1a | Pre-trained | – | – | 12.2 | 3.6 | 0.0 | 0.0 | – | – | 0.1 | 0.0 | 0.0 | 26.5 | 0.5 | 1.6 | 0.0 | 43.2 | 1.9 | 27.0 | 2.3 | 21.7 | 35.1 | 43.5 |
| 1b | Scratch | 0.01 | 50k | 53.3 | 32.6 | **85.1** | 30.2 | – | – | 68.0 | 59.1 | 0.0 | 86.6 | **5.4** | 74.6 | 1.9 | 86.7 | 38.9 | 71.6 | 55.3 | 67.8 | 66.9 | 76.5 |
| 1c | Fine-Tune | 0.01 | 50k | 55.6 | 30.8 | 84.2 | **35.2** | – | – | 74.7 | 63.2 | 0.0 | 86.8 | 3.7 | 74.1 | 2.6 | 89.2 | 42.9 | 76.6 | 60.1 | **71.8** | 70.9 | 78.4 |
| 2a | Scratch | $3\times10^{-4}$ | 50k | 52.7 | 32.9 | 83.6 | 29.7 | – | – | 63.4 | 61.4 | **0.1** | 85.7 | 2.6 | 71.7 | 3.0 | 85.5 | 41.3 | 70.0 | 56.1 | 66.5 | 67.0 | 75.6 |
| 2b | Fine-Tune | $3\times10^{-4}$ | 50k | 57.8 | 60.7 | 83.5 | 22.2 | – | – | 74.9 | **65.5** | 0.0 | 87.3 | 4.4 | 73.1 | **5.6** | **92.1** | 48.0 | **77.9** | 66.5 | 69.9 | 71.8 | 79.0 |
| 3a | Scratch | $3\times10^{-4}$ | 10k | 46.2 | 16.4 | 73.2 | 22.5 | – | – | 57.9 | 54.6 | 0.0 | 75.6 | 1.1 | 66.2 | 0.2 | 81.6 | 27.6 | 63.4 | 47.9 | 63.9 | 60.4 | 73.2 |
| 3b | Fine-Tune | $3\times10^{-4}$ | 10k | 55.9 | 47.8 | 75.4 | 21.9 | – | – | **77.3** | 63.6 | 0.0 | 84.7 | 2.1 | 75.0 | 0.6 | 90.9 | 47.5 | 74.8 | 70.6 | 66.8 | 70.0 | **81.1** |
| 3c | Scratch | $3\times10^{-4}$ | 30k | 51.5 | 33.2 | 81.2 | 24.7 | – | – | 62.2 | 61.0 | 0.0 | 84.4 | 2.3 | 72.3 | 1.3 | 85.4 | 35.3 | 68.3 | 55.5 | 66.1 | 67.2 | 75.2 |
| 3d | Fine-Tune | $3\times10^{-4}$ | 30k | **58.6** | **65.1** | 80.2 | 21.6 | – | – | **77.3** | 65.2 | 0.0 | **88.7** | 5.0 | **75.3** | 4.6 | **92.1** | **49.1** | 77.5 | **70.9** | 71.1 | **72.0** | 80.5 |

### *F. Performance on the Test Set*

The best-performing model on the validation set is tested on the test set. Table IV compares the performance between the model that was pre-trained on the SemanticKITTI dataset versus the fine-tuned model from **Exp. 3d**. The fine-tuned model achieves an mIoU of 63.6%, which is significantly higher than the pre-trained model's performance of 13.8%.

### *G. Classification of the Bike-Path*

We conduct an initial exploration of the model's ability to classify dedicated bike paths, motivated by their distinct visual appearance (burgundy color) often found in the Netherlands. To investigate this, we conduct **Exp. 4**, where *bike-path* is treated as a separate semantic class instead of being merged with *road* (as done in the standard 19-class mapping). This preliminary study was conducted before the hyperparameter optimization detailed in Experiments 2 and 3. It utilized the FRNet model trained from scratch on the BikeScenes dataset for 50k iterations with a peak LR of 0.01 (consistent with the setup of **Exp. 1b**). This experiment yielded an overall mIoU of 50.2% across the 18 present classes in the validation set and an IoU of 51.6% and 51.4% on the *bike-path* and *road* classes, respectively. Table IX in Appendix V provides the segmentation performance for all classes evaluated in this experiment.

### *H. ROS2 Implementation and the Impact of Automatic Mixed Precision*

The goal of this experiment is to evaluate the impact of Automatic Mixed Precision (AMP) on both the segmentation accuracy (mIoU) and the processing speed (Hz) of our best-performing model configuration (Exp. 3d from Table III: Fine-tune, LR $3\times10^{-4}$, 30k iterations) when deployed in the `FRNetROSInferenceNode` during an online streaming scenario.

The FRNet model with weights obtained from Experiment 3d (Fine-tune, LR $3\times10^{-4}$, 30k iterations) was used. This model is integrated into our custom `FRNetROSInferenceNode` for on-bike deployment. We compared two configurations:

1) **FP32:** Standard full-precision (32-bit floating point) for all model weights and activations.
2) **AMP:** Automatic Mixed Precision enabled within the ROS2 node, utilizing PyTorch's `autocast` functionality to dynamically employ half-precision (FP16) for suitable operations.

We assess segmentation performance using the BikeScenes test set by comparing mIoU and per-class IoU scores. Processing speed was benchmarked over 1000 LiDAR scan samples from an online stream from the RoboSense M1 Plus sensor. Mean times for critical pipeline stages within the ROS2 node—data conversion, model inference, and post-processing—were recorded.

The segmentation performance comparison between FP32 and AMP is presented in Table VII. Our findings indicate that enabling AMP results in a negligible difference in segmentation performance compared to the FP32 implementation. The mIoU remained at **63.6%**, with only minor per-class IoU variations, typically within $\pm0.1$ percentage points for a few classes.

Regarding processing speed, Table VIII shows that the AMP implementation demonstrated a significant improvement. The total average processing time per scan was reduced from 1029.7 ms (FP32) to 766.3

TABLE IV: Class-wise IoU (%) on the BikeScenes test set (19-class mapping). A dash "–" marks classes absent in the test set. mIoU is averaged over the 16 present classes.

| **Exp.** | **Strategy** | **LR** | **Iter.** | mIoU | Car | Bicycle | Motorcycle | Truck | Other-Veh. | Person | Bicyclist | Motorcyclist | Road | Parking | Sidewalk | Other-Gnd. | Building | Fence | Vegetation | Trunk | Terrain | Pole | Traffic-Sign |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1a | Pre-trained | – | – | 13.8 | 12.3 | 0.0 | 0.0 | – | – | 0.0 | 0.0 | – | 44.1 | 0.6 | 1.5 | 0.5 | 45.1 | 1.4 | 24.7 | 1.4 | 26.1 | 29.7 | 33.0 |
| 3d | Fine-tune | $3\times10^{-4}$ | 30k | 63.6 | 78.9 | 26.9 | 32.4 | – | – | 83.2 | 87.5 | – | 87.2 | 3.4 | 83.4 | 2.6 | 91.6 | 79.4 | 82.5 | 61.6 | 67.2 | 72.0 | 78.5 |

ms (AMP). This corresponds to an increase in overall throughput from 0.97 FPS for the FP32 implementation to **1.31 FPS** with AMP, representing a **35% speed-up**. The model inference component itself averaged 607.9 ms per scan with AMP.

The results demonstrate that AMP provides a substantial processing speed advantage for our ROS2 pipeline without having a notable loss in segmentation accuracy. This makes AMP a highly beneficial optimization for on-bike deployment where computational resources are constrained and higher throughput is desirable. While the achieved 1.31 FPS with AMP is not yet at conventional real-time levels for dynamic perception tasks, the 35% improvement is a step towards this goal.

### *I. Qualitative Results*

*1) Performance on Test Set Examples:* Figure 8 compares the segmentation output of the baseline SemanticKITTI pre-trained model (**Exp. 1a**) with the best fine-tuned model (**Exp. 3d**), alongside the ground truth, for two distinct scenes from the BikeScenes test set. The columns represent, from left to right: the pre-trained model, our fine-tuned model, and the ground truth.

The output from the pre-trained model (**Exp. 1a**, left column in both scenes) starkly illustrates the impact of the domain gap. Its predictions show almost no coherent labeling of the environment, with only occasional, fragmented correct predictions for some *road* and *traffic-sign* points. Most of the scene is misclassified, failing to capture the semantic structure.

In sharp contrast, the fine-tuned model (Exp. 3d, middle column) demonstrates near-perfect segmentation of these scenes, closely matching the ground truth (right column). Common classes such as *road*, *building*, *vegetation*, *car*, and *bicyclist* are accurately labeled with precise boundaries. The only consistent discrepancy observed in these scenes involves ghost points (reflections), visible as black outliers in the ground truth. Because these outliers were excluded from the training labels, the model has no *outlier* category to assign. Instead, it consistently labels these points as *building* when they lie within building volumes.

*2) Generalization Assessment on Unlabeled Data:* To evaluate the generalization capability of our best fine-tuned model from **Exp. 3d** further, we qualitatively assessed its predictions on unlabeled data. This data was collected on a public road approximately 600 meters away from where the BikeScenes training data was captured, representing a novel environment for the model. Our observations are as follows:

**Static and Common Dynamic Objects:** The model generally exhibits strong performance on common static object classes such as *trunk*, *building*, *pole*, and *traffic-sign*. For dynamic objects, points belonging to the *car* class are also regularly predicted with high accuracy. However, challenges arise with less common or oddly shaped vehicles; for instance, buses (which should be labeled as *other-vehicle*) were observed to be partially misclassified as *car* and partially as *building*.

