# OpenDR — Open Deep Learning Toolkit for Robotics

Project Start Date: 01.01.2020
Duration: 48 months
Lead contractor: Aristotle University of Thessaloniki

**Deliverable D4.3: Third report on deep environment active perception and cognition**

Date of delivery: 31 December 2022

Contributing Partners: AU, AUTH, ALU-FR, TUD, TAU, AGI
Version: v2.0

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871449.

| Title | **D4.3: Third report on deep environment active perception and cognition** |
|---|---|
| **Project** | **OpenDR** (ICT-10-2019-2020 RIA) |
| **Nature** | Report |
| **Dissemination Level:** | **PU**blic |
| **Authors** | Niclas Vödisch (ALU-FR), Tim Welschehold (ALU-FR), Abhinav Valada (ALU-FR), Illia Oleksiienko (AU), Negar Heidari (AU), Alexandros Iosifidis (AU), Anastasios Tefas (AUTH), Nikolaos Nikolaidis (AUTH), Paraskevi Nousi (AUTH), Maria Tzelepi (AUTH), Nikolaos Passalis (AUTH), Moncef Gabbouj (TAU). Anton Muravev (TAU), Jelle Luijkx (TUD), Jens Kober (TUD), Robert Babuška (TUD), Alea Scovill (AGI), Kevin Grooter (AGI), Rasmus Nyholm Jørgensen (AGI), Nima Teimouri (AGI) |
| **Lead Beneficiary** | AU (Aarhus University) |
| **WP** | 4 |
| **Doc ID:** | OPENDR_D4.3.pdf |

# Document History

| Version | Date | Reason of change |
|---|---|---|
| v0.1 | 1/9/2022 | Deliverable structure template ready |
| v1.0 | 21/11/2022 | Initial version of deliverable submitted for review |
| v1.1 | 16/12/2022 | Reviewer comments sent and minor comments added in the document |
| v2.0 | 22/12/2022 | Review comments addressed and deliverable revised |

# Contents

# Executive Summary

This document presents the status of the work performed for **WP4–Deep environment active perception and cognition** during the third year of the project. This work package consists of four main tasks, which are *Task 4.1–Object detection/recognition and semantic scene segmentation and understanding*, *Task 4.2–2D/3D object localization and tracking*, *Task 4.3–Deep SLAM and 3D scene reconstruction*, and *Task 4.4–Sensor information fusion*. After a general introduction that provides an overview of the individual chapters with a link to the main objectives of the project, the document dedicates a chapter to each of the tasks. Each chapter *(i)* provides an overview on the state of the art for the individual topics, *(ii)* details the partners' current work in each task with (initial) performance results, and *(iii)* describes the next steps for the individual tasks. Finally, a concluding chapter provides a final overview of the work and the planned future work for each individual task.

# 1 Introduction

This document describes the work progress of the third year of the project in the four major tasks of *WP4–Deep environment active perception and cognition*, namely *Task 4.1–Object detection/recognition and semantic scene segmentation and understanding*, *Task 4.2–2D/3D object localization and tracking*, *Task 4.3–Deep SLAM and 3D scene reconstruction* and *Task 4.4–Sensor information fusion*. In this section, a brief description of the work conducted by the consortium in these tasks is provided, along with a short description on how this work contributes to the Objectives of the project. A more detailed description of the conducted work is provided in the following sections.

## 1.1 Object detection/recognition and semantic scene segmentation and understanding (T4.1)

Performance of recent methods for object detection/recognition and semantic scene segmentation and understanding can be impressive, when the underlying deep learning models used are of high number of parameters. To make such methods applicable in the context of OpenDR where efficiency in terms of memory and computational cost is required, one needs to use high-performing lightweight deep learning models. Thus, emphasis needs to be placed in the design of the adopted deep learning and the training processes used to optimize the parameters of such models. To this end, OpenDR participants developed a variety of methods and tools towards tackling challenges that arise in deep environment active perception and cognition. These methods target improving performance and efficiency of deep learning models used for classification or semantic segmentation of images, via improved model architectures, or by allowing for modeling the model's uncertainty related to its response. ALU-FR proposed a method for amodal panoptic segmentation of RGB images [82] to provide pixel-wise labels of both visible and occluded regions of an image (Section 2.1).

AU proposed Layer Ensembles (Section 2.2), an uncertainty estimation method for deep neural networks that outperforms Deep Ensembles [89] in memory, speed and uncertainty quality. Layer Ensembles consider an independent Categorical distribution over the weights of each layer of the network, and randomly sample weights for these layers on each forward pass. This results in a high number of sampled models with common layer weights, but not all of them are needed to achieve good uncertainty quality. By using sample ranking, one can achieve a twice higher uncertainty quality than in Deep Ensembles while using less memory and lower number of samples. Additionally, AU continued working on Variational Neural Networks (VNNs) [87] and developed implementations in PyTorch and JAX Machine Learning frameworks [88] which provide an easy way to introduce uncertainties into an existing Machine Learning project (Section 2.3).

Furthermore, AUTH developed a deep active object detection pipeline that provides active perception capabilities to existing object detectors (Section 2.4). The proposed pipeline employs a separate planning network that regresses the rotation and translation that a robot should follow in order to increase the object detection confidence.

## 1.2 2D/3D object localization and tracking (T4.2)

Localization and tracking of objects in the 3D space requires the development of efficient methods that are able to operate in real time. OpenDR participants developed methods for 3D object

tracking operating on different types of input data, i.e. LiDAR-generated point cloud, radar, and color images.

AU and AUTH proposed a 3D single object tracking method called Voxel Pseudo Image Tracking (VPIT) [86] (Section 3.1). VPIT outperforms other 3D single object tracking methods in real-time evaluation scenarios using embedded devices, during which frames that appear before the model finishes processing the previous ones are skipped. VPIT uses a Siamese approach to compare the object features with the features of a search region to find the position and rotation of the target in the current frame, and uses PointPillars [60] as a backbone to extract features. Additionally, AU performed optimization of 3D Object Detection methods PointPillars and FairMOT by applying channel pruning on the trained models leading on a speed-up of 1.9× on NVIDIA Xavier and 2.7× on NVIDIA TX2 embedded platforms compared to the original models (Section 3.2).

ALU-FR proposed a novel method for 3D multi-object tracking using a graph neural network [12] (Section 3.3). The approach called Batch3DMOT leverages multiple modalities including camera, LiDAR, and radar, and provides 3D tracks of multiple objects in a given scene.

## 1.3 Deep SLAM and 3D scene reconstruction (T4.3)

For the field of mapping and localization, OpenDR aims to use the latest advances in deep learning to integrate network structures into classic SLAM methods to make learning more robust and efficient. Lately, several methods have put some effort into embedding traditional SLAM structures into deep network architectures. This approach has been shown to have more robust performance and better generalization capabilities compared to methods that approximate the whole procedure with a black box function approximator. For example, Neural SLAM [130] embeds the motion model, mapping, and localization procedures [13] into a completely differentiable deep neural network. The network includes an external memory architecture which is used as a "map" for the agent to learn to write onto and read from. The method in [49] encodes particle filtering processes into a network [102]. Active neural localizer incorporates traditional filtering-based localization methods into a completely differentiable network and demonstrates the effectiveness of the learned policy in both 2D and 3D simulated environments [15]. Following these ideas, OpenDR will train separate deep learning modules for each component of the SLAM pipeline. In this way, each module could either be trained end-to-end, or they could still use traditional methods but take deep features as inputs. For all the methods, the use of a lightweight deep learning methodology will be explored to allow deployment on embedded hardware.

In particular, in Section 4.1, ALU-FR introduces their work on deep learning-based loop closure detection and point cloud registration for LiDAR-based SLAM systems. During training time, PADLoC [2] leverages an attention-based network architecture with ground truth panoptic labels to enhance both tasks compared to a purely spatial approach. Moreover, in Section 4.2, ALU-FR describes a new task called continual SLAM [112] that combines lifelong SLAM with online domain adaptation to reflect challenges that occur when deploying a SLAM system to the open world. To address this task, the proposed CL-SLAM efficiently exploits both a dual-network architecture and experience replay, enabling continual online training of the visual odometry network.

In Section 4.3, AGI introduces the developed SLAM for Row Guidance System, which includes detecting crops in real time using a combination of GNSS (Global Navigation Satellite System) and deep learning. The agriculture field robot uses GNSS to determine its location

and follow a planned path provided by the farmer. The robot also has forward-facing cameras that capture images of the field. An algorithm uses these images, along with deep learning techniques, to identify the locations of crop rows. As the robot moves through the field, it navigates based on the actual positions of the crop rows and builds a map of the field in real time using Amazon Web Services (AWS). This map can be accessed through the Robotti website. This approach allows the Robotti to adapt to changes in the field.

## 1.4 Sensor information fusion (T4.4)

Computer Vision is another essential part of robotics, as visual information from the environment is widely utilized in numerous successful solutions. Deep learning algorithms in particular have shown powerful capabilities for solving tasks such as image classification, face identification, and object detection. However, they remain sensitive to input variations, which can be caused by illumination changes or other harsh visual conditions. Therefore, robotic vision can also benefit greatly from the performance boost and increased redundancy provided by sensor information fusion. Effective fusion methodologies are crucial for achieving robustness against asymmetric sensor failures and minimizing uncertainties in the predictions.

To that end, TUD built on the methodologies developed last year for efficient sensor fusion in the context of object detection for RGB and IR/depth sensors for harsh lighting conditions and data augmentation by the Random Shadows-Highlights (RSH) method, and proposed an extension to polygon-shaped masks called "Semantic Shadow and Highlights" (SSH), in order to further improve robustness against lighting perturbations (Section 5.1). Moreover, TUD is currently also working on evaluations in vision-based robot manipulation task.

Meanwhile TAU continued the developmental work on previously reported Multimodal Feature Fusion Framework (Section 5.2), tackling the issues of simulation setup, computational efficiency, robustness via input reconstruction and cross-modal compensation. Initial experiments with constructing feature encoders by neural architecture search were also performed, to limited results.

## 1.5 Connection to Project Objectives

The work carried out by the consortium in the context of WP4, as summarized in the previous subsections, is directly related to the overarching project goals. In particular, the work described here progressed the state-of-the-art towards meeting the project goals O1 and O2, as will be presented in detail below.

O1 *To provide a modular, open and non-proprietary toolkit for core robotic functionalities enabled by lightweight deep learning*

O1a *To enhance the robotic autonomy exploiting lightweight deep learning for on-board deployment*

AU and AUTH proposed a new voxel-based 3D Single Object Tracking method that operates with real-time speed on embedded GPU platforms and achieves high tracking accuracy under the real-time evaluation scenario (Section 3.1).

AU introduced a novel uncertainty estimation method called Layer Ensembles that achieves higher uncertainty quality than Deep Ensembles, while achieving higher speed and lower memory usage (Section 2.2). Additionally, AU introduced a Variational Neural Networks

(VNNs) implementation in PyTorch and JAX that allows for easy experimentation and application of VNNs to the existing projects (Section 2.3), and performed structured pruning to 3D Object Detection models for increasing their inference speed (Section 3.2).

O2 *To leverage AI and Cognition in robotics: from perception to action*

O2a *To propose, design, train and deploy models that go beyond static computer perception, towards active robot perception*

AUTH developed a novel active perception object recognition approach that provides active perception capabilities for any existing DL-based object detector (Section 2.4).

O2c *To provide tools for enhanced robot navigation, action and manipulation capabilities that can exploit if needed deep environment active robot perception*

TUD proposed an extension to its lightweight learning-free data augmentation method which creates random highlights and shadows to mimic harsh lighting conditions (Section 5.1).

ALU-FR proposed PAPS, a method for amodal panoptic segmentation, to enable robots to see "behind" objects by providing panoptic annotations for both visible and occluded regions of an image (Section 2.1). Additionally, ALU-FR proposed Batch3DMOT, an offline 3D tracking framework that leverages multiple sensor modalities to solve a multi-frame, multi-object tracking objective (Section 3.3). Furthermore, ALU-FR proposed PADLoC, an attention-based loop closure detection and point cloud registration method for LiDAR-SLAM exploiting panoptic annotations during training time (Section 4.1). Finally, ALU-FR proposed the novel task of Continual SLAM combining lifelong SLAM with online domain adaptation to effectively address challenges that occur when deploying a SLAM system in the real world (Section 4.2).

TAU committed to improvements and optimizations of its multimodal feature fusion framework for control, extending its functionality and laying the foundation for its successful integration in manipulation-requiring use cases (Section 5.2).