**Bicyclists**: The model's performance on bicyclists appears sensitive to viewing angle and orientation. Cyclists positioned directly in front of the sensor (in the same or opposite travel direction) are often correctly labeled. However, when cyclists are tangential to the sensor's view or oriented at 90 degrees, misclassifications are more frequent. An example is illustrated in Figure 9, where a bicyclist is captured from the side. In this instance, the bicycle frame is labeled as *bicycle* and the rider as *bicyclist*, while, following the SemanticKITTI and BikeScenes labeling convention, the entire entity should have been labeled as *bicyclist*. This specific misclassification pattern (separating rider and bicycle) for side views might stem from biases in the training data, where most bicyclist instances are captured moving along the same path as our LiDAR sensor, and most *bicycle* instances (without a rider) are parked and thus often viewed from the side.

**Ground Classes:** In many cases, the model appears

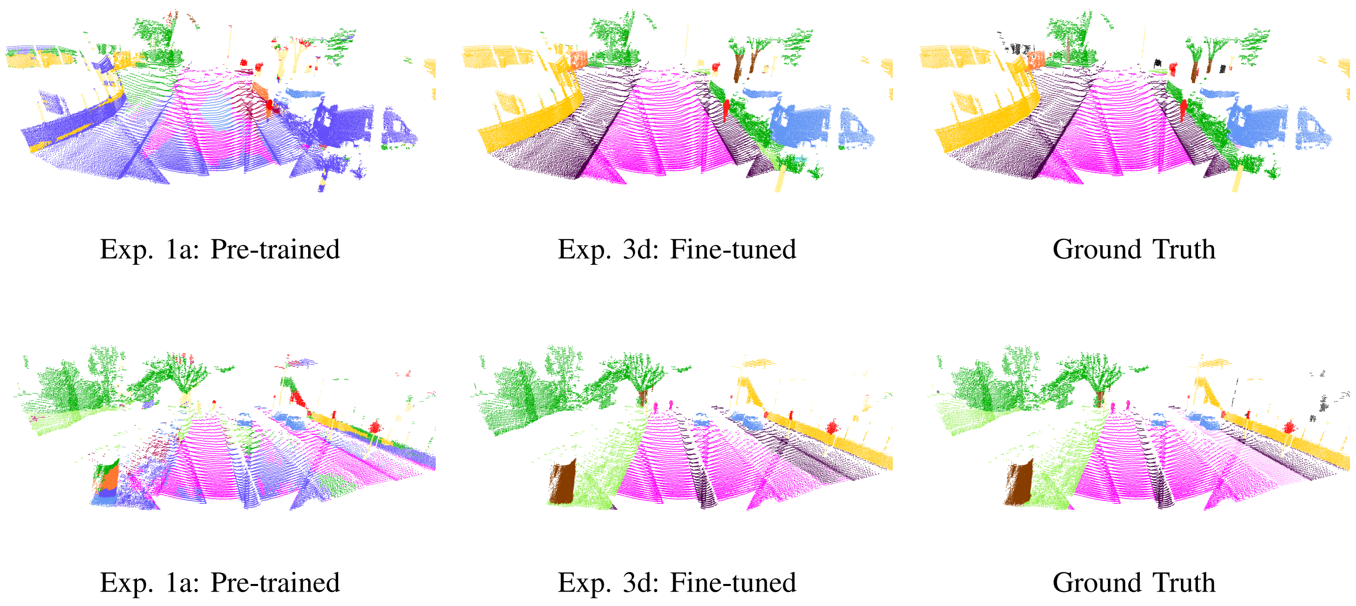


Fig. 8: Labels predicted by the baseline SemanticKITTI pre-trained model (left) vs. the best performing fine-tuned model (middle) on two scenes of the test set of BikeScenes. Legend: 🟨 building; 🟪 road; ⬛ sidewalk; 🟩 vegetation; 🟩 terrain; 🟫 trunk; 🟦 car; 🟥 person; 🟥 traffic-sign; 🟪 bicyclist;

to be struggling with fine-grained distinctions between ground classes, notably separating *sidewalk* from *road* and *terrain* from *vegetation*. These classes often share similar characteristics. The difficulty in distinguishing *terrain* from *vegetation* may also be compounded by the SemanticKITTI labeling convention (which BikeScenes follows), where low-cut grass is labeled as *terrain* while taller plants are labeled as *vegetation*. This distinction might be too subtle for LiDAR-only perception. We also find that the ground beneath cars is frequently predicted as `parking`, even when the vehicle is driving on the road, indicating that the model treats the presence of a car as a strong contextual cue for the *parking* class.

## VI. DISCUSSION

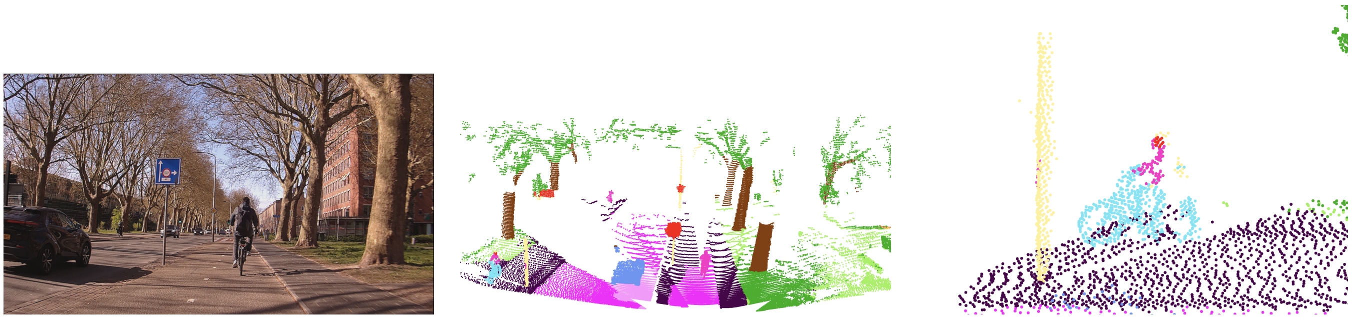
### A. *Key Findings*

The experiments evaluated FRNet [27] training configurations on the BikeScenes dataset to determine a suitable model for onboard deployment. The results in Tables III and IV yield several key findings.

**Necessity of Domain Adaptation:** Training explicitly on the BikeScenes dataset is crucial. The SemanticKITTI pre-trained model (**Exp. 1a**) achieved only 12.2% mIoU when evaluated directly on the BikeScenes validation set. In contrast, all models trained or fine-tuned on BikeScenes equaled or surpassed 46.2% mIoU, demonstrating a significant performance increase of over 34 percentage points. This highlights a substantial domain gap, likely stemming from differences in sensor type, mounting height (car vs. bicycle), field of view, and environmental characteristics, which needs to be overcome with own-domain data.

**Effectiveness of Fine-tuning:** Fine-tuning the SemanticKITTI pre-trained weights consistently outperformed training from scratch across all comparable schedules (**Exp. 1c** vs. **1b**, **Exp. 2b** vs. **2a**, **Exp. 3b** vs. **3a**, **Exp. 3d** vs. **3c**). For instance, at 30k iterations with low learning rate, fine-tuning (**Exp. 3d**) achieved 58.6% mIoU compared to 51.5% from scratch (**Exp. 3c**). This indicates that the features learned on SemanticKITTI provide a valuable initialization and transfer to some degree, despite the domain differences.

**Learning Rate Schedule:** The initial training configuration (**Exp. 1b** & **1c**, which used the peak learning rate ($LR_{max} = 0.01$), as was done by the authors of FRNet for training on SemanticKITTI, served as an important baseline. While fine-tuning with this LR (**Exp. 1c**) achieved an mIoU of 55.6% on the validation set (Table III), it also showed less stable training dynamics in the form of a brief increase in total loss during training (Fig. 7). This suggested that the learning



(a) Camera image. (b) Predicted labels on LiDAR scan. (c) Clos-up side view of the bicyclist.

Fig. 9: Predicted labels of the selected model on unlabeled data. Legend: 🟨 building; 🟪 road; 🟪 sidewalk; 🟩 vegetation; 🟩 terrain; 🟫 trunk; 🟦 car; 🟥 person; 🟥 traffic-sign; 🟪 bicyclist; 🟦 bicycle; 🟪 parking; 🟨 pole;

rate might be too high for optimal adaptation to our smaller BikeScenes dataset.

Consequently, we investigated a significantly lower peak learning rate of $LR_{max} = 3{\times}10^{-4}$ (**Exp. 2a** & **2b** in Table III). For the more effective fine-tuning strategy, this reduction in learning rate led to more stable training loss convergence (Fig. 7) and an improved mIoU of 57.8% (**Exp. 2b**) on the validation set when training for 50k iterations. This confirmed the benefit of a smaller learning rate for fine-tuning on our dataset. For training from scratch, however, the higher LR of 0.01 (Exp. 1b: 53.3% mIoU) remained slightly better than the lower LR (**Exp. 2a**: 52.7% mIoU) at 50k iterations on the validation set.

**Iteration Count:** Using the more promising lower learning rate of $LR_{max} = 3{\times}10^{-4}$ for fine-tuning, we then explored different training durations on the validation set. Extending training from 10k iterations (**Exp. 3b**, 55.9% mIoU) to 30k iterations (**Exp. 3d**, 58.6% mIoU) further improved performance, and showed to be better than 50k iterations (**Exp. 2b**, 57.8% mIoU). For from-scratch training with the lower LR, more iterations resulted in better performance. This pattern might suggest that starting with pre-trained weights could lead to faster overfitting on the training data.