AGI proposed a method using SLAM that allows an agriculture field robot to navigate based on actual crop row positions instead of fully relying on RTK GNSS (Section 4.3). This method includes DL-based method to increase the confidence of the row placement and decrease possible noise from weeds and wind.

# 2 Object detection/recognition and semantic scene segmentation and understanding

## 2.1 Perceiving the Invisible: Proposal-Free Amodal Panoptic Segmentation

### 2.1.1 Introduction and objectives

Amodal perception enables human perceptual and cognitive understanding of the world [83]. This allows humans to understand the physical state and the 3D structure of the objects that are partially visible. However, robots are limited to modal perception [109, 137, 79]. A recently introduced task [81] aims to model this ability in robots. This task aims to provide both visible and occluded pixel-wise semantic segmentation labels for *things* classes (i.e. humans, cars, etc.) and only visible labels for *stuff* classes (i.e. road, sky, etc.); where each pixel can have multiple labels to keep the information for *things* classes. In this work, we present a proposal-free amodal panoptic segmentation architecture (PAPS) that addresses and eliminates the chronic issues of proposal-based architectures, e.g., generating overlapping inmodal instance masks. PAPS decomposes the amodal masks of objects in a given scene into several layers based on their relative occlusion ordering in addition to conventional instance center regression for visible object regions of the scene referred to as inmodal instance center regression. Hence, the network can focus on learning the non-overlapping segments present within each layer. An overview of the approach is depicted in Figure 1.

A summary of this work is provided hereafter. The corresponding paper is referenced below and can be found in Appendix 7.1:

- [82] R. Mohan and A. Valada, "Perceiving the Invisible: Proposal-Free Amodal Panoptic Segmentation", *IEEE Robotics and Automation Letters (RA-L)*, vol. 7, no. 4, pp. 9302-9309, 2022.

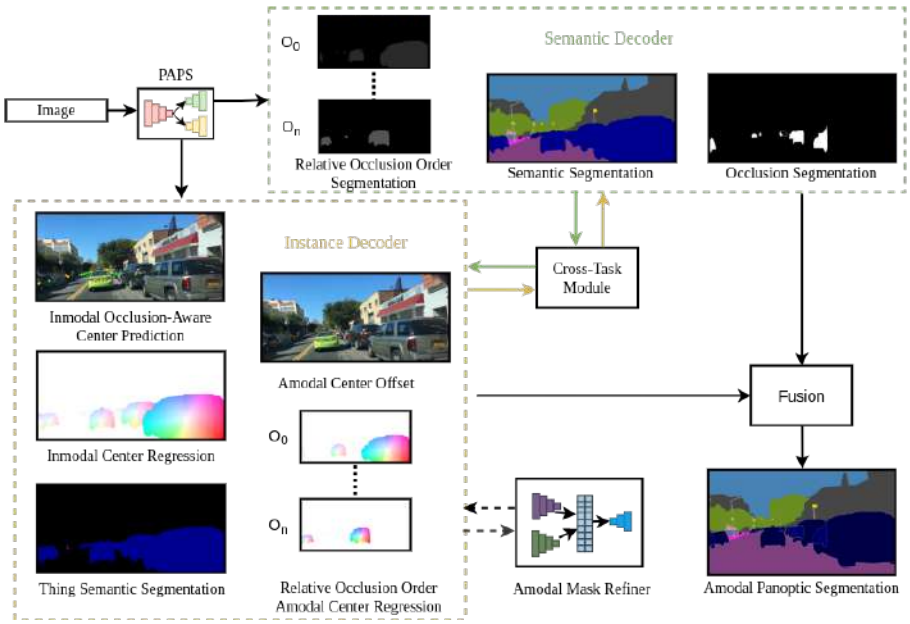



Figure 1: Overview of our proposed PAPS architecture for amodal panoptic segmentation.

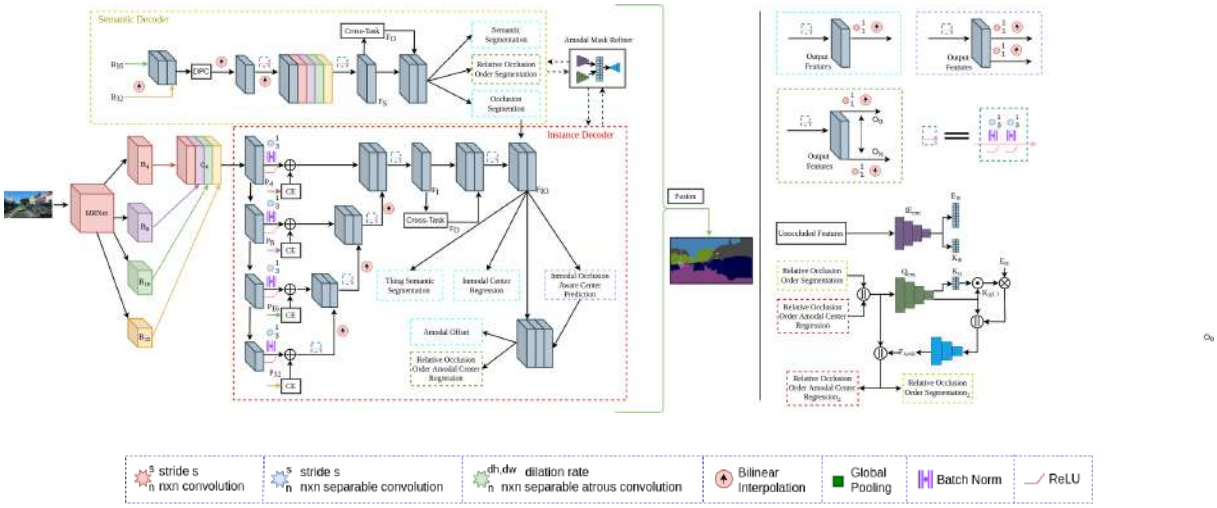



Figure 2: Illustration of our proposed PAPS architecture consisting of a shared backbone and asymmetric dual-decoder followed by a fusion module that fuses the outputs of the multiple heads of the decoder to yield the amodal panoptic segmentation output.

### 2.1.2 Summary of state of the art

Amodal panoptic segmentation is a task that has emerged recently [81]; thus, it is still an area open to research. The most recent method, APSNet [81] which is the current state-of-the-art top-down approach focuses on explicitly modeling the complex relationships between the occluders and occludees. On the other hand, panoptic segmentation has already been an extensively researched topic. In the current state of research, panoptic segmentation models can be classified in two main families: top-down approaches [95, 37, 80, 106], and bottom-up approaches [30, 20]. In this work, we present the first bottom-up approach that learns the complex relationship between the occluder and occludee by focusing on learning the relative occlusion ordering of objects, and we follow the aforementioned schema with instance center regression to obtain the panoptic variant of our proposed architecture.

### 2.1.3 Description of work performed so far

In this section, we first give a brief overview of our proposed PAPS architecture followed by a summary of each of its constituting components. For an in-depth description of the approach, we refer to the full paper [82], which is also included in Appendix 7.1. Our network structure can be seen in Figure 2. It consists of one main encoder shared backbone and two decoders that perform semantic segmentation and amodal instance segmentation. The outputs of the decoders are fused during inference to obtain the panoptic segmentation predictions.

**Backbone**: The backbone is built upon HRNet [114] which specializes in preserving high-resolution information throughout the network. It has four parallel outputs. We then upsample the feature maps and concatenate the representations of all the resolutions, followed by reducing the channels. Lastly, we aggregate multi-scale features by downsampling high-resolution representations to multiple levels.

**Context Extractor**: We design a lightweight module called the context extractor which is based on the concept of spatial pyramid pooling and is known for efficiently capturing multi-scale contexts from a fixed resolution.

**Cross Task Module**: We propose the cross-task module to enable bilateral feature propagation between the decoders to mutually benefit each other. Given feature inputs $F_I$ and $F_S$ from the two decoders, we fuse them adaptively by employing cross-attention followed by self-attention as:

$$F_R = (1 - g_1(F_S)) \cdot F_I + (1 - g_2(F_I)) \cdot F_S, \tag{1}$$
$$F_O = g_3(F_R) \cdot F_R, \tag{2}$$

where $g_1(\cdot)$, $g_2(\cdot)$, and $g_3(\cdot)$ are functions to compute feature confidence score of $F_S$ and $F_I$. $F_O$ is the output of the cross-task module.

**Semantic Decoder**: The semantic decoder takes feature maps and the output of the cross-task module as its input. First, feature maps are upsampled and concatenated to the dense prediction cell (DPC) [17]. The output of DPC is iteratively upsampled and passed through multiple convolution layers. Afterwards, we concatenate the resulting $F_S$ with the output of the cross-task module ($F_O$) and feed it to the three heads, namely, relative occlusion order segmentation ($L_{roo}$), semantic segmentation ($L_{ss}$), and occlusion segmentation ($L_{os}$). The relative occlusion order segmentation head predicts foreground mask segmentation for $O_N$ layers. We use the binary cross-entropy loss ($L_{roo}$) to train this head. Next, the semantic segmentation head predicts semantic segmentation of both *stuff* and *thing* classes, and we employ the weighted bootstrapped cross-entropy loss [123] ($L_{ss}$) for training. The overall semantic decoder loss is given as

$$L_{sem} = L_{ss} + L_{os} + L_{roo}. \tag{3}$$

The predictions from all the heads of the semantic decoder are used in the fusion module to obtain the final amodal panoptic segmentation prediction.

**Instance Decoder**: The instance decoder employs a context encoder at each scale and adds the resulting feature maps; then, the decoder repeatedly uses a processing block until $F_I$ feature resolution is obtained, which is fed to the cross-task module. Subsequently, the features from the occlusion segmentation head of the semantic decoder are concatenated to incorporate explicit pixel-wise local occlusion information referred to as $F_{IO}$ features. The instance decoder employs five prediction heads, namely, inmodal center prediction head $L_{icp}$ with occlusion awareness $L_{ico}$, the *thing* semantic segmentation $L_{tss}$, the inmodal center regression $L_{icr}$, amodal center offset $L_{aco}$, relative occlusion order amodal center regression $L_{rooacr}$. The overall loss for the instance decoder is

$$L_{inst} = L_{tss} + L_{ico} + \alpha L_{icp} + \beta (L_{icr} + L_{aco} + L_{rooacr}), \tag{4}$$

where the loss weights are $\alpha = 200$ and $\beta = 0.01$.

**Amodal Mask Refiner**: We propose the amodal mask refiner module to model the ability of humans to leverage priors on complete physical structures of objects for amodal perception, in addition to visually conditioned occlusion cues. The amodal mask refiner shown in Figure 2 consists of two encoders, unoccluded feature embeddings, and a decoder. We employ the RegNet [99] topology with its first and last stages removed. The two encoders are an inmodal embedding encoder that encodes unoccluded objects features and a query encoder that encodes the amodal features. Then after several processes, we obtain the output. We refer to this output as $F_{AMR}$. The resulting features enrich the amodal features of occluded objects with similar unoccluded object features, thereby enabling our network to predict more accurate amodal masks.