**Final Performance of Selected Model on the Test Set:** As shown in Table IV, the model from **Exp. 3d** achieved an mIoU of 63.6% on the BikeScenes test set. This result represents a substantial improvement of nearly 50 percentage points over the baseline pre-trained model (**Exp. 1a**), which achieved only 13.8% mIoU on the same test set, underscoring the benefits of the developed domain adaptation strategy using the BikeScenes dataset.

On this test set, the model from **Exp. 3d** demonstrated strong performance across several key classes essential for bicycle perception. It achieved high IoU scores for static environment classes such as *Road* (87.2%), *Sidewalk* (83.4%), *Building* (91.6%), and *Vegetation* (82.5%). Performance on dynamic classes was also strong, with *Person* at 83.2% IoU and *Bicyclist* at 87.5% IoU. While its IoU for some classes like *Bicycle* (26.9%), *Motorcycle* (32.4%) and *Parking* (3.4%) remained low.

**Confusion Matrix Analysis:** The row-normalized confusion matrix for the model from Exp. 3d (Figure 10) provides further insights into the performance on the test set. The model demonstrates high recall (diagonal values $> 0.80$) for most static classes. However, the matrix also highlights significant error patterns. Notably, *Car* and *Motorcycle* points are frequently misclassified as *Building*. It also shows the model's limited ability to distinguish *Parking* from *Road* or *Sidewalk*. Surprisingly, the model does not show significant confusion between *Person* and *Bicyclist*. We also observe that the model often predicted outlier points to be of the classes *building* (0.53) and *vegetation* (0.34). These are most likely *"ghost points"* that emerge when highly reflective surfaces mirror the objects. Often, this phenomenon occurs when trees are in front of buildings with large glass facades. As these points were not derived from real objects directly, they were labeled as outliers.

**Inference Speed:** When optimized with Automatic Mixed Precision (AMP), the `FRNetROSInferenceNode` processes LiDAR scans at an average overall throughput of 1.31 Hz, with the model inference stage itself operating at approximately 1.65 Hz (607.9 ms per scan). This AMP configuration represented a 35% speed-up in total processing time compared to the FP32 implementation within our ROS2 pipeline.

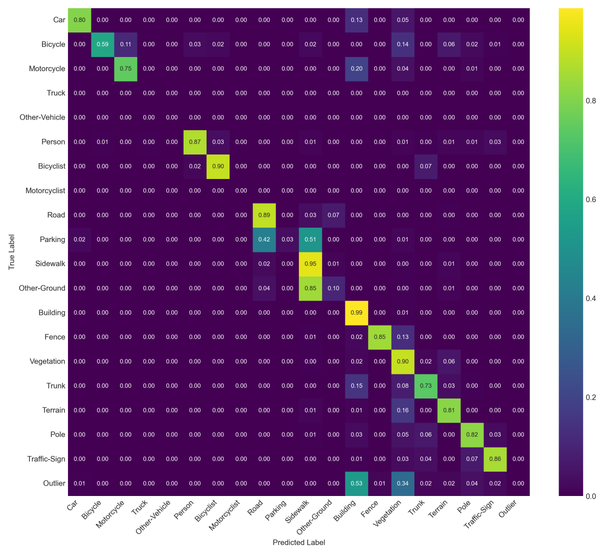


Fig. 10: Normalized confusion matrix for Exp. 3d (Fine-Tune-30k, $LR = 3{\times}10^{-4}$) on the BikeScenes test set (Seq. 02 & 06).

However, a notable performance gap exists when comparing these figures to offline benchmarks. Using the same model and AMP, the MMDetection3D [6] `benchmark.py` script achieved approximately 2.9 Hz for pure model inference. This benchmark performance decreased slightly to 2.7 Hz when other nodes of our ROS2 system (excluding the inferencer itself) were active, indicating minor overhead from the general ROS2 environment. The primary discrepancy, therefore, lies between our ROS2 node's inference component (1.65 Hz) and the more optimized benchmark script (2.7-2.9 Hz). This remaining performance difference is likely attributable to the data loader, which processes saved LiDAR scans faster than incoming scans from an online pipeline.

While the use of AMP significantly enhanced the operational speed of our segmentation pipeline, the 1.31 Hz is likely not yet sufficient for safety-critical downstream tasks.

## B. Limitations and Future Work

**Dataset Partitioning and Model Validation:** A challenge in this study was the establishment of appropriate data splits for training, validation, and testing from our newly created BikeScenes dataset. Splitting a single, continuous sequence of consecutive LiDAR scans into disjoint segments can still present subtle data leakage challenges. This is because samples chronologically or spatially close to the separation between sets often share high amounts of similarity in environmental structure, objects, and geometries. To minimize these effects for the final test evaluation, we strategically used the long straight segments (0, 4, 8) for training and the shorter straight segments (2, 6) for testing, maximizing their separation. Our validation set (the corner sequences 1, 3, 5, 7), while geographically separating some train and test segments, is adjacent to both. It is therefore more affected by this phenomenon of shared characteristics, meaning its independence from the training data, while formally maintained by using disjoint scans, might be less absolute than if it were from an entirely separate recording session. Consequently, model selection guided by this validation set, could carry some risk of optimistic bias for eventual generalization.

The single-loop structure of the BikeScenes dataset, though effective for achieving good scan-to-scan alignment with manual loop closure and for labeling of scans at bulk, also inherently complicates the creation of truly independent and diverse data splits compared to a dataset comprised of multiple, shorter, distinct recordings.

Addressing these data-related limitations is a priority for future work. Establishing a more robust train/validation/test splitting methodology is crucial. This will likely involve acquiring new, distinct data sequences specifically for the validation and test sets, ideally captured from diverse locations and conditions. Such an approach will enable more objective model selection, and provide less biased estimates of generalization performance. Future data collection should therefore focus on obtaining multiple, completely disjoint sequences.

**Inference Speed Optimization:** While Automatic Mixed Precision yielded a significant 35% improvement, the achieved 1.31 FPS overall throughput for the ROS2 pipeline is not yet sufficient for reliable real-time performance in many safety-critical dynamic scenarios encountered by cyclists. Therefore, an important direction for future research is improving inference speed to meet practical on-bike deployment requirements. This will involve implementing advanced model optimization techniques, including network quantization (e.g., to INT8 precision), compilation using inference accelerators like NVIDIA TensorRT, and potentially exploring more lightweight model architectures or pruning techniques.

## VII. CONCLUSION

Our work addressed the challenge of training and deploying 3D LiDAR semantic segmentation on a bicycle platform, motivated by the potential to enhance cyclist safety through improved environmental awareness. We presented a complete pipeline implemented on the SenseBike research platform, encompassing sensor calibration, real-time ego-motion compensation for LiDAR scan deskewing, and the creation of the novel BikeScenes dataset specifically captured from a bicycle's perspective.

The experiments demonstrated the critical need for domain-specific data. Fine-tuning the efficient FRNet model on our BikeScenes dataset yielded a substantial performance increase, achieving 63.6% mIoU, compared to 13.8% obtained when using only SemanticKITTI pre-trained weights. Notably, this fine-tuning approach proved more effective than training the model from scratch using only BikeScenes data, underscoring the value of transferring learned features despite a significant domain shift. Our best fine-tuned model demonstrated high accuracy on common classes critical to the bicycle environment, such as roads, buildings, persons, and bicyclists.

This research's primary contributions are the demonstration of a feasible pipeline for bicycle-mounted LiDAR segmentation, the release of the BikeScenes dataset to facilitate further research in this domain, and a quantitative evaluation of a segmentation model adapted to the unique challenges of bicycle-based and hardware-constrained perception.

## REFERENCES

[1] 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: *ICCV*. 2019. 2, 3, 5.

[2] Boréal Bikes. *Connected Micromobility – AI ADAS – V2X*. https://www.borealbikes.com/. 2024. 1.

[3] James Bowman and Patrick Mihelich. *camera_calibration*. https://wiki.ros.org/camera_calibration. 14.

[4] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. "nuScenes: A multimodal dataset for autonomous driving". In: *CVPR*. 2020. 2, 3, 5.

[5] Hui–Xian Cheng, Xian–Feng Han, and Guo–Qiang Xiao. "Cenet: Toward Concise and Efficient Lidar Semantic Segmentation for Autonomous Driving". In: *IEEE International Conference on Multimedia and Expo*. 2022. 2, 4.

[6] MMDetection3D Contributors. *MMDetection3D: OpenMMLab next-generation platform for general 3D object detection*. https://github.com/open-mmlab/mmdetection3d. 2020. 11.

[7] Dustin Franklin. *ros_deep_learning: Deep learning inference nodes for ROS using TensorRT*. https://github.com/dusty-nv/ros_deep_learning. Accessed: 2025-04-22. 2024. 14.

[8] Yuenan Hou, Xinge Zhu, Yuexin Ma, Chen Change Loy, and Yikang Li. "Point-to-Voxel Knowledge Distillation for LiDAR Semantic Segmentation". In: *CVPR*. 2022. 2.

[9] Kenji Koide, Masashi Yokozuka, Shuji Oishi, and Atsuhiko Banno. "GLIM: 3D range-inertial localization and mapping with GPU-accelerated scan matching factors". In: *Robotics and Autonomous Systems* (2024). 4.

[10] CCNY Robotics Lab. *imu_tools: ROS package for IMU processing*. https://github.com/CCNYRoboticsLab/imu_tools. Accessed: 2025-04-23. 2023. 16.

[11] Xin Lai, Yukang Chen, Fanbin Lu, Jianhui Liu, and Jiaya Jia. "Spherical Transformer for LiDAR-based 3D Recognition". In: *CVPR*. 2023. 2, 4.

[12] T. Moore and D. Stouch. “A Generalized Extended Kalman Filter Implementation for the Robot Operating System”. In: *Proceedings of the 13th International Conference on Intelligent Autonomous Systems (IAS-13)*. Springer, July 2014. 2, 17.