Table 1: Comparison of amodal panoptic segmentation benchmarking results on the KITTI-360-APS and BDD100K-APS validation set. Subscripts *S* and *T* refer to *stuff* and *thing* classes respectively. All scores are in [%].

| Model | KITTI-360-APS | | | | | | BDD100K-APS | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | APQ | APC | $APQ_S$ | $APQ_T$ | $APC_S$ | $APC_T$ | APQ | APC | $APQ_S$ | $APQ_T$ | $APC_S$ | $APC_T$ |
| Amodal-EfficientPS | 41.1 | 57.6 | 46.2 | 33.1 | 58.1 | 56.6 | 44.9 | 46.2 | 54.9 | 29.9 | 64.7 | 41.4 |
| ORCNN [26] | 41.1 | 57.5 | 46.2 | 33.1 | 58.1 | 56.6 | 44.9 | 46.2 | 54.9 | 29.9 | 64.7 | 41.5 |
| BCNet [50] | 41.6 | 57.9 | 46.2 | 34.4 | 58.1 | 57.6 | 45.2 | 46.4 | 55.0 | 30.7 | 64.7 | 42.1 |
| VQ-VAE [47] | 41.7 | 58.0 | 46.2 | 34.6 | 58.1 | 57.8 | 45.3 | 46.5 | 54.9 | 30.8 | 64.7 | 42.2 |
| Shape Prior [122] | 41.8 | 58.2 | 46.2 | 35.0 | 58.1 | 58.2 | 45.4 | 46.6 | 55.0 | 31.0 | 64.8 | 42.6 |
| ASN [98] | 41.9 | 58.2 | 46.2 | 35.2 | 58.1 | 58.3 | 45.5 | 46.6 | 55.0 | 31.2 | 64.8 | 42.7 |
| APSNet [81] | 42.9 | 59.0 | 46.7 | 36.9 | 58.5 | 59.9 | 46.3 | 47.3 | 55.4 | 32.8 | 65.1 | 44.5 |
| PAPS (Ours) | **44.6** | **61.4** | **47.5** | **40.1** | **59.2** | **64.7** | **48.7** | **50.4** | **56.5** | **37.1** | **66.4** | **51.6** |

### 2.1.4 Performance evaluation

We have trained the PAPS module in two different datasets, namely *KITTI-360-APS* and *BDD100K-APS* [81]. Both of these datasets provide amodal panoptic annotations for the respective datasets KITTI-360 [69] and BDD100K [125]. All our models are trained using the PyTorch library on 8 NVIDIA TITAN RTX GPUs with a batch size of 8. We present results comparing the performance of our proposed PAPS architecture against current state-of-the-art amodal panoptic segmentation approaches. We report the APQ and APC metrics of the existing state-of-the-art methods directly from the published manuscript [81]. Table 1 presents the benchmarking results on both datasets. PAPS outperforms the current methods for both datasets and can be considered the new state-of-the-art method for the amodal panoptic segmentation task. Detailed results on the specific modules that are proposed, together with different architectural comparisons can be found in Appendix 7.1.

### 2.1.5 Future Work

In this section, we presented the first proposal-free amodal panoptic segmentation architecture that achieves state-of-the-art performance on both the KITTI-360-APS and BDD100K-APS datasets. To facilitate learning proposal-free amodal panoptic segmentation, our PAPS network learns amodal center offsets from the inmodal instance center predictions while decomposing the scene into different relative occlusion ordering layers such that there are no overlapping amodal instance masks within a layer. In the future, we will extend this framework to the task of panoptic tracking to obtain consistent object IDs across a sequence of frames.

## 2.2 Layer Ensembles

### 2.2.1 Introduction and objectives

Bayesian Neural Networks (BNNs) [73, 121, 16] can be used to estimate prediction uncertainty by sampling different models from the weights' distribution and computing the mean and variance of their outputs. Depending on the type of the adopted distribution, different types of BNNs arise, including Bayes By Backpropagation (BBB) [7] with Gaussian distribution, Hypermodel [24] methods, Monte Carlo Dropout (MCD) [29] with Bernoulli distribution, and Deep Ensembles [89] with Categorical distribution.

We introduce Layer Ensembles, which consider multiple weight options for each layer that are sampled using independent Categorical distributions, resulting in a high number of possible models that can have common layer samples. The preprint describing this method in more detail can be found in Appendix 7.2:

- [85] I. Oleksiienko and A. Iosifidis, "Layer Ensembles", arXiv:2210.04882, 2022.

### 2.2.2 Summary of state of the art

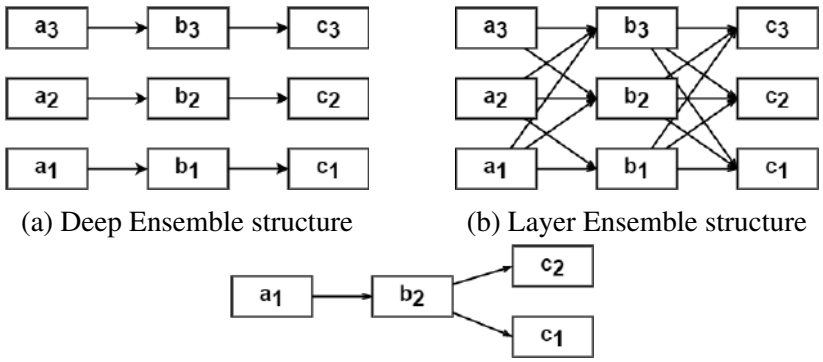
Different approaches to uncertainty estimation result in different statistical quality and computation requirements of the process. This means that in order to decide which methods are better for the task at hand, they should be ranked by the uncertainty quality and computational requirements needed to achieve such quality. To estimate the uncertainty quality of the method, one needs a dataset containing the true uncertainty values, that can be used to compute the error betweeen the model predictions to the ideal ones. However, it is a hard task to create such dataset for real-life data, and therefore Epistemic Neural Networks (ENNs) [90] propose a framework to estimate uncertainty quality using a synthetic dataset with a Neural Network Gaussian Process (NNGP) [61] trained on it, representing the true predictive distribution for each test point. The method of interest is evaluated using the KL-divergence [58] between the true predictive distribution from NNGP and the predictive distribution of the model of interest.

Monte Carlo Dropout (MCD) [29] uses Dropout [107] layers during both training and inference phases. This leads to multiple randomly sampled models providing different outputs for an input, the variance of which is an estimate of the network uncertainty to its response. Bayes By Backpropagation (BBB) [7] considers a Gaussian distribution over the network weights and trains it using the Backpropagation algorithm by applying the reparametrization trick [53]. Variational Neural Networks (VNNs) [87] consider a Gaussian distribution over the output of each layer, the mean and variance of which are parametrized by the outputs of the corresponding sublayers. Hypermodels [24] consider an additional hypermodel $\theta = g_\nu(z)$ to generate parameters of a base model $f_\theta(x)$ using a random variable $z \sim \mathcal{N}(0, I)$ as an input to the hypermodel.

Deep Ensembles [89] provide hiqh-quality uncertainty and can be described as a BNN with Categorical distribution over the network weights, where the target number of samples is equal to the number of networks in the ensemble. The uncertainty quality of Deep Ensembles can be improved by adding predictions of untrained prior networks to the outputs, as shown in [89]. Deep Sub-Ensembles [110] divide the network into two parts, where the first one is the same for all ensembles and is computed only once, while the last part is a regular ensemble of smaller subnetworks. This reduces the memory and time requirements, but also negatively impacts the uncertainty quality of the resulting ensemble. Batch Ensembles [118] optimize Deep Ensembles by using Hadamard product operations instead of matrix multiplications, resulting in better inference speed and lower memory requirements.

### 2.2.3 Description of work performed so far

A neural network $F(x, w)$ parametrized by weights $w$ processes an input $x$ and contains $N$ layers. A Deep Ensemble formed by $K$ neural networks contains $K$ instances of weights $w_i$, $i \in [1, K]$ which are trained independently and applies the same transformation $F(x, w_i)$ with different values of $w_i$. We formulate Layer Ensembles as a stochastic neural network with $N$ layers




(a) Deep Ensemble structure (b) Layer Ensemble structure

(c) Two samples of a Layer Ensemble network with common first two layer options

Figure 3: Example structures of (a) Deep Ensembles and (b) Layer Ensembles for a 3-layer network with 3 ensembles ($N = 3$, $K = 3$). While the memory structure remains identical, Layer Ensembles have many more options for sampling that can be optimized considering the common layers in samples. Layer Ensembles with common layers earlier in the architecture lead to faster inference (c).

$\text{LE}_i(x, w_q^i)$, $i \in [1, K]$, $q \in [1, N]$ and $K$ weight options for each layer:

$$w_q^i \sim \text{Categorical}(K). \tag{5}$$

The memory structure of Layer Ensembles is identical to the Deep Ensembles with $KN$ layer weight tensors, but layer weights in Layer Ensembles are independently sampled and, therefore, layers of different ensembles can be connected to form a model sample. This gives a much higher number of possible samples, many of which will contain common subnetworks that can be computed once for multiple samples. Fig. 3 shows how the same memory structure is used differently in Deep Ensembles (Fig. 3a) and Layer Ensembles (Fig. 3b).

Layer Ensembles are trained in the same way as Deep Ensembles, with a regular loss and averaging predictions of different model samples. The number of samples for Deep Ensembles is usually selected to be $K$, i.e, equal to the number of neural networks in the ensemble, but Layer Ensembles with $K$ options for each of $N$ layers has $K^N$ possible samples. Using all possible ensembles in that case is not computationally efficient but, in order to effectively train the model it is not required to use all samples, as many of them are overlapping. This means that the network will be expected to use all layer samples on average with at least $K$ samples per training step.

Following [89], we add untrained prior networks to Layer Ensembles and use the same draws from Categorical distributions for their inference, as for the corresponding trained networks. Experiments show that this improves the uncertainty quality of Layer Ensembles by a factor of 2 for each number of ensembles.

Deep Ensembles [89] and Deep Sub-Ensembles [110] can be viewed as specific cases of Layer Ensembles. Considering $K$ layer options for each of $N$ layers, the number of Layer Ensemble samples is $K^N$. By sampling $K$ networks where no layers are used in two different networks, we obtain a Deep Ensemble. By, further, considering 1 layer option for the first $T$ layers and $K$ options for the remaining $N - T$ layers, we obtain a Deep Sub-Ensemble.

**Inference Optimization** When required to compute multiple samples at once, the overlapping parts of the samples that receive identical inputs can be computed only once for two or

more sub-networks. Fig. 3c shows a Layer Ensemble in which the first two layers are the same for both networks, but the last layer is different. Instead of computing $c_2(b_2(a_1(x)))$ and $c_1(b_2(a_1(x)))$ independently, we first compute the common value $V = b_2(a_1(x))$ first, and then $c_2(V)$ and $c_1(V)$. An implementation of this idea is provided in Algorithm 1, which recursively computes the output of Layer Ensembles for a set of sorted samples. The samples are represented as an array of layer indices, such as $[1,2,2]$ and $[1,2,1]$ for the model in Fig. 3c, these samples are sorted by the first-most values, and if equal, by the later indices. This allows to placing the most overlapping samples in a row and to compute the common layer value at the start of a sequence. The result of Optimized Layer Ensembles (OLE) is an array of all outputs, which can be later used to calculate the mean and variance of predictions. The memory saved and speed up of OLE, depending on the number of ensembles, of a 4-layer CNN on the MNIST dataset are shown in Fig. 1.

**Algorithm 1** Optimized Layer Ensembles

**Require:** Network $F(x)$, list of sorted samples $S$, layer index $i$, input $x$

**function** OLE($F, S_i, i, x$)
  result $\leftarrow$ []
  $s_{i+1} \leftarrow$ []
  **if** $i =$ size($s$) **then return** $[x]$ ▷ Final layer computed
  **end if**
  $s_l \leftarrow S_i[0]$
  $l \leftarrow F[i][s_l](x)$ ▷ First sampled option for layer $i$
  **for** $t \in [0..\text{size}(S_i)]$ **do** ▷ For each sample
    **if** $S_i[t] \neq s_l$ **then**
      result = result $\cup$ OLE($F, s_{i+1}, i+1, l$)
      $s_l \leftarrow S_i[t]$
      ▷ Next sampled option for layer $i$
      $l \leftarrow F[i][s_l](x)$
      $s_{i+1} \leftarrow$ []
    **end if**
    ▷ Update sub-samples list for input $l$
    $s_{i+1} \leftarrow s_{i+1} \cup s_i[t][1:]$
  **end for**
  result = result $\cup$ OLE($F, s_{i+1}, i+1, l$)
  **return** result
**end function**
**return** OLE($F, S, 0, x$)

### 2.2.4 Performance evaluation

We implement Layer Ensembles as a part of Epistemic Neural Networks (ENNs) [90] framework to estimate the uncertainty quality of the method and to compare it with Deep Ensembles. ENNs consider a regression task $y = f(x) + \varepsilon$ and create a synthetic dataset for this problem in order to train an NNGP that represents the true predictive distribution. The dataset is described as $D_T = \{(x,y)_t \text{ for } t \in [0, T-1]\}$, where $x$ is a $D_x$-dimensional input vector, $y$ is an output scalar, $\varepsilon \sim \mathcal{N}(0, \sigma^2)$ is a random noise, and $T = D_x\lambda$ is a dataset size. For each data point,