[13] Armin Niedermüller. *Salzburg Bicycle LiDAR Data Set*. 2023. 2, 5.

[14] NVIDIA Corporation. *NVIDIA Ampere GA10x GPU Architecture Whitepaper*. https://www.nvidia.com/en-us/geforce/news/rtx-30-series-ampere-architecture-whitepaper-download/. 2020. 19.

[15] Yancheng Pan, Biao Gao, Jilin Mei, Sibo Geng, Chengkun Li, and Huijing Zhao. “Semanticposs: A point cloud dataset with large quantity of dynamic instances”. In: *IEEE Intelligent Vehicles Symposium*. 2020. 5.

[16] Polycam. *Polycam - 3D Scanner App*. https://poly.cam. https://poly.cam. 2024. 2.

[17] PyTorch Team. *Automatic Mixed Precision Package — PyTorch Documentation*. https://pytorch.org/docs/stable/amp.html. 2025. 21.

[18] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. “Pointnet: Deep learning on point sets for 3d classification and segmentation”. In: *CVPR*. 2017. 2.

[19] Leslie N. Smith and Nicholay Topin. *Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates*. 2018. arXiv: 1708.07120 [cs.LG]. URL: https://arxiv.org/abs/1708.07120. 5.

[20] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, Vijay Vasudevan, Wei Han, Jiquan Ngiam, Hang Zhao, Aleksei Timofeev, Scott Ettinger, Maxim Krivokon, Amy Gao, Aditya Joshi, Yu Zhang, Jonathon Shlens, Zhifeng Chen, and Dragomir Anguelov. “Scalability in Perception for Autonomous Driving: Waymo Open Dataset”. In: *CVPR*. June 2020. 2, 5.

[21] Haotian Tang, Zhijian Liu, Shengyu Zhao, Yujun Lin, Ji Lin, Hanrui Wang, and Song Han. “Searching Efficient 3D Architectures with Sparse Point-Voxel Convolution”. In: *ECCV*. 2020. 2, 4.

[22] Hugues Thomas, Charles R Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J Guibas. “Kpconv: Flexible and deformable convolution for point clouds”. In: *ICCV*. 2019. 2.

[23] Xiaoyang Wu, Li Jiang, Peng-Shuai Wang, Zhijian Liu, Xihui Liu, Yu Qiao, Wanli Ouyang, Tong He, and Hengshuang Zhao. “Point Transformer V3: Simpler, Faster, Stronger”. In: *CVPR*. 2023. 2, 4.

[24] Xiaoyang Wu, Yixing Lao, Li Jiang, Xihui Liu, and Hengshuang Zhao. “Point transformer V2: Grouped Vector Attention and Partition-based Pooling”. In: *NIPS*. 2022. 2, 4.

[25] Xiaoyang Wu, Zhuotao Tian, Xin Wen, Bohao Peng, Xihui Liu, Kaicheng Yu, and Hengshuang Zhao. “Towards Large-scale 3D Representation Learning with Multi-dataset Point Prompt Training”. In: *CVPR*. 2024. 4.

[26] Pengchuan Xiao, Zhenlei Shao, Steven Hao, Zishuo Zhang, Xiaolin Chai, Judy Jiao, Zesong Li, Jian Wu, Kai Sun, Kun Jiang, et al. “Pandaset: Advanced sensor suite dataset for autonomous driving”. In: *International Intelligent Transportation Systems Conference*. 2021. 5.

[27] Xiang Xu, Lingdong Kong, Hui Shuai, and Qingshan Liu. “FRNet: Frustum-Range Networks for Scalable LiDAR Segmentation”. In: *IEEE Transactions on Image Processing* (2025). 1, 2, 4, 9.

[28] Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip H.S. Torr, and Vladlen Koltun. “Point Transformer”. In: *ICCV*. 2021. 2.

[29] Xinge Zhu, Hui Zhou, Tai Wang, Fangzhou Hong, Yuexin Ma, Wei Li, Hongsheng Li, and Dahua Lin. “Cylindrical and asymmetrical 3d convolution networks for lidar segmentation”. In: *CVPR*. 2021. 2, 4.

## Appendix I
## Sensor Calibration

Here we detail the procedures for both the intrinsic and extrinsic calibration used to align all sensors to `base_link` of the SenseBike.

### A. *Intrinsic Calibration*

*1) Camera - ArduCam IMX477:* For our project, we only make use of the front-facing ArduCam IMX477 camera. The camera is connected to the NVIDIA Jetson through the HDMI port, and we publish the images with timestamps into ROS using the `ros_deep_learning` package [7] on the `/video_source/raw topic`. We then utilize the `camera_calibration` package [3] to obtain the intrinsic parameters of the camera using a checkerboard and the calibration procedure. We publish the camera information under the `/video_source/camera_info` topic.

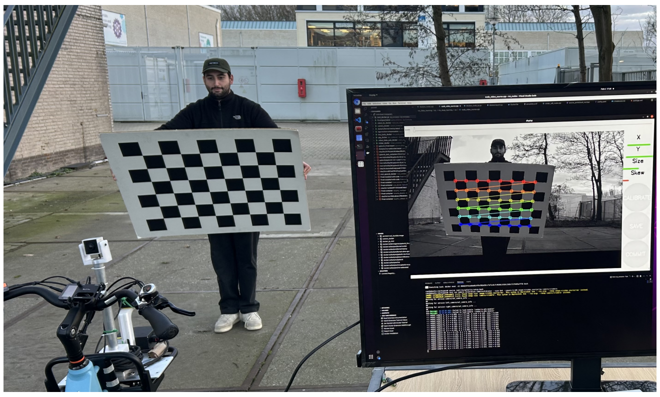
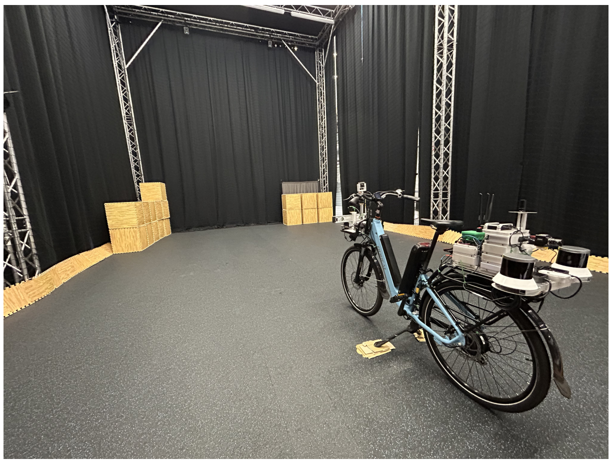


Fig. 11: Intrinsic camera calibration using a checkerboard.

### B. *Extrinsic Calibration*

*1) LiDARs:* To accurately determine the static transformations between the LiDAR sensors, the SenseBike was first positioned within the drone cage. The front-facing solid-state LiDAR (RoboSense M1P) was manually aligned using a water level, and its approximate position was initially measured using a tape measure and plumb lines.


(a) SenseBike placed inside the drone cage.

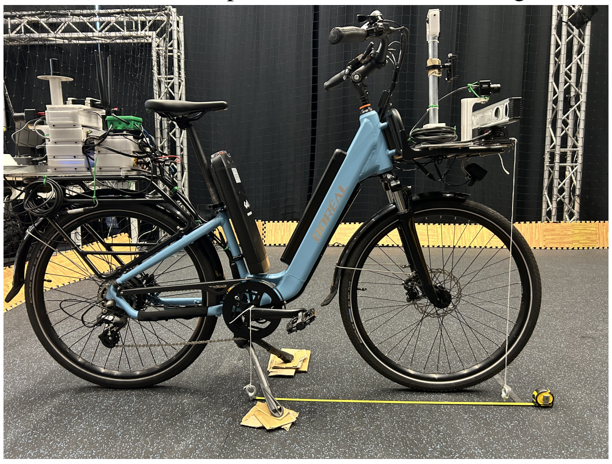
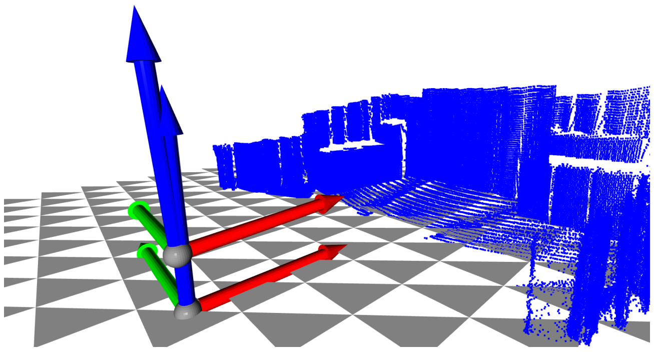

(b) Measurement of the M1P sensor w.r.t. `base_link`.

Fig. 12: Setup for sensor calibration.

*a) M1 Plus Sensor Calibration via Ground Plane Detection:* Since the precise sensor origin is unknown, we refined the M1 Plus sensor's pose by detecting the ground plane using the RANSAC algorithm. A set of 78 pointclouds was processed to compute the average distance from the sensor's $z$-axis to the detected ground plane and to estimate the corresponding pitch angle. Since the sensor was level (verified by the water level), the roll angle was set to zero.

*b) Helios LiDAR Calibration via ICP:* Next, the positions of the two rear LiDAR sensors (helios_L and helios_R) were determined with respect to the M1 Plus sensor. An Iterative Closest Point (ICP) algorithm was used to register the rear sensors' point clouds with that of the M1 Plus sensor. In this process, the yaw angle of the M1P sensor was fine-tuned by iteratively varying it until the axes of the rear LiDAR sensors aligned with the 3D model of the setup.