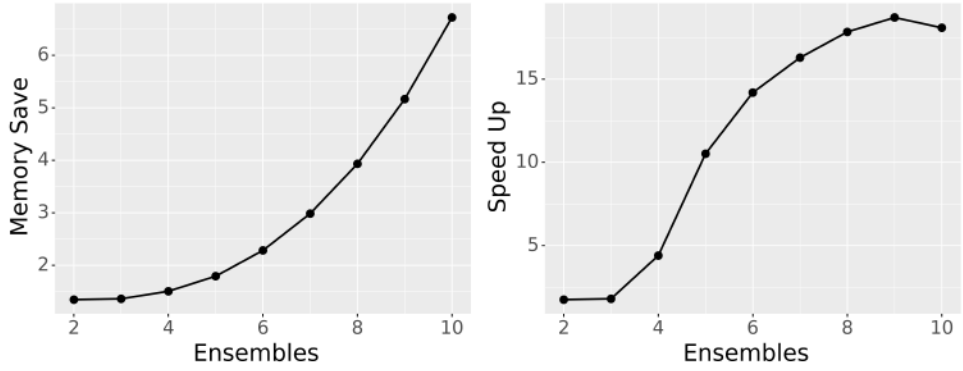



Figure 4: Speed up and memory saved during inference of Optimized Layer Ensembles compared to the regular Layer Ensembles for different number of ensembles of a 4-layer CNN for MNIST classification. This excludes memory used for the ML framework and model weights.

the model of interest should provide mean and variance of predictions, which are treated as a Gaussian predictive distribution $\mathcal{N}(\mu, \sigma^2)$. These predictions are compared to the true ones by computing the KL-divergence between Gaussians, obtained from the model $M$ and from the NNGP. The result $Q(M)$ is averaged across all data points:

$$\begin{aligned} Q(M) &= \frac{1}{T}\sum_{t=0}^{T-1} \mathrm{KL}(\mathcal{N}_M \parallel \mathcal{N}_{\mathrm{NNGP}}), \\ \mathcal{N}_M &= \mathcal{N}(\mathbb{E}[M(x_t)], \mathrm{Var}[M(x_t)]), \\ \mathcal{N}_{\mathrm{NNGP}} &= \mathcal{N}(\mathbb{E}[\mathrm{NNGP}(x_t)], \mathrm{Var}[\mathrm{NNGP}(x_t)]), \end{aligned} \tag{6}$$

where $M$ and NNGP are the model of interest and the true NNGP model, respectively, and KL is a Kullback–Leibler divergence function [58].

Fig. 5 illustrates the comparison between Deep Ensembles and Layer Ensembles for different numbers of ensembles and samples. Layer Ensembles start to achieve good uncertainty quality at a much lower number of ensembles and outperform Deep Ensembles for the same number of ensembles, however, requiring more samples to be processed. To overcome this problem, we introduce a layer sample ranking process that allows to improve performance in both speed and memory, while keeping better uncertainty quality.

**Layer sample ranking** Many Layer Ensemble samples overlap, and even though there is a way to efficiently compute all the outputs, we do not always need them all to make a good prediction. Random selection of samples, as shown in Fig. 5 decreases the uncertainty quality of a model. Instead of that, we can specifically select desired samples by using a ranking process. This can be done by computing the uncertainty quality of all possible combinations on the validation set and selecting the best ones but, to reduce the computational cost of the ranking process we progressively add samples to the best sample set by computing the validation uncertainty quality of the existing set with each such addition and selecting the best one. We start from a single sample with the best mean error $s^1$:

$$s^1 = \underset{s_j}{argmax}\ Q(M^{\{s_j\}}), \tag{7}$$

where $Q(\cdot)$ is the uncertainty quality score function, $M^{\{s_j\}}$ is the Layer Ensemble model applied to a set of layer samples with only one sample $s_j$. For a size $P$, a set of optimal layer samples

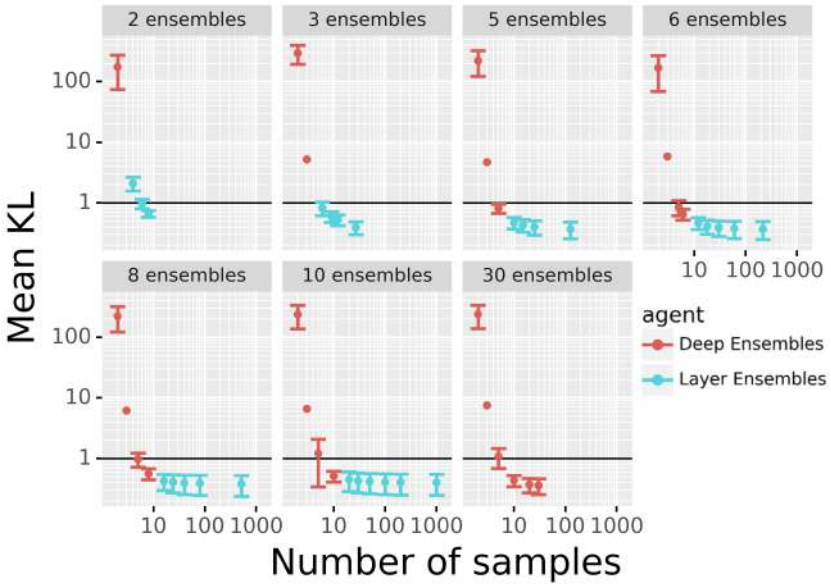



Figure 5: Comparison of mean KL values with 1 STD range for Deep Ensembles and Layer Ensembles with random unique layer samples, averaged across all experiment parameters.

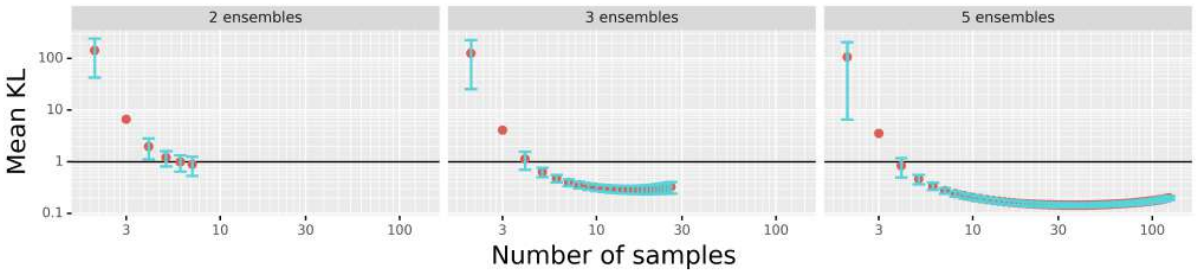



Figure 6: Comparison of mean KL values with 1 STD range for Layer Ensembles with different number of ensembles and sampled layers, averaged across all experiment parameters. The best layer samples are selected based on the validation set and evaluated on the test set.

$S^P = \{s_i^P | i \in [1..P]\}$ is enriched by the best sample as follows:

$$\begin{aligned} s^{P+1} &= \underset{s_j}{argmax}\ Q(M^{S^P \cup s_j}), \\ S^{P+1} &= S^P \cup s^{P+1}. \end{aligned} \tag{8}$$

Fig. 6 shows the uncertainty quality results obtained by using the best samples for different number of samples and different number of ensembles. Starting from 3 ensembles, the optimal number of samples is reached far before the maximum, with a clear optimum, which means that sample ranking improves both inference speed and uncertainty quality. Layer Ensembles with 5 ensembles achieve the best uncertainty quality at 36 samples, and with 20 samples the uncertainty quality is lowered only by 6%. This is, still, 2 times better than the uncertainty quality of Deep Ensembles with 30 ensembles, while using 6 times less memory and at least 1.5 times less inference time. This advantage of Layer Ensembles in computation time can be possibly improved using OLE, depending on the level of overlap between samples.

## 2.3 Variational Neural Networks

### 2.3.1 Introduction and objectives

Uncertainty estimation is an important task for critical problems where silent failures of neural networks may lead to costly results, like those found in robotics applications. Many different approaches to uncertainty estimation have been proposed, with Bayesian Neural Networks (BNNs) [73, 121, 16] being the most popular one. BNNs consider a distribution over the network weights that is updated after observing the data following the Bayes rule. The choice of distribution changes the statistical and computational properties of the model, as well as the training procedure. Different distributions are used in different types of models, including Gaussian [7], Bernoulli [29] and Categorical [89] distributions. Direct Bayesian approach is hard to be achieved considering the high-dimensionality of the weight space in neural networks, which is the reason to develop methods that approximate the posterior distribution or follow indirect sampling procedures, such as Variational Inference [6] or Markov Chain Monte Carlo (MCMC) [38].

In year-2 of the OpenDR project, we introduced Variational Neural Networks (VNNs) [87] which, instead of considering a distribution over the network weights, generate in run-time a Gaussian distribution over the outputs of each layer, with mean and covariance parameters being outputs of the corresponding sub-layers. We followed [90] to estimate the uncertainty quality of VNNs and compared it to other methods, showing that VNNs outperform methods belonging to the same Bayesian Model Averaging group, i.e, Monte Carlo Dropout (MCD) [29] and Bayes By Backprop (BBB).

### 2.3.2 Description of work performed so far

We implemented VNNs in PyTorch [94] and JAX [10] Machine Learning frameworks to ensure reproducibility of image classification and uncertainty quality experiments, and to allow for an intuitive and easy way to implement VNNs into an existing Machine Learning project. The paper describing this library in more detail can be found in Appendix 7.3:

- [88] I. Oleksiienko, D. T. Tran and A. Iosifidis, "Variational Neural Networks implementation in PyTorch and Jax", *Software Impacts*, 14:100431, 2022.

## 2.4 Deep Active Object Detection

### 2.4.1 Introduction, objectives and summary of state of the art

Object detection is critical for many robotics applications. However, recent works demonstrated that there is is a multitude of factors affecting the performance of object detectors [91]. Indeed, there are many real-world scenarios in which detection accuracy will be below the expected standards. That is due to a set of factors that sometimes are controllable, e.g., robot position, camera zoom, etc., and sometimes they are not, e.g., obstacles. Most of these controllable factors have been studied with relative success, however, the optimal location/rotation of a sensor in the environment is still a less studied subject. A camera sensor that may be exposed in a non-optimal positioning/angle setting related to the object of interest, is a problem that can be solved with a translation and/or rotation in the 3D space. In this work, we follow the active perception paradigm [93] [9] and we propose a neural network architecture which searches for optimal viewpoints in the 3D space that maximize detection accuracy while at the same time

minimizing the movement-related cost. We also explore the properties of such a system, ways of optimization, as well as the experimental results of these algorithms in simulation environments.

### 2.4.2 Description of work performed so far

The process of active object detection consists of three discrete stages: (i) object detection is executed and the confidence of the object of interest is evaluated. If the detection is below a desired threshold, e.g., 80% (ii) the navigation module triggers and proposes a position/rotation in the 3D space. Further on, (iii) we evaluate the detection again on the proposed point, as well as in some points in between.

**Phase 1: Initial Object Detection**
A regular deep learning-based object detector is employed during this step. This detector can be described as a function $f_{detect}(\mathbf{x}_i)$, where $\mathbf{x}_i$ is the observation at the $i$-th time step. The observation in this particular case is a 2D projection (a camera image) of the 3D world in which the robot resides. The detector produces a confidence score $p_i$ which describes the certainty of the appearance of the object of interest in the observation $\mathbf{x}_i$, as well as its location $(y_1, y_2)$ in the 2D grid (image). If the confidence score $p_i$ of the object of interest is below a threshold (if the detection is inadequate), then we proceed with *Phase 2* and *Phase 3* in order to improve the results.

**Phase 2: Navigation Proposal**
The next step consists of inferring the movement that the robot should perform in order to increase the detection confidence. To this end, in this work we employ a dedicated *navigation network*. The navigation network is a regression convolutional neural network capable of estimating optimal navigation plan in order to maximize $p_{i+1}$ and provide active perception capabilities. In other words we try to maximize the confidence of the object detector at the next observation $\mathbf{x}_{i+1}$ at the next time step $i+1$. The navigation module is a function with inputs the observation $\mathbf{x}_i$ which outputs the translation and rotation vector for the robot, i.e., $\mathbf{z}_{i+1}$ and $\mathbf{r}_{i+1}$, that lead to the *next* optimal point in the 3D space in terms of confidence score. The final step of this phase is to apply these transformations to our robot. That means that we have to rotate around the object of interest by $\mathbf{r}_{i+1}$ and then move by $\mathbf{z}_{i+1}$ frontal or backwards.

**Phase 3: Object Detection in Trajectory**
Since we have a translation vector $\mathbf{z}_{i+1}$, and a rotation vector $\mathbf{r}_{i+1}$ we can formulate a trajectory from the starting point at time step $i$ to the ending (optimal) point at time step $i+1$. While the robot moves in this trajectory we can evaluate the object detection confidence multiple times until it reaches its final destination in order to further increase the perception accuracy. An example of robot movement is depicted in Fig. 7.

### 2.4.3 Performance evaluation

The proposed method was evaluated using the Webots simulation environment using a DJI Mavic drone as a robot, as shown in Fig. 8. We have developed a core controller for the drone which provides the necessary functionalities regarding movement and sensor data acquisition. More specifically, the drone can be moved in two ways, either by utilizing the Supervisor class in Webots [77], which allows instant translation/rotation anywhere in the 3D space without the cost, or by fully activating the drone motors and the atmosphere/gravity in the environment in

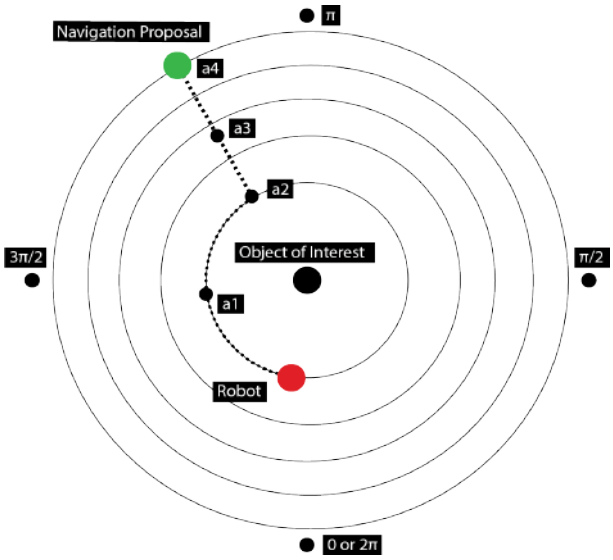



Figure 7: Movement example of a robot that performs active perception in order to increase object detection confidence. Red point: Initial Position, Green point: Navigation Proposal.

Figure 8: The Webots simulation environment was used for the conducted experiments.

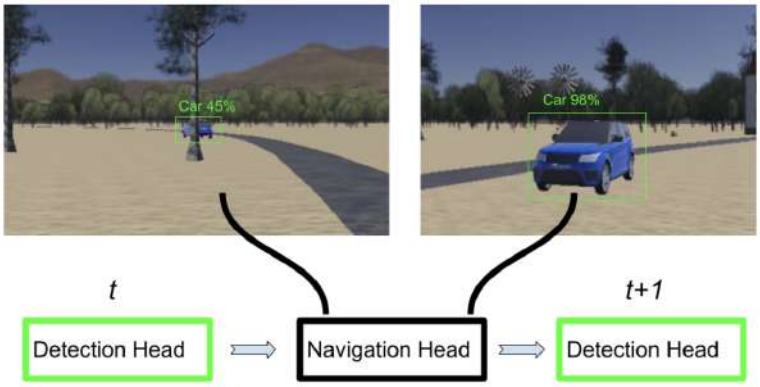



Figure 9: Active perception pipeline for object detection

order to test for real-world conditions. We have also implemented a camera system running on the drone. The camera captures data in 30 frames per second. Using the developed simulation environment we have created two datasets for training the employed DL models: a) a classic object detection dataset and a b) a active perception dataset that can support the corresponding task. For the needs of the latter dataset, we utilized simulated 3D models of a Tesla Model S and Toyota Corolla. All of the data in the synthetic car dataset were produced in this way and were later used to fine-tune pre-trained object detectors in this environment and train from scratch the custom navigation head. For the regular object detection dataset images of cars were extracted in different world settings and angles. The object detector was fine-tuned with 5,000 images around the car in 65 different radius values for 76 angles. This extra training/finetuning step was done in order for the detector to achieve better results for our use-case, instead of using the generic trained car detector. The label for each image is a pair of rotation and translation vectors that lead to a "better" point of view in the 3D space for the next observation. The max distance from the car was 60 meters.

The neural network module consists of two neural networks. The object detection network and the navigation proposal network. The object detection network always scans the area for objects, in this particular experiment cars. When it works adequately (over a pre-defined detection confidence threshold) there is no need for improvement. When it does not, the navigation proposal neural network takes over, in order to propose a movement in the 3D space which will improve object detection accuracy. These two neural networks take as input the same data (video stream frames from the camera of the drone). In this work we use an SSD-based architecture for the object detection [71] with the addition of a VGG feature extraction layer. Furthermore, instead of the original VGG fully connected layers, a set of auxiliary convolutional layers (from conv6 onwards) were added, allowing features to be extracted at multiple scales and the size of the input to each subsequent layer to be progressively reduced. For the navigation module, we used a Resnet-18 [39] convolutional architecture. The input of the network is adjusted to fit the dimensions of our camera input stream, i.e., (420x240x3), while the output of the network was adjusted in order to regress the rotation and translation vectors. The sigmoid activation function was used for the output layer of the network. Note that the training targets were appropriately normalized to support this architecture, i.e., they were normalized to (0, 1).

The complete active perception pipeline involves both of these networks. First, the detection network runs and produces confidences for the detected objects. This is compared against a pre-

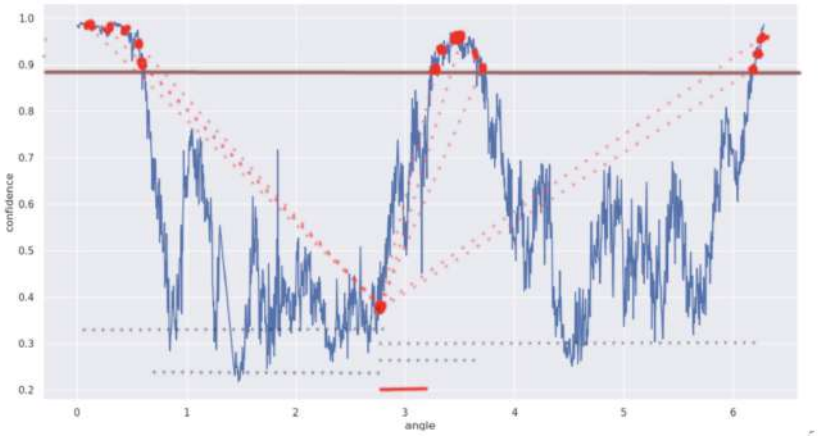



Figure 10: Mapping different position to optimal navigation plan according to object detection confidence

defined threshold. The thresholds used for the production of the navigation proposal network dataset were 0.7, 0.8, 0.9, 0.95, and 0.98. We achieved the best results using the 0.9 threshold for the dataset creation. Thus, for our use case when the detector detects a car with confidence below 0.9 accordingly, the next navigation network is enabled. Note that the navigation network takes the same input as the detection network, but produces the rotation and translation vectors instead. In order to train the navigation network, we needed ground-truth 3D translation and rotation vectors, that lead to optimal viewpoints in the 3D world. Based on the detection evaluations above we map every point inside the training 3D worlds, to another one that maximizes the viewpoint detection confidence, and then create navigation labels based on these. We map these points to the closest possible points near them over a viewpoint confidence threshold (0.95) as seen in Fig.10. For this particular example, the point that eventually gets mapped as the optimal viewpoint is the closest to the current position. The red bar below Fig. 10 represents the closest point distance from the best viewpoint. The label for this example would be the distance from the best viewpoint signed positively (meaning rotation around the object to the right), which would be +0.40 rads. For the 3D case, we do the same as in 2D, adding another dimension (translation movement). In Fig.11 we plot the confidence of an object detector at different angles and distances in order to further highlight the process of dataset creation for object detection in the 3D case.

Then, we run 100 experiments for each evaluation case and averaged the total improvement of the detector. We tested in an unknown environment for the navigator model. In every experiment from the total of 100 for each case, we randomized the robot position-rotation making it spawn in different coordinates of the 3D map, as shown in Fig. 12. The proposed method manages to increase the object detection confidence in 67% of the evaluation cases, i.e., in 67% of the evaluated cases the final object detection confidence was higher than the initial object detection confidence. A random navigation policy lead to a significantly lower improvement rate of 22%. Finally, we also evaluated a classification navigation policy, where instead of regressing the actual movement, we performed classification to select the direction of movement. In this case, the active perception policy increased the confidence in 51% of the evaluated cases.

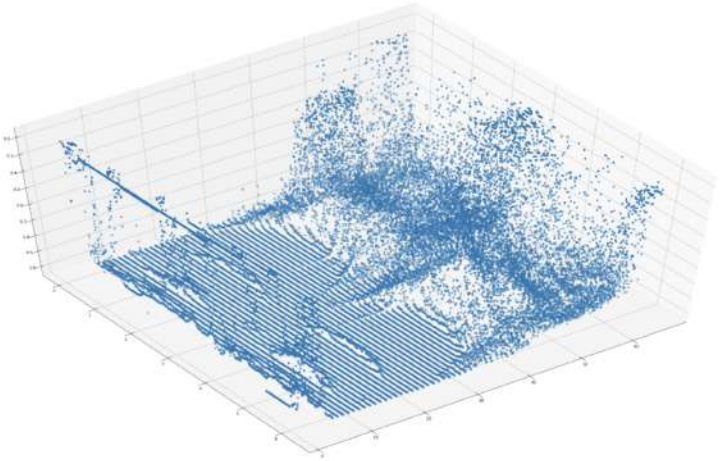


Figure 11: Object detection confidence at different angles and distances

Figure 12: Random initialization during evaluation

### 2.4.4 Conclusions and Future Work

The conducted experiments demonstrated that it is possible to train DL models to regress a robot plan that can increase object detection confidence, providing an easy and practical way to equip robots with active perception capabilities. This promising result can be further explored by a) incorporating the regression network to the object detection model, providing active perception feed at virtually no cost, b) exploiting co-integrated simulation and training in order to reduce sim-to-real gap and ensure that models will perform as expected in real conditions and c) train the models using reinforced learning-based formulations, in order to remove the time-consuming data generation step.

# 3 2D/3D object localization and tracking

## 3.1 VPIT: Real-time Embedded Single Object 3D Tracking Using Voxel Pseudo Images

### 3.1.1 Introduction

3D Single Object Tracking (3D SOT) focuses on tracking a single object and combines challenges from both 3D Object Detection and 3D Multiple Object Tracking, requiring to accurately detect the object of interest and correctly identify it, avoiding losing it due to the presence of similar objects in the scene. SOT in both 2D and 3D is usually performed by using correlation filters [8, 44], deep learning models that regress to the predicted object offset [43], voting [97, 103], or a Siamese approach[25, 4, 63, 62] where the features of the target object are computed in the previous frame and compared to the features of a search region in the current frames. We propose a novel 3D SOT method called Voxel Pseudo Image Tracking (VPIT) that uses a modified PointPillars model for generating voxel pseudo images and processes them to create features for target and search regions.

Most Lidars operate at 10-20 Hz, which means that a 3D SOT method cannot receive inputs faster than 10/20 times per second, but having higher FPS may allow other resource-heavy methods to be used in parallel. However, this is not the only benefit for tracking methods, as their performance directly depends on the sequence of inputs that are received. If the operation speed of a method is not enough compared to the sampling frequency of the Lidar, the input frames should either be queued, resulting in a constantly increasing delay between inputs and predictions, or some inputs should be skipped, which may decrease the accuracy of a tracker. For this reason, following [66] we implemented a real-time evaluation protocol that simulates frames lost due to model latency and evaluates results with the predictions "available" at the time step of a ground-truth sample.

The preprint describing this method in more detail can be found in Appendix 7.4:

- [86] I. Oleksiienko, P. Nousi, N. Passalis, A. Tefas and A. Iosifidis, "VPIT: Real-time Embedded Single Object 3D Tracking Using Voxel Pseudo Images", arXiv:2206.02619, 2022

### 3.1.2 Summary of state of the art

3D SOT methods are often formulated as extensions of 2D SOT methods. The main difficulty arises from the unordered nature of Point Clouds which cannot be processed with regular CNNs. SC3D [36] uses a Siamese approach and encodes the target point cloud shape from the initial frame and constantly fuses it with the predicted point cloud of new frames. The object search is performed by using cosine similarity between point cloud features. The search region is defined to be a small area around the target, assuming that the object offset should be small in consecutive frames. P2B [97] uses a point-wise network to create similarity maps between target and search regions and find candidates for target centers, which are later processed by a voting algorithm to select the best position candidate. BAT [134] is based on P2B and enriches point cloud features by adding Box Cloud representations which include distances between points and the center and corners of a corresponding 3D bounding box. 3D-SiamRPN [25] uses a Siamese point-wise approach, creating target and search point cloud features, which are later compared by a cross-correlation algorithm to find points of the target in a new frame. The

final bounding box is predicted using the region proposal subnetwork. F-siamese tracker [136] fuses camera and Lidar data by applying a 2D Siamese tracker on an image and then using the generated 2D proposals to create 3D frustums as inputs to a 3D Siamese model that predicts the final 3D position. Point-Track-Transformer (PTT) [103, 48] uses transformers for point-based SOT methods and employs it based on a P2B model. 3D Siam-2D [129] uses a fast 2D Siamese model in Bird's-eye View (BEV) coordinates to create BEV proposals and a second Siamese model that uses projected BEV proposals to identify which of them belongs to the object of interest. The BEV projection loses information due to pixel overlap but voxel pseudo images, despite also being in the BEV space, represent all the information from the point cloud by processing the corresponding points with small subnetworks.