This two-step calibration approach—using RANSAC for initial ground plane detection and ICP for refining the relative sensor poses—ensures that the LiDAR sensor locations are accurately defined relative to the `base_link` frame.


(a) Pose of M1P w.r.t. ground plane.

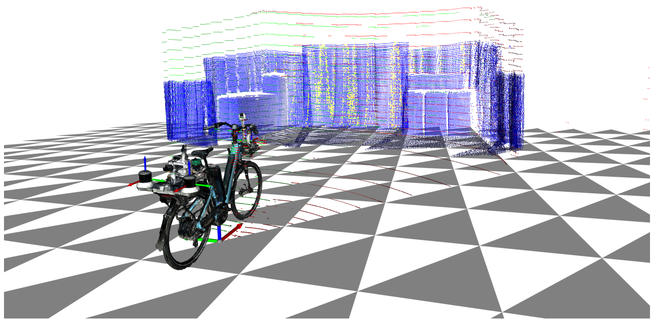
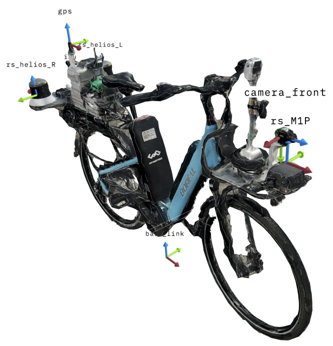

(b) Refinement of the rear Helios LiDARs with ICP.

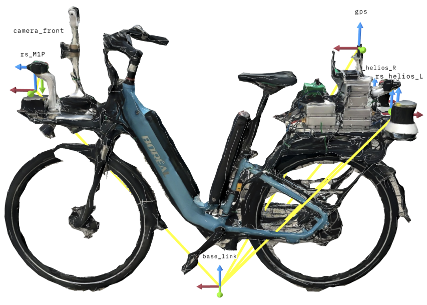
Fig. 13: Setup for sensor calibration.

*c) IMU & GPS.:* Offsets from `base_link` measured and verified using the 3D model of the SenseBike.

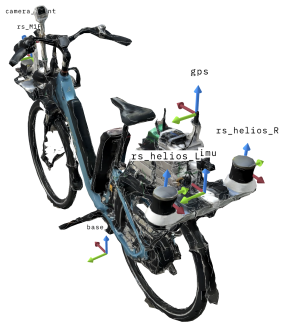
### *C. URDF Model*



(a)



(b)



(c)

Fig. 14: URDF of the SenseBike.

| **Child link** | **Translation** [m] | **RPY** [rad] |
|---|---|---|
| rs_M1P | $(0.810, -0.004, 0.955)$ | $(0, -0.029, 0)$ |
| rs_helios_R | $(-0.737, -0.185, 0.864)$ | $(-0.042, 0.043, 3.109)$ |
| rs_helios_L | $(-0.729, 0.235, 0.886)$ | $(-0.063, 0.035, 3.116)$ |
| gps | $(-0.624, -0.0611, 1.123)$ | $(0, 0, 0)$ |
| imu | $(-0.760, 0, 0.910)$ | $(0, 0, 0)$ |
| camera_front | $(0.780, -0.004, 1.055)$ | $(-1.178, 1.518, -0.395)$ |

TABLE V: Static transformations of each sensor frame with respect to `base_link`.

## Appendix II
## Ego-Motion Compensation

In this appendix, we provide full details on the motion-induced distortions in LiDAR scans and our pipeline to compensate for this effect.

### A. *Motion Distortion*

When a LiDAR sensor is capturing a pointcloud, it scans the environment point by point following a specific scan pattern. The RoboSense M1P LiDAR has a rotating mirror that deflects the beams from side to side over 125 scan lines. The timestamps of the moment of capturing of each point in the pointcloud are stored, and can be observed in Figure 15.

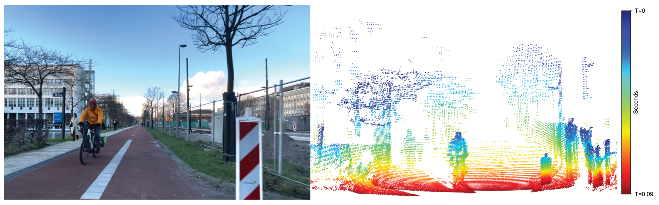


Fig. 15: Time of capture of the points in one LiDAR scan.

We can see that higher points of the pointcloud are captured earlier compared to the lower points. As the bike moves forward, the lower points that are recorded later therefore appear closer to the sensor than the higher points. This effect is more visible from a side view, which can be seen in figure 16. Motion distortion mostly affects objects close to the sensor, as the time difference between the capturing of the highest and lowest points of the object becomes larger, since more scan lines are used to capture the object. With dynamic objects, this effect is even greater as the object is also moving while the sensor is capturing.

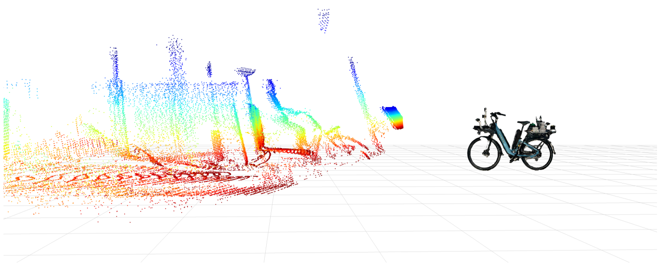


Fig. 16: Effects of motion distortion.

Motion distortion degrades point-to-point registration, mapping, and downstream perception networks; correcting it is therefore mandatory. We deskew the pointclouds by estimating the motion of the bike and transforming the points of the pointclouds accordingly. Note that purely ego-motion–based deskewing cannot correct *dynamic* objects, whose own motion is unknown. Our method will undercorrect for dynamic objects moving towards the sensor, and overcorrect for the ones moving away from the sensor. Residual artefacts around moving cyclists or cars are therefore expected.

### B. *Odometry Estimation*

To correct for the motion distortion in the LiDAR scans, an estimation of the bike's motion is required. This estimation needs to work in real-time and be able to estimate the velocity of the bike reliably. We achieve this by fusing the data from the IMU and GPS sensors on the SenseBike using the `robot_localization` Package in ROS. The Robot Localization Package uses an Extended Kalman Filter (EKF) to fuse data from multiple sensors and provides an estimate of the bike's velocity, orientation, and position.

*1) IMU – orientation pre-filtering:* The SenseBike is equipped with a SparkFun ICM-20948, a nine-degree-of-freedom inertial-measurement unit (3-axis gyroscope, accelerometer and magnetometer). The Teensy 4.0 controller board that processes the data is placed above the IMU and connected with wires without a rigid mounting, resulting in the fact that during riding it oscillates a few millimetres relative to the sensor. The current loops through the wires and ferromagnetic components therefore distort the magnetometer's readings, creating large, orientation-dependent disturbances that hard/soft-iron calibration cannot remove. Magnetometer measurements are thus discarded, leaving us with acceleration $\mathbf{a} = (\ddot{X}, \ddot{Y}, \ddot{Z})$ and angular rate $\boldsymbol{\omega} = (\omega_x, \omega_y, \omega_z)$.

We apply the Madgwick filter implementation in the `imu_tools` package [10]. The filter fuses $\boldsymbol{\omega}$ with $\mathbf{a}$, minimizing the error between the estimated and measured gravity vectors while integrating the quaternion rate. The resulting orientation quaternion is converted to roll, pitch and yaw Euler angles $(\phi, \theta, \psi)$ and published on the `/imu/data` topic.

*2) GPS – covariance fix:* The ArduSimple simpleRTK2B is read via `gpsd`. Newer `gpsd` versions compute the horizontal-error fields `epx` and `epy` only once at start-up; if the bike boots indoors their values stay $\gg$ 10 m even after good reception is regained. Consequently the ROS driver keeps pub-

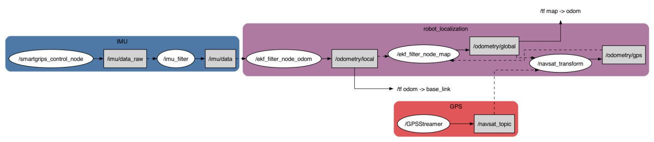


Fig. 17: Robot localization in our ROS2 environment.

lishing an overly large $P_{xy}$ entry, so the EKF in `robot_localization` ignores otherwise accurate GPS fixes. We rebuild the 3×3 position covariance using the real-time `eph` (for $x, y$) and `epv` (for $z$) fields from each TPV report, restoring proper weighting of the GPS measurement.

*3) Robot_Localization:* For the state estimation, we rely on the `robot_localization` package [12], which contains an Extended Kalman Filter (EKF) implementation and a set of helper nodes for ROS 2. The EKF tracks a 15-state vector

$$\mathbf{x} = \left[X, Y, Z, \phi, \theta, \psi, \dot{X}, \dot{Y}, \dot{Z}, \dot{\phi}, \dot{\theta}, \dot{\psi}, \ddot{X}, \ddot{Y}, \ddot{Z}\right]^{\mathsf{T}},$$

combining position, orientation, linear velocity, angular velocity and linear acceleration.

*a) Dual-EKF architecture:* Following the recommended "dual EKF" configuration, we launch three nodes:

- *EKF local* (`/ekf_filter_node_odom`)—operates in the `odom` frame and fuses IMU data (`/imu/data`) at 100Hz.
- *EKF global* (`/ekf_filter_node_map`)—adds GPS information at 10Hz; it ingests the output of the local EKF plus a GPS odometry message (`/odometry/gps`).
- *`navsat_transform`*—converts raw GPS fixes (`/navsat_topic`) into a metric odometry message in the robot's `map` frame and feeds it to EKF global.