### 3.1.3 Description of work performed so far

Siamese methods use an identical transformation $\theta(\cdot)$ to process target and search inputs $x$ and $z$. These inputs are further combined by some function $g(\cdot)$, i.e. $f(x,z) = g(\theta(x), \theta(z))$. For Siamese tracking, $\theta(\cdot)$ is usually selected to be an embedding function, such as CNN, and $g(\cdot)$ to be a similarity function, such as convolution.

SOT is performed by first initializing the target region $t_0$ with the provided ground truth object location and creating a search region $s_0 = \sigma(t_0)$. Given the frames $F_{\tau-1}$ and $F_\tau$, the previous target and search regions $t_{\tau-1}$ and $s_{\tau-1}$, the target and search regions for the current frame $F_\tau$ are predicted as follows:

$$\begin{aligned} t_\tau &= t_{\tau-1} + \delta(f(\xi_t(t_{\tau-1}), \xi_s(s_{\tau-1}))), \\ s_\tau &= \sigma(t_\tau), \end{aligned} \tag{9}$$

where $\delta(\cdot)$ uses the similarity map, predicted by a model $\theta(\cdot)$ to create an offset between last and current target region positions, $\xi_t(\cdot)$ and $\xi_s(\cdot)$ transform target and search regions into respective voxel pseudo images to be used as inputs for the $\theta(\cdot)$ function. This process is applied for each new frame to find the new target position. We adapt PointPillars' [60] Region Proposal Network (RPN) (Fig. 13) by creating a Feature Generation Network (FGN) as a down-sampling part of RPN, excluding transposed convolutions. The FGN is used as $\theta(\cdot)$ that creates features from the input voxel pseudo-images, which are then compared by a 2D convolution $g(a,b) = \text{conv2D}_{\omega=b}(a)$, where $\omega$ are the weights of the layer, resulting in a similarity map.

The architecture of VPIT is shown in Fig. 14. The input point cloud is voxelized around the region of interest and processed by a Pillar Feature Network to create a voxel pseudo image. The search and target regions are processed by the same FGN module and compared using the convolutional cross-correlation function to create the score map, that is used in post-processing to predict the new target position.

Target and search regions are represented by a 5-dimensional vector $(x, y, w, h, \alpha)$, where $(x, y)$ is the position of the region center in pseudo image space in pixels, $(w, h)$ is the size of the region and $\alpha$ is the rotation angle. The initial ground truth bounding box is described by a 7-dimensional vector $(x, y, z, w, h, d, \alpha)$ with 3D position, size and a rotation angle around the vertical axis. The corresponding target and search regions are created as follows:

$$\begin{aligned} t_0 &= \kappa_c((B^x_{gt}, B^y_{gt}, B^w_{gt}, B^h_{gt}, B^\alpha_{gt})), \\ s_0 &= \sigma(t_0), \end{aligned} \tag{10}$$

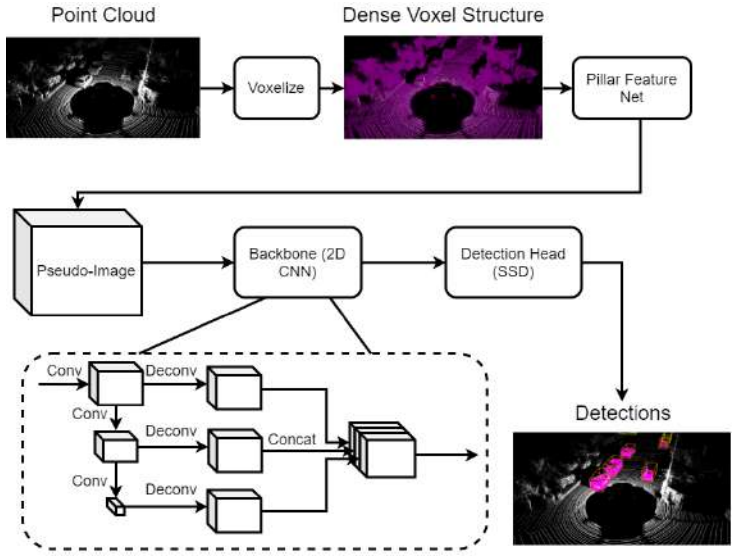



Figure 13: Structure of PointPillars 3D object detection model. The RPN is a 2D CNN that takes a pseudo image as input.

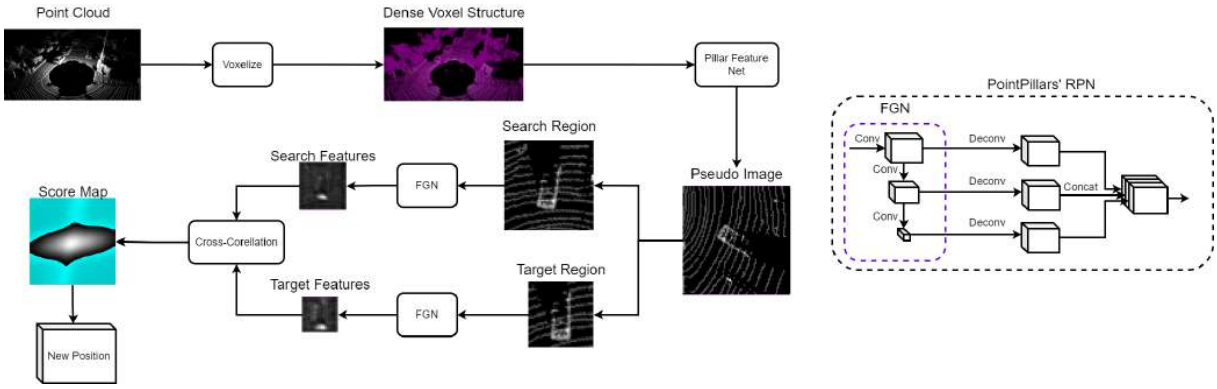



Figure 14: Structure of the proposed Voxel Pseudo Image Tracking model. The input point cloud is voxelized and processed with the PointPillars' Pillar Feature Network to create a voxel pseudo image, which serves as an input to the Siamese model. The Feature Generation Network (FGN), which is a convolutional subnetwork of the PointPillars' RPN, processes the target and search regions to create corresponding features that are then compared to find a position of the best similarity.

where $t_0$ and $s_0$ are the initial target and search region and $\kappa_c(\cdot)$ adds context to the target region

based on the amount of context parameter $c$:

$$\begin{aligned}
&\kappa_c(t) = \begin{cases} \kappa_{c_+}(t), & \text{if } c > 0, \\ \kappa_{c_-}(t), & \text{otherwise,} \end{cases} \\
&\kappa_{c_+}(x,y,w,h,\alpha) = (x,y,m_n,m_n,\alpha), \\
&m_n = \sqrt{(w+m)(h+m)}, \\
&m = c(w+h), \\
&\kappa_{c_-}((x,y,w,h,\alpha)) = (x,y,w(1-c),h(1-c),\alpha),
\end{aligned} \tag{11}$$

where $\kappa_{c_+}(\cdot)$ adds context by creating a square region around the original one, and $\kappa_{c_-}(\cdot)$ increases the dimensions of the original independently. The $\sigma(\cdot)$ function creates a search region, which is $\sigma_s$ times bigger than the target region:

$$\sigma(x,y,w,h,\alpha) = (x,y,\sigma_s w,\sigma_s h,\alpha). \tag{12}$$

The predicted output at frame $\tau$ is computed as:

$$B_\tau = (t_\tau^x, t_\tau^y, B_{gt}^z, t_\tau^w, t_\tau^h, B_{gt}^d, t_\tau^\alpha). \tag{13}$$

Description of the training and inference procedures can be found in the preprint in Appendix 7.4.

### 3.1.4 Performance evaluation

We use KITTI [31] tracking training dataset split to train and test our model, with tracks 0-18 for training and validation and tracks 19-20 for testing (as is common practice [25, 36, 129, 97, 103]). Precision and Success metrics are used to measure the accuracy of a model, as defined in One Pass Evaluation [56]. Precision is describing the difference between ground-truth and predicted object centers in 3D, while Success is computed based on the 3D Intersection over Union (IoU) between predicted and ground truth 3D bounding boxes.

Out of 3 feature blocks in PointPillars' RPN, we use only 1 with 4 layers in a block. We train the model for $64,000$ steps with BCE loss, $1 * 10^{-5}$ learning rate and 2 positive label radius. We create 3 search region rotations with 0.15 step during inference, with 0.98 rotation penalty. A penalty map multiplier of 0.85 is used, a score upscale of 8, original target/search sizes are used, together with context amount of 0.27, rotation interpolation of 1, offset interpolation of 0.3, target feature merge scale of 0.005 and linear search position extrapolation.

Evaluation results on 1080Ti GPU are given in Table 2. VPIT is the fastest method on 1080Ti and achieves competitive Precision and Success values. We evaluate the fastest methods (P2B, PTT, VPIT) on different high-end and embedded GPUs to show how the architecture of devices influences the inference speed. We compute all time spent while processing the inputs, including the pre- and post-processing times, but excluding the data loading time.

As can be seen from Table 3, in terms of speed, VPIT outperforms P2B by 67% on 1080Ti GPU, 99% on 2080 GPU and 86% on 2080Ti GPU with 104-core CPU, but for embedded devices, VPIT outperforms P2B by 135% on TX2 and by 98% on Xavier.

Table 2: Results of 3D Car tracking on KITTI dataset. Modality represents the type of data the tracking is performed on (PC for point cloud, BEV for Birds-Eye-View and VPI for voxel pseudo image). FPS values are reported on a 1080Ti GPU by a corresponding paper. FPS values with a star notation are obtained by running the methods' official implementations on a 1080ti GPU considering the full runtime of the network.

| **Method** | **Modality** | **Success** | **Precision** | **FPS** |
|---|---|---|---|---|
| 3DSRPN PCW [25] | PC | 56.32 | 73.40 | 16.7 |
| 3DSRPN PW [25] | PC | 57.25 | 75.03 | 20.8 |
| SCD3D-KF [36] | PC | 40.09 | 56.17 | 2.2 |
| SCD3D-EX [36] | PC | **76.94** | 81.38 | 1.8 |
| 3D Siam-2D [129] | PC+BEV | 36.3 | 51.0 | - |
| BAT [134] | PC | 65.38 | 78.88 | 23.96* |
| PTT-Net [103] | PC | 67.8 | **81.8** | 39.51* |
| P2B [97] | PC | 56.2 | 72.8 | 30.18* |
| VPIT (Ours) | VPI | 50.49 | 64.53 | **50.45** |

Table 3: Evaluation of the fastest methods for 3D Car tracking on KITTI dataset. The evaluation is performed with official implementations on high-end and embedded GPU platforms with different combinations of GPU/CPU. 32C CPU and 104C CPU correspond to 32-core and 104-core CPUs, respectively.

| **Method** | **Modality** | **Success** | **Precision** | **FPS** | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | 1080Ti 32C CPU | 2080 32C CPU | 2080Ti 104C CPU | TX2 | Xavier |
| P2B [97] | PC | 56.2 | 72.8 | 30.18 | 26.34 | 38.93 | 6.20 | 10.37 |
| PTT-Net [103] | PC | **67.8** | **81.8** | 39.51 | 33.34 | 50.25 | 8.04 | 13.49 |
| VPIT (Ours) | VPI | 50.49 | 64.53 | **50.45** | **52.52** | **72.53** | **14.61** | **20.55** |

**Real-time evaluation** We implemented a real-time evaluation protocol, following [66], to test how the performance of the fastest 3D SOT methods drops when the data processing time is limited and may lead to dropped frames from the Lidar. For a time step $\tau$, only the set of inputs that appeared before $\tau$ is seen to the model: $S_{\text{in}} = (x_i, \tau_i | i \leq \tau)$, where $(x_i, \tau_i)$ is a pair of an input frame and a corresponding time step. The prediction for time step $\tau$ will appear later than $\tau$, as the model inference is not instant, meaning that the output $p_i$ will appear at time $\hat{\tau}_i$ and cannot be compared to the label from $y_i$ frame the same frame, and therefore each label $y_i$ will be compared with the latest available prediction at the corresponding time step $p_{l_{\text{pr}}(i)}$. Given the regular error function $E$, the predictive real-time error is computed as follows:

$$\begin{aligned} E_{\text{pr}}(y_i) &= E(y_i, p_{l_{\text{pr}}(i)}), \\ l_{\text{pr}}(i) &= \underset{j}{argmax}\ \hat{\tau}_j \leq \tau_i. \end{aligned} \tag{14}$$

As shown in [66], some techniques can be applied to existing models to counter the time difference between input and prediction, increasing the real-time evaluation score. To eliminate

Table 4: Evaluation of the fastest methods for 3D Car tracking on KITTI dataset in real-time settings without the predictive requirement. The evaluation is performed with official implementations on embedded GPU platforms for 10 and 20 Hz Lidars (Data FPS). Frame drop represents the percentage of frames that could not be processed due to the model's latency. Regular represents evaluation without real-time requirements.

| **Method** | **Data FPS** | **Success (non-predictive)** | | | **Precision (non-predictive)** | | | **FPS** | | **Frame drop** | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Regular | TX2 | Xavier | Regular | TX2 | Xavier | TX2 | Xavier | TX2 | Xavier |
| P2B [97] | 10 | 56.20 | 21.90 | 36.50 | 72.80 | 21.70 | 42.40 | 6.17 | 10.07 | 37.41% | 7.00% |
| PTT-Net [103] | 10 | **67.80** | 29.50 | **63.60** | **81.80** | 30.00 | **75.10** | 6.90 | 12.38 | 28.21% | 0.81% |
| VPIT (Ours) | 10 | 50.49 | **50.31** | 50.49 | 64.53 | **64.08** | 64.53 | **14.31** | **20.55** | **0.68%** | **0.00%** |
| P2B [97] | 20 | 56.20 | 10.90 | 16.70 | 72.80 | 7.90 | 15.0 | 5.54 | 9.61 | 70.57% | 51.56% |
| PTT-Net [103] | 20 | **67.80** | 17.90 | 26.50 | **81.80** | 15.70 | 26.60 | 6.61 | 11.88 | 64.14% | 38.95% |
| VPIT (Ours) | 20 | 50.49 | **38.96** | **47.70** | 64.53 | **45.50** | **59.87** | **14.37** | **20.06** | **30.17%** | **2.38%** |

Table 5: Evaluation of the fastest methods for 3D Car tracking on KITTI dataset in real-time settings with the predictive requirement. The evaluation is performed with official implementations on embedded GPU platforms for 10 and 20 Hz Lidars (Data FPS). Frame drop represents the percentage of frames that could not be processed due to the model's latency. Regular represents evaluation without real-time requirements

| **Method** | **Data FPS** | **Success (predictive)** | | | **Precision (predictive)** | | | **FPS** | | **Frame drop** | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Regular | TX2 | Xavier | Regular | TX2 | Xavier | TX2 | Xavier | TX2 | Xavier |
| P2B [97] | 10 | 56.20 | 19.10 | 30.30 | 72.80 | 17.80 | 33.70 | 6.17 | 10.07 | 36.31% | 6.79% |
| PTT-Net [103] | 10 | **67.80** | 24.90 | **50.70** | **81.80** | 23.40 | 59.00 | 7.07 | 12.26 | 26.15% | 0.90% |
| VPIT (Ours) | 10 | 50.49 | **45.82** | 46.68 | 64.53 | **57.76** | **59.28** | **14.61** | **20.55** | **0.57%** | **0.00%** |
| P2B [97] | 20 | 56.20 | 9.10 | 13.50 | 72.80 | 6.00 | 10.90 | 5.54 | 9.47 | 69.57% | 50.31% |
| PTT-Net [103] | 20 | **67.80** | 15.40 | 23.20 | **81.80** | 13.10 | 21.50 | 6.67 | 12.06 | 62.49% | 37.35% |
| VPIT (Ours) | 20 | 50.49 | **34.00** | **41.39** | 64.53 | **36.61** | **49.36** | **14.40** | **20.77** | **29.41%** | **1.56%** |

the factor of "predictive errors", we introduce a non-predictive benchmark where the labels are available one frame after the inputs:

$$l_{\text{npr}}(i) = \underset{j}{argmax}\ \hat{\tau}_j \leq \tau_{i+1}. \tag{15}$$

Table 4 contains evaluation results of P2B, PTT and VPIT on embedded devices for the non-predictive benchmark and Table 5 contains results for the predictive benchmark. Data FPS is selected to represent the most popular Lidars of 10 and 20 Hz, with 10 Hz evaluation on Xavier being the easiest case and 20 Hz on TX2 the hardest one.

VPIT is the only method that did not suffer from frame drop on the easiest case. The highest frame drop is suffered by P2B and PTT on the hardest case with $60-70\%$ of frames not processed, resulting in 6 and 4 times worse results than during the regular evaluation, respectively, meaning that these methods could not work under real-time conditions on TX2 with a 20 Hz lidar. As shown in Fig. 15, Success drop of VPIT is the smallest, but other methods loose most of their tracking ability under real-time evaluation conditions.

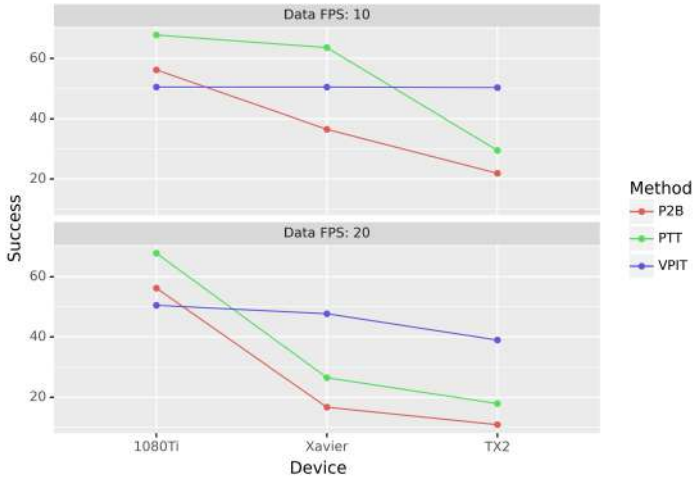



Figure 15: Evaluation of the fastest models with real-time requirements on embedded devices and a desktop GPU. Data FPS represents 10 and 20 Hz Lidars. Devices are sorted in descending order by their computational power.

## 3.2 Speed up of 3D Object Detection models

### 3.2.1 Introduction and objectives

To reach the desired inference speed of 3D detection methods, we deploy several post-training optimization techniques, including ONNX optimization and model pruning. ONNX optimization is performed by representing the model in ONNX format that is used for inference only and usually leads to faster inference than using the general-purpose frameworks, such as PyTorch. Model pruning is the procedure of removing model weights or neurons iteratively to reduce the number of parameters and possibly increase the inference speed, while aiming to suffer little to no accuracy of the model.

Pruning can be performed in an unstructured way by removing specific model parameters without any restrictions, or in a structured way by removing parameters corresponding to neurons or channels. While the former can greatly reduce the number of model parameters, it rarely increases inference speed. This is because, when sparse matrix multiplication packages are not used, data processing on GPU will mostly remain the same. Structured pruning, if performed on channels, can directly reduce the inference time, as it leads to a "thinner" model with reduced computations having the standard structure of layers included in most of the programming libraries. Selection of the weights or channels that can be pruned can be done in different ways, with magnitude-based criteria being among the most widely used, where the weights with the lowest n-norm are considered less important and are pruned first.

### 3.2.2 Description of work performed so far

We follow [42] and implement channel pruning of PointPillars and FairMOT. We prune $10-20\%$ of channels in each layer and fine-tune for $5-15$ epochs after each pruning step, computing the speed and the accuracy of the model after each fine-tuning step. We found the best results using 10% pruning and fine-tuning for 10 epochs on each iteration for 10 iterations for PointPillars. We implemented the results of [84] and apply pruned and original models on the

near sub-scene setup.

Tables 6 and 7 show that PointPillars achieves 1.9× speed-up on NVIDIA Xavier with 14% mAP loss when using pruning on near depth zone, resulting in inference speed of 16 FPS, while it achieves 11 FPS with pruning only and 8% mAP loss. The speed-up on NVIDIA TX2 is 2.7× with 7 FPS for a pruned model on the near depth zone. The resulting models operate in real-time for the 10 Hz Lidars on Xavier.

Table 6: Evaluation of PointPillars models speed and accuracy with pruning and near depth zones on NVIDIA Xavier.

| **Method** | **Pruning** | **Near** | **mAP** | **FPS** |
|---|---|---|---|---|
| PointPillars | | | 77 | 8.40 |
| PointPillars | ✓ | ✓ | 66 | 16.00 |
| PointPillars | ✓ | | 71 | 11.50 |
| PointPillars | | ✓ | 69 | 9.66 |

Table 7: Evaluation of PointPillars models speed and accuracy with pruning and near depth zones on NVIDIA TX2.

| **Method** | **Pruning** | **Near** | **mAP** | **FPS** |
|---|---|---|---|---|
| PointPillars | | | 77 | 2.55 |
| PointPillars | ✓ | ✓ | 66 | 7.03 |
| PointPillars | ✓ | | 71 | 5.11 |
| PointPillars | | ✓ | 69 | 4.90 |

## 3.3 3D Multi-Object Tracking Using Graph Neural Networks With Cross-Edge Modality Attention

### 3.3.1 Introduction and objectives

3D multi-object tracking (MOT) is an essential component of the scene understanding pipeline of autonomous robots. It aims at inferring associations between occurrences of object instances at different time steps in order to predict plausible 3D trajectories. These trajectories are then used in various downstream tasks such as trajectory prediction [100] and navigation [78]. In this section, we present Batch3DMOT, an offline 3D tracking framework that follows the tracking-by-detection paradigm and utilizes multiple sensor modalities (camera, LiDAR, radar) to solve a multi-frame, multi-object tracking objective. Sets of 3D object detections per frame are first turned into attributed nodes. In order to learn offline 3D tracking, we employ a graph neural network (GNN) that performs time-aware neural message passing with intermediate frame-wise attention-weighted neighborhood convolutions.

Our main contributions can be summarized as follows:

- A novel multimodal GNN framework for offline 3D multi-object tracking on multi-category tracking graphs including k-NN neighborhood attention across semantic graph components.
- A cross-edge attention mechanism that uses intermittent sensor data to substantiate the differentiation between active and inactive edges.
- Methodology and pre-processing pipeline for constructing category-disjoint tracking graphs over multiple timesteps as well as a novel agglomerative trajectory clustering scheme for effective trajectory generation.

A summary of this work is provided hereafter. The corresponding paper is referenced below and can be found in Appendix 7.5:

- [12] M. Büchner and A. Valada, "3D Multi-Object Tracking Using Graph Neural Networks With Cross-Edge Modality Attention", *IEEE Robotics and Automation Letters (RA-L)*, vol. 7, no. 4, pp. 9707-9714, 2022.

### 3.3.2 Summary of state of the art

Multi-object tracking (MOT) can be categorized into online and offline settings. Whereas online methods are limited to using past and current data, offline methods can efficiently leverage future data to find solutions to the global data association problem. Besides a temporal categorization, MOT can be applied in the 2D [131, 46, 111] or the 3D [51, 119, 124, 28] domain, exploiting either 2D or 3D object detections. Finally, two commonly followed approaches involve the tracking-by-detection paradigm [131, 11, 124] and joint object detection and tracking [46]. 2D MOT is an already well-studied research area, where both online and offline methods have been jointly evaluated on a single benchmark [23]. Most offline methods formulate 2D MOT as a graph association problem solved using optimization techniques from graph and network theory [115, 128, 108, 45]. On the other hand, 3D MOT datasets are more challenging since they involve intricate sensor motion and significantly smaller frame rates [27, 32, 23]. Recent state-of-the-art approaches [120, 127] facilitate graph neural networks to capture higher-order artifacts on graph structures. OGR3MOT [127] follows NMPTrack [11] in using neural message passing but solves the online 3DMOT problem while leveraging Kalman state predictions for improved track representations. Our approach differs from NMPTrack [11] by introducing a novel modality and node representation scheme relevant for 3D tracking and a novel agglomerative trajectory clustering scheme that yields higher recall rates and fewer false positives. Compared to OGR3MOT [127], we include multiple sensor modalities and model trajectories based on object similarity instead of exploiting Kalman filters for predictive track representation in online tracking.