Measurement covariance matrices come directly from each sensor, and the process noise parameters are tuned empirically. The two EKF instances publish the odometry (and corresponding transforms):

- `/odometry/local` (`odom`→`base_link`)
- `/odometry/global` (`map`→`odom`)

*b) Limitations:* Because the magnetometer is *not* fused, the filter lacks an absolute yaw reference. We therefore initialize every run with the bicycle's forward axis aligned to the map's negative-$x$ direction (facing west) to avoid an unknown heading offset. Over time the estimate is still susceptible to slow gyroscope drift, so the yaw angle in `map` can gradually diverge from the bicycle's true heading.

## Appendix III
## BikeScenes LidarSeg Dataset

### A. GLIM

To create an aggregated map of all our LiDAR scans, we ran GLIM offline as follows. We play back the rosbag within the GLIM software: incoming scans are fused into small submaps using VGICP, an iterative registration technique that minimizes the distance between points and local surfaces, combining point-to-point and point-to-plane ICP. We tuned the submapping parameters to our RoboSense M1 Plus setup so that the pipeline would stay stable even in segments containing many ghost points caused by reflective surfaces, as can be seen in Figure 23. Scans are added to the submap until one of three conditions is met: the submap already holds 35 scans, the bike has moved more than 0.4 m or rotated more than 3.1 rad since the first scan, or the overlap with the first scan drops below 60%. At that point, the scans are fused into a single pointcloud submap and its pose is added to the global factor-graph (Fig. 18). In this step, the relative motion between the first and last scan of the submap provided by the IMU, is given as an initial estimate for the optimizer.

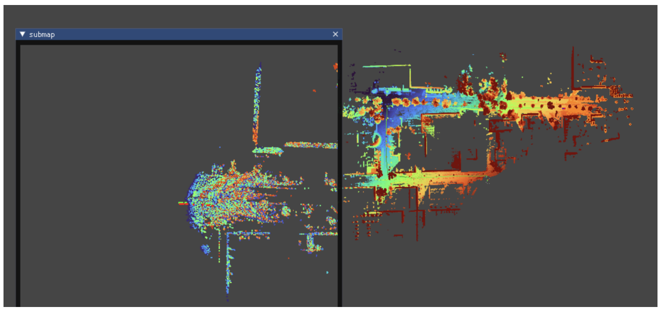


Fig. 18: Submap created and added to the global map.

The factor graph merges the submaps, again using VGICP (with slightly different parameters), and continuously updates when new submaps are appended. Each submap pose appears in the viewer as a red node with RGB axes, connected by blue scan-matching factors; once the entire loop is finished, we mark the start and finish nodes, which adds a single green loop-closure factor (Fig. 19). GLIM then overlays the two corresponding submaps and lets the user give a rough manual alignment (Fig. 20). The optimizer then aligns these begin and end submaps further, before optimizing the full graph based on the newly created

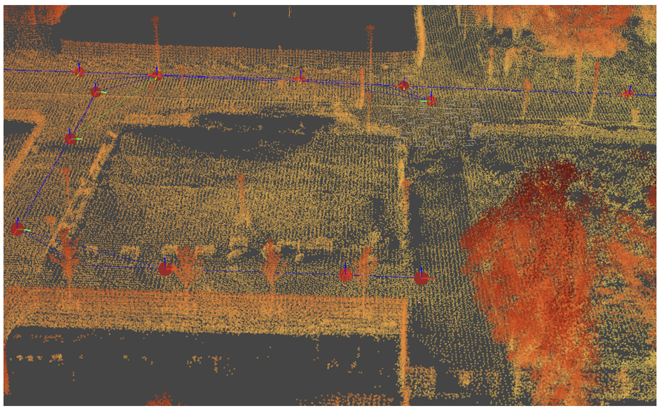
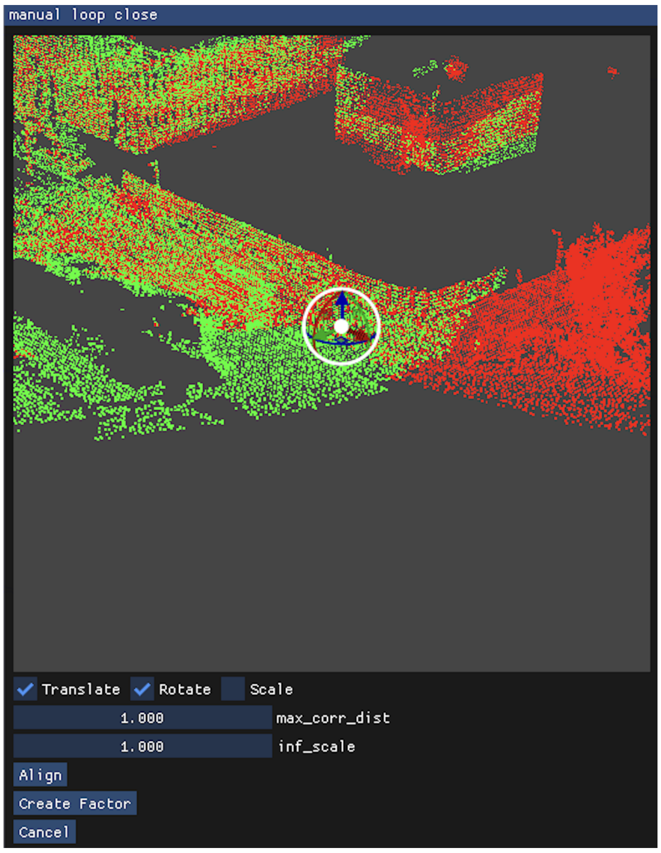


Fig. 19: Pose graph: red nodes with RGB axes depict the poses of the submaps, blue lines the factors between them.



1.000 max_corr_dist

1.000 inf_scale

Fig. 20: Manual rough alignment of the begin and end submaps.

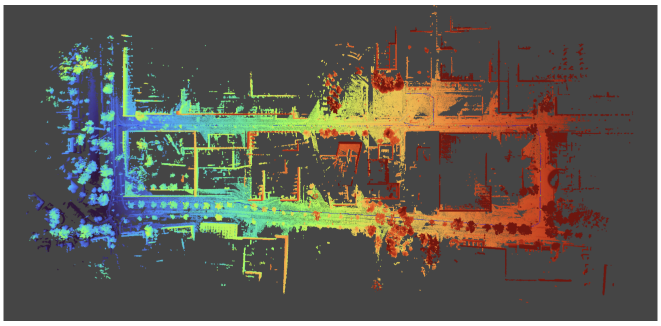


Fig. 21: Final optimized map: closed loop and aggregated pointcloud.

factor, producing a smooth closed trajectory and an aggregated pointcloud map (Fig. 21).

## B. *File Structure*

The BikeScenes LidarSeg dataset is provided in two main directories: `entire_sequence` and `subsequences`.

The `entire_sequence` directory contains the full, uninterrupted recording, along with associated images and poses for each scan. In `subsequences`, we provide the same core data, but split into sequential segments (e.g., 00, 01, etc.) designed for train/test splits to reproduce the results in our paper.

The data within `subsequences` is structured to be compatible with the SemanticKITTI format, including the velodyne (for LiDAR point clouds) and labels folder structure, to allow for easy integration with existing data loaders and models. The folder structure is as follows:

## C. *Additional Statistics and Visualizations*

Table VI shows the number of points per class per set.

## APPENDIX IV
## ROS2 SYSTEM OVERVIEW

Figure 26 shows the GUI of the SenseBike, with our addition of the Semantic Segmentation buttons. The frontend keeps track of the status of the segmentation container and informs the user with a green or red light.

### A. *ROS2 Inference Node, Mixed-Precision Optimization, and Performance*

This subsection details the implementation of the `FRNetROSInferenceNode` used for LiDAR segmentation on the SenseBike and the application of Automatic Mixed Precision (AMP) for performance enhancement.

*a) ROS2 Segmentation Node Architecture:* The `FRNetROSInferenceNode` is a Python-based

```
bikescenes/
└─entire_sequence/
  ├─velodyne/
  │ └─1739357499107089281.bin
  ├─labels/
  │ └─1739357499107089281.label
  ├─images/
  │ └─1739357499107089281.png
  ├─calib.txt
  └─poses.txt
└─subsequences/
  ├─00/
  ├─01/
  ├─.../
  └─08/
    ├─velodyne/
    │ └─000000.bin
    └─labels/
      └─000000.label
```

Fig. 22: Directory layout of the BikeScenes LidarSeg Dataset

TABLE VI: Number of points per class for Train, Validation, and Test sets (only classes used for evaluation).

| **Class** | **Train (0,4,8)** | **Test (2,6)** | **Validation (1,3,5,7)** |
|---|---:|---:|---:|
| road | 4,969,427 | 3,253,314 | 1,884,663 |
| bike path | 13,535,982 | 901,946 | 1,704,297 |
| sidewalk | 20,284,161 | 2,394,652 | 2,211,220 |
| parking | 1,335,466 | 86,388 | 4,495 |
| other-ground | 606,139 | 106,643 | 1,986 |
| building | 38,252,174 | 3,460,649 | 4,330,261 |
| car | 2,313,824 | 485,070 | 26,111 |
| truck | 1,209,792 | - | - |
| motorcycle | 298,396 | 3,185 | 19,309 |
| bicycle | 2,125,345 | 13,736 | 272,970 |
| other-vehicle | 49,691 | - | - |
| vegetation | 13,482,394 | 4,569,886 | 2,540,286 |
| trunk | 2,302,863 | 665,337 | 651,141 |
| terrain | 13,952,921 | 1,625,296 | 1,624,875 |
| person | 432,112 | 168,106 | 80,396 |
| bicyclist | 379,658 | 254,079 | 119,936 |
| motorcyclist | 3,267 | - | 33,949 |
| fence | 7,577,011 | 185,166 | 118,515 |
| pole | 2,166,914 | 281,095 | 377,510 |
| traffic sign | 467,571 | 196,617 | 225,151 |
| **Total** | **125,745,108** | **18,651,165** | **16,227,071** |

ROS2 node that performs end-to-end LiDAR semantic segmentation. Its core tasks are:

- **Subscription:** Subscribes to `/rslidar/M1P_deskewed` (`sensor_msgs/PointCloud2`), which provides distortion-corrected point clouds.
- **Conversion:** Converts each message into an `XYZI` NumPy array in the same format as SemanticKITTI `float32 .bin` files. The raw intensity values (0–255) are rescaled to the $[0, 1]$ range used by SemanticKITTI; these intensities are utilised by the FRNet model.
- **Inference:** Performs semantic segmentation with a custom `LidarSeg3DInferencer` that loads the selected FRNet model (Section III-C) and the weights from the best performing training configuration (Exp. 3, V.
- **Color mapping:** Maps each point's predicted semantic class labels to RGB colors for visualisation. The class colors of SemanticKITTI are used here.
- **Publication:** Publishes the segmented pointcloud (`XYZRGB`) as a `sensor_msgs/PointCloud2` message on `/rslidar/segmented`.

*b) Automatic Mixed Precision (AMP) for Inference Acceleration:* To improve inference speed on the GPU of the NVIDIA Jetson, Automatic Mixed Precision (AMP) is employed. This is enabled via PyTorch's `torch.cuda.amp.autocast` function during the model's forward pass. AMP utilizes both single-precision (FP32) and half-precision (FP16) floating-point numbers.

The speed benefits of FP16 stem from several factors specific to GPU architectures, particularly those from NVIDIA. Firstly, FP16 data types are half the size of FP32 (16 bits vs. 32 bits), significantly reducing memory requirements for transferring model weights and activations between a GPU's memory and compute units. Less data movement translates to faster data access. Secondly, NVIDIA GPUs can execute FP16 arithmetic operations at a much higher throughput by using specialized hardware units known as Tensor Cores. Tensor Cores are designed to perform matrix multiply operations at much higher rates when using FP16 inputs compared to FP32 [14].

While FP16 offers these performance gains, its smaller bit representation inherently provides a more limited dynamic range and lower precision compared to FP32. This means it cannot represent numbers as finely or cover as wide a range, which could theoretically lead to a loss of information or slight deviations in model output.

PyTorch's `autocast` manages precision intelligently. `Autocast` selectively applies FP16 to oper-

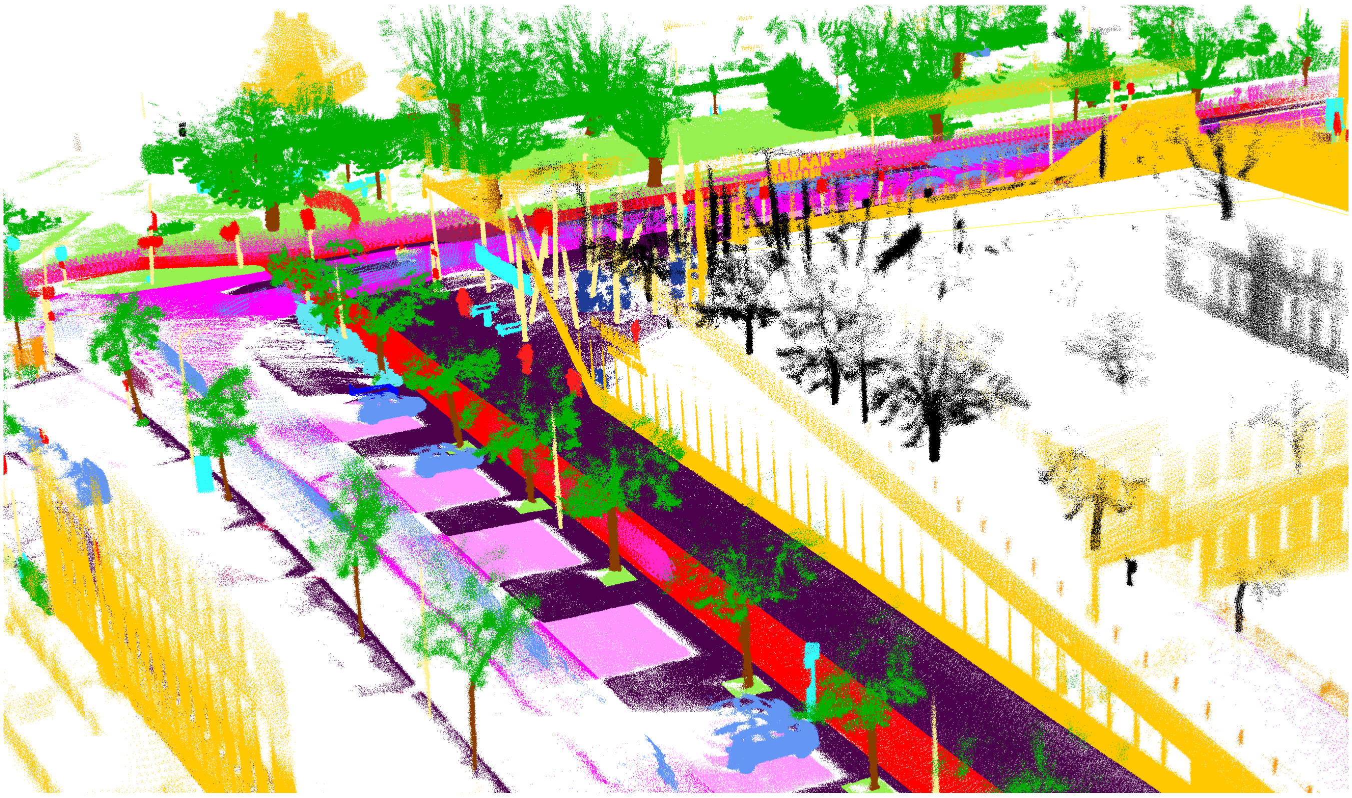


Fig. 23: The effects of ghosting: reflections of trees inside buildings (visualised in black).

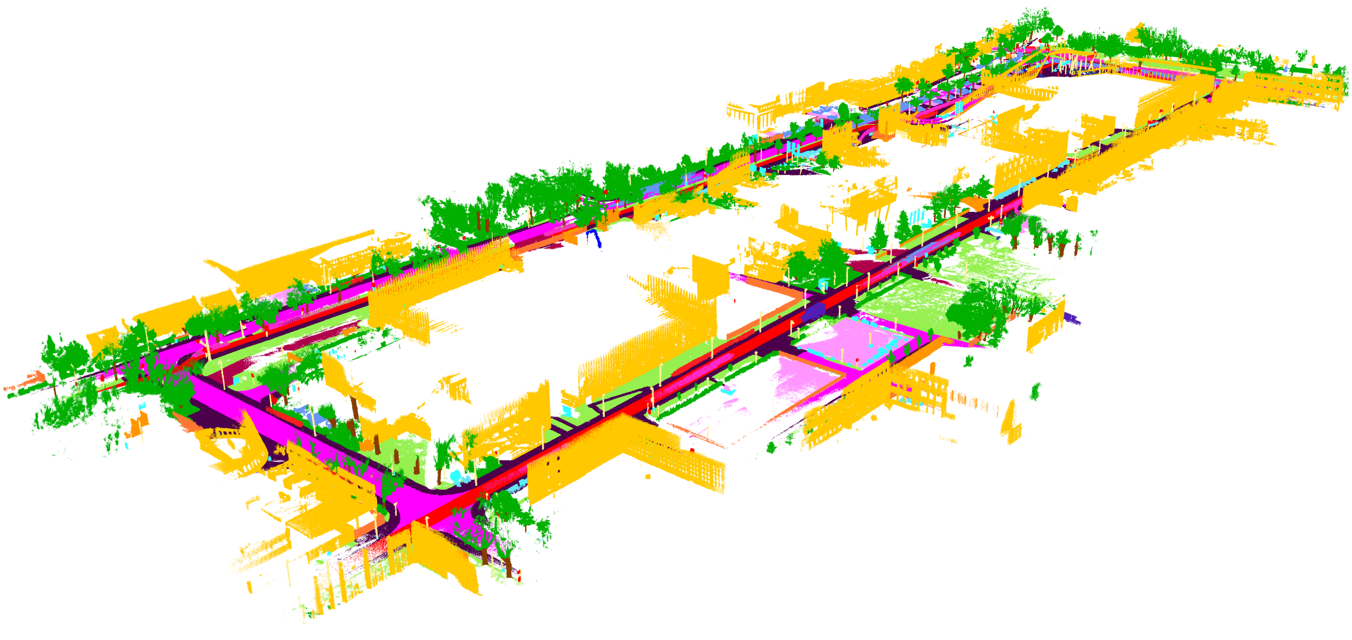


Fig. 24: Map of all the aggregated labeled scans in the BikeScenes LidarSeg Dataset

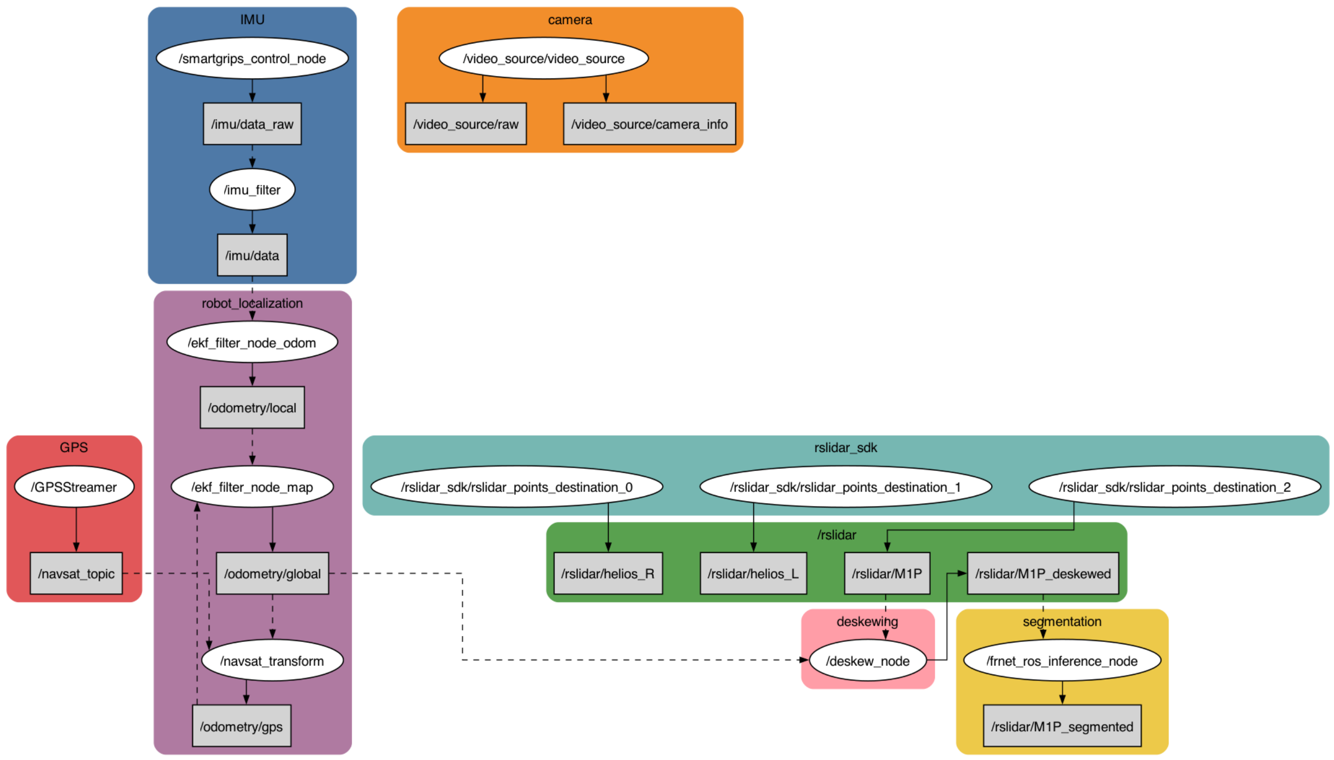


Fig. 25: Overview of our ROS2 system for online LiDAR segmentation. Ellipses represent the nodes, squares the topics, and the lines the publisher (solid) and subscriber (dashed) edges.

ations that benefit most from the speed-up and are less sensitive to precision changes (e.g., large matrix multiplications handled by Tensor Cores), while keeping operations that require higher fidelity or are numerically unstable in FP16 (such as certain normalizations or reductions) in full FP32 [17]. This strategic management ensures substantial acceleration while preserving the inference process's overall accuracy.

The effectiveness of AMP was evaluated by comparing inference speeds and segmentation accuracy with and without AMP enabled. The tests were conducted on the NVIDIA Jetson Orin NX using sequences from our BikeScenes dataset. Table VII presents the per-class IoU and overall mIoU, while Table VIII summarizes the detailed pipeline timings and resulting processing speeds.

The performance metrics detailed in Table VIII show that enabling AMP led to a substantial 30.0% reduction in average model inference time (from 868.69 ms down to 607.88 ms) and a corresponding 35.1% increase in overall system throughput, boosting the achieved FPS from 0.97 Hz to 1.31 Hz. Concurrently, the impact on segmentation accuracy, as measured by mIoU and detailed in Table VII, was negligible. This confirmed that AMP provides a beneficial trade-off for our application, significantly improving throughput with no degradation in segmentation quality, making it suitable for enhancing the online capabilities of the SenseBike platform.

## APPENDIX V
## ADDITIONAL MATERIAL

TABLE VII: Segmentation IoU: FP32 vs. AMP (zero-IoU classes excluded from mIoU calculation)

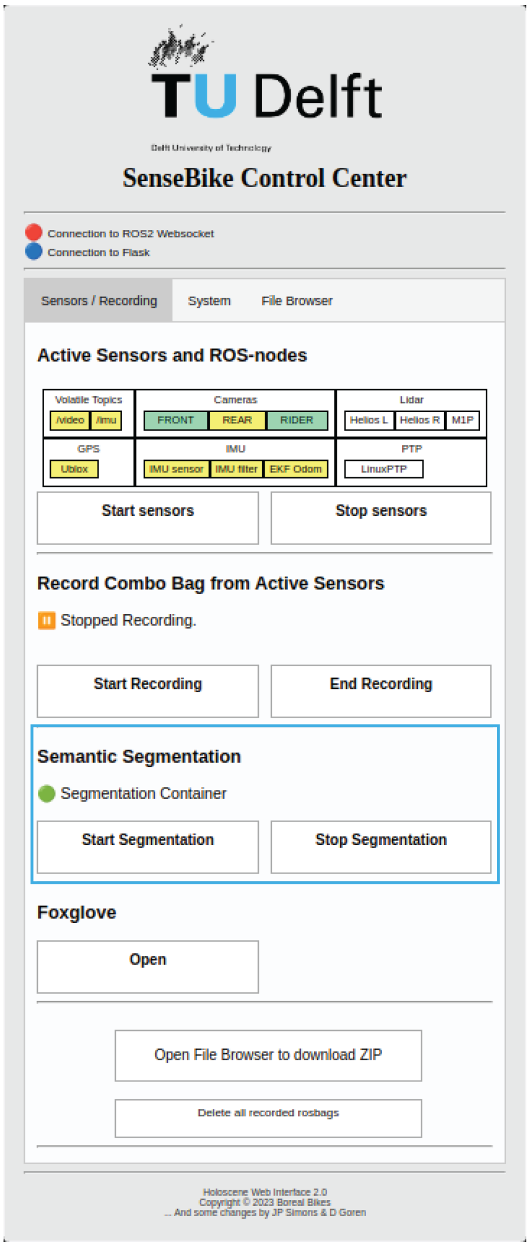
| **Mode** | **Inference Speed (Hz)** | **mIoU** | car | bicycle | motorcycle | truck | other-vehicle | person | bicyclist | motorcyclist | road | parking | sidewalk | other-ground | building | fence | vegetation | trunk | terrain | pole | traffic-sign |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FP32 | 0.97 | 63.6 | 78.9 | 26.9 | 32.4 | - | - | 83.2 | 87.5 | - | 87.2 | 3.4 | 83.4 | 2.6 | 91.6 | 79.4 | 82.5 | 61.6 | 67.2 | 72.0 | 78.5 |
| AMP | 1.31 | 63.6 | 78.9 | 26.8 | 32.2 | - | - | 83.1 | 87.6 | - | 86.9 | 3.5 | 83.3 | 2.5 | 91.6 | 79.5 | 82.5 | 61.6 | 67.1 | 72.0 | 78.5 |

TABLE VIII: Performance Comparison of the ROS2 Inference Node with and without Automatic Mixed Precision (AMP) on the NVIDIA Jetson Orin NX, averaged over 1000 samples from our online pipeline.

| **Pipeline Stage / Metric** | **Unit** | **FP32 (AMP Disabled)** | **FP16&FP32 (AMP Enabled)** |
|---|---|---|---|
| Data Conversion | ms | 72.59 | 71.46 |
| Point Filtering | ms | 2.45 | 2.41 |
| Model Inference | ms | 868.69 | 607.88 |
| Post-processing | ms | 1.63 | 1.63 |
| Data Publishing | ms | 84.35 | 82.86 |
| **Total Cycle Time** | **ms** | **1029.73** | **766.27** |
| **Achieved FPS** | **Hz** | **0.97** | **1.31** |

TABLE IX: Class-wise IoU (%) with *bike-path* kept separate (20-class evaluation). A dash "–" marks classes absent in the validation set; mIoU is averaged over the 18 present classes.

| **Exp.** | **Strategy** | **LR** | **Iter.** | mIoU | Car | Bicycle | Motorcycle | Truck | Other-Veh. | Person | Bicyclist | Motorcyclist | **Road** | **Bike-Path** | Parking | Sidewalk | Other-Gnd. | Building | Fence | Vegetation | Trunk | Terrain | Pole | Traffic-Sign |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | Scratch | 0.01 | 50k | 50.2 | 27.2 | 84.9 | 17.7 | – | – | 65.8 | 63.5 | 0.6 | 51.6 | 51.4 | 4.8 | 67.8 | 0.2 | 87.3 | 42.5 | 74.7 | 50.8 | 66.9 | 69.1 | 76.8 |



SenseBike Control Center

🔴 Connection to ROS2 Websocket
🔵 Connection to Flask

Active Sensors and ROS-nodes

| Volatile Topics | Cameras | Lidar |
|---|---|---|
| /video, /imu | FRONT, REAR, RIDER | Helios L, Helios R, M1P |
| GPS | IMU | PTP |
| Ublox | IMU sensor, IMU filter, EKF Odom | LinuxPTP |

Record Combo Bag from Active Sensors

⏸ Stopped Recording.

Semantic Segmentation

🟢 Segmentation Container

Foxglove

Holoscene Web Interface 2.0
Copyright © 2023 Boreal Bikes
... And some changes by JP Simons & D Goren

Fig. 26: Addition of Semantic Segmentation to the GUI of the SenseBike.