### 3.3.3 Description of work performed so far

Following the tracking-by-detection paradigm, we turn a set of detections per frame $O_t = \{o_1, ..., o_n\}$ into nodes on a directed acyclic graph $G = (V, E)$ that holds an ordered set of frames. Instead of using detection edges, we follow the approach of Brasó *et al.* [11] in collapsing them. Since our approach involves learning on graph-structured data, both nodes and edges are attributed. In the subsequent section, we first outline the approach for extracting features from the various modalities, followed by the graph construction process. We then present the

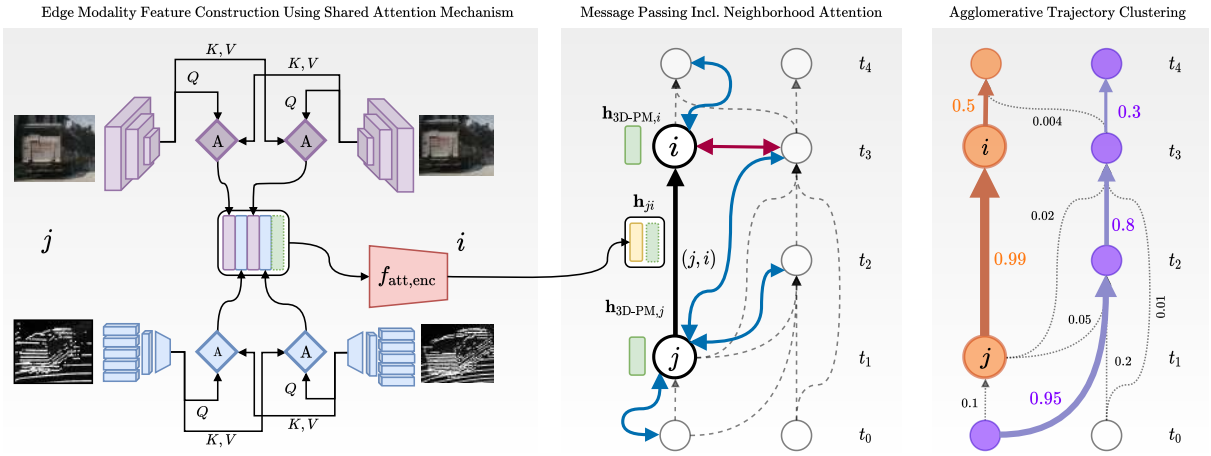



Figure 16: Overview of our Batch3DMOT architecture.

graph neural network architecture and the graph traversal algorithm that yields the final trajectories. More detailed explanations can be found in Appendix 7.5. An overview of our modal architecture can be seen in Figure 16.

**Feature Representation**: In typical tracking scenarios such as autonomous driving, we are confronted with a multitude of sensor modalities such as camera, LiDAR, radar, or even thermal images. While the detections are often derived only from a single sensor modality such as LiDAR or camera, the entirety of modalities can still be utilized for improved similarity finding of detections in the tracking task. Our approach fuses 3D pose & motion features (3D-PM) from bounding boxes with 2D as well as 3D appearance features from (surround) cameras (2D-A), LiDAR (3D-AL) as well as radar sensors (3D-R). Different from tracking in the image plane, 3D bounding box information essentially represents a more discriminative feature in 3D tracking due to available depth information [120]. Most importantly, this simplifies re-association after false negatives (FN) generated by occlusions or missed detections but also eases the identification of false positives (FP). Instead of solely exploiting bounding box information in terms of relative node differences for an initial edge feature [11], the 3D-PM feature constitutes the primary node feature in the proposed approach.

**Graph Construction**: The chosen approach arranges nodes on a tracking graph over 5 frames in a sliding window manner with a stride of 1. The tracking performance improves drastically when learning the tracking task with actual detections instead of ground truth annotations since FP filtering and FN handling pose major challenges in real-world tracking. Consequently, ground truth annotation identifiers need to be paired with actual detections in order to construct edge labels for the learning stage. Under the assumption that ground truth annotations generally do not show significant intra-category overlap, we match detection results to geometrically close annotations in the birds-eye view (BEV).

**Message Passing Graph Neural Network**: This work employs the principle of time-aware neural message passing [11], which is extended to allow information exchange between inter-category nodes that reside in particular disconnected graph components. In addition, we present a novel way to include intermittent sensor modalities across edges.

**Inference and Graph Traversal**: The outputs of the GNN architecture are Sigmoid-valued scores that represent whether an edge is likely to be active/inactive. Instead of thresholding at

an edge score of 0.5 to find active/inactive edges to turn into trajectories, we follow the spirit of *ByteTrack* [131] and try to associate (nearly) *every* detection with a preliminary trajectory. Based on the assumption that the predicted edge scores show some inherent order, i.e., FP edges exhibit lower scores than TP edges within local neighborhoods of the graph, we propose a score-based agglomerative trajectory clustering paradigm (Algorithm 2). The edge score predictions of multiple overlapping batches are averaged per edge. All edges are arranged in descending order and empty (ordered) clusters are initialized that will later hold output trajectories. As shown in Algorithm 2, we loop through all edges from the highest to lowest score and check whether the edge is constrained or unconstrained. If constrained, it is checked whether the edge would essentially add time-wise leading or trailing nodes to one of the temporary clusters or if it joins two clusters. In the case of joining two clusters, an additional score-wise threshold needs to be met. Otherwise, the edge does not violate any tracking constraints and a new cluster is initialized.

**Algorithm 2** Agglomerative Trajectory Clustering.

$E_{pred}, N_{meta} \leftarrow$ CombineBatches($GNN(\mathbf{X}, \mathbf{X}_e)$)
$E^*_{pred} \leftarrow$ DescSortEdgesByScore($e_{pred}$)
$vis \leftarrow$ CreateVisitedNodesDict()
$\mathscr{C} \leftarrow$ CreateEmptyClustersDict()
**for** $e_{ji}, score$ in $E^*_{pred}$ **do**
    **if** $j \notin vis$ and $i \notin vis$ **then**
        $\mathscr{C} \leftarrow$ CreateNewCluster($e_{ji}$)
        UpdateVisitedNodes($e_{ji}, \mathscr{C}$)
    **else**
        **if** $j \notin vis$ and $i \in vis$ **then**
            **if** $i$ is leading node in C($i$) **then**
                $\mathscr{C} \leftarrow$ AddToCluster($e_{ji}$)
                $vis \leftarrow$ UpdateVisitedNodes($e_{ji}, \mathscr{C}$)
            **end if**
        **else if** $j \in vis$ and $i \notin vis$ **then**
            **if** $j$ is trailing node in $\mathscr{C}(j)$ **then**
                $\mathscr{C} \leftarrow$ AddToCluster($e_{ji}$)
                $vis \leftarrow$ UpdateVisitedNodes($e_{ji}, \mathscr{C}$)
            **end if**
        **else if** $j \in vis$ and $i \in vis$ **then**
            **if** $j$ is trailing $\mathscr{C}(j)$ and $i$ is leading $\mathscr{C}(i)$ **then**
                $\mathscr{C} \leftarrow$ JoinClusters($e_{ji}$)
                $vis \leftarrow$ UpdateVisitedNodes($e_{ji}, \mathscr{C}$)
            **end if**
        **end if**
    **end if**
**end forreturn** TurnClustersIntoTrajectories($\mathscr{C}$)

Table 8: Comparison of AMOTA scores on the nuScenes validation set. Bold/underlined numbers denote **best**/second best model scores, respectively.

| Method | Overall | Bicyc. | Bus | Car | Moto. | Ped. | Trailer | Truck |
|---|---|---|---|---|---|---|---|---|
| *AB3DMOT [119][135]* | 0.179 | 0.09 | 0.489 | 0.36 | 0.051 | 0.091 | 0.111 | 0.142 |
| *Prob3DMOT [21][135]* | 0.561 | 0.272 | 0.741 | 0.735 | 0.506 | 0.755 | 0.337 | 0.580 |
| CenterPoint [124] | 0.665 | 0.458 | 0.801 | 0.842 | 0.615 | 0.777 | 0.504 | 0.656 |
| ProbMM-3DMOT [57] | 0.687 | 0.490 | 0.820 | 0.843 | 0.702 | 0.766 | **0.534** | 0.654 |
| 3D-PM-MEGVII[135] | 0.623 | 0.368 | 0.759 | 0.789 | 0.655 | 0.796 | 0.378 | 0.617 |
| 3D-PM-CP [124] | 0.708 | 0.540 | 0.837 | 0.849 | 0.728 | 0.813 | 0.497 | 0.689 |
| 3D-PM-C-CP [124] | 0.709 | 0.542 | 0.837 | **0.851** | 0.733 | 0.813 | 0.502 | 0.688 |
| 3D-PM-CL-CP [124] | **0.715** | 0.540 | **0.855** | **0.851** | **0.748** | **0.821** | 0.493 | 0.695 |
| 3D-PM-CLR-CP [124] | 0.713 | **0.545** | 0.851 | 0.850 | 0.736 | 0.820 | 0.494 | **0.696** |

### 3.3.4 Performance evaluation

In this section, we present quantitative and qualitative evaluations of our proposed Batch3DMOT on the nuScenes [27] and KITTI [32] datasets using the average multiple-object tracking accuracy (AMOTA) and multiple-object tracking accuracy (MOTA) metrics, respectively. Similar to existing methods, we evaluate our model on the nuScenes test set as well as the KITTI 2D MOT benchmark. One can see a comparison of AMOTA scores on the nuScenes in Table 8. The rest of the experimental results can be found in Appendix 7.5.

*Detections and GT Matching*: In this approach, we use the detections provided by MEGVII [135] and CenterPoint [124] for nuScenes. On the KITTI dataset, we use Point-RCNN detections [104] as also used by FG3DMOT [96] and AB3DMOT [119]. We match detections to ground truth trajectory labels to obtain identifiers. As proposed earlier [135, 27], the L2 center distance is often used for matching, which is beneficial for faraway objects. Our empirical findings show that especially large objects suffer from this heuristic, e.g., their respective length is not predicted correctly, which leads to considerable object center translations and effectively renders the L2 distance uninformative. Therefore, we follow a bi-level approach by first selecting a close radius (L2) and then checking whether detection and annotation exhibit a significant IoU overlap in the Bird's-Eye-View.

### 3.3.5 Future Work

In this section, we proposed a framework for addressing the offline 3D MOT task using a multi-modal graph neural network including a novel agglomerative trajectory construction scheme. Our method was able to improve tracking accuracy and demonstrate enhanced false positive filtering, compared to current online methods, using the same detections. In the future, we plan to extend our approach to also cope with long-term occlusions.

# 4 Deep SLAM and 3D scene reconstruction

## 4.1 PADLoC: LiDAR-Based Deep Loop Closure Detection and Registration using Panoptic Attention

### 4.1.1 Introduction and objectives

One of the essential tasks in autonomous robotics is Simultaneous Localization and Mapping (SLAM). Many SLAM systems have been proposed for different sensor modalities including cameras [112] and LiDARs [65]. While vision-based methods fail in challenging lighting conditions such as illumination changes, LiDAR-based approaches are more robust to such alterations and provide a more accurate representation of the environment. In this work, we address the joint problem of loop closure detection and map registration for LiDAR-based SLAM. Using the inspiration from semantic mapping approaches [19, 101] and methods that exploit panoptic information for vision-based loop closure detection [126], we leverage panoptic segmentation of point clouds in this work. We propose a more versatile approach, which only requires panoptic labels during training. Furthermore, we demonstrate that the proposed PADLoC approach achieves state-of-the-art performance, compared to the existing methods, on three well-established datasets.

The main contributions of this work are as follows:

1. We propose PADLoC, a transformer encoder architecture for point cloud matching and registration. Unlike existing methods, we use separate inputs as keys, values, and queries effectively, exploiting the transformer structure.

2. We define a novel loss function that leverages panoptic information for registration. We further propose formulating both geometric and panoptic registration losses as bidirectional functions that greatly improve performance.

3. We study the effect of multiple weighting methods in SVD to enhance point matching.

A summary of this work is provided hereafter. The corresponding paper is referenced below and can be found in Appendix 7.6:

- [2] J. Arce, N. Vödisch, D. Cattaneo, W. Burgard, and A. Valada, "PADLoC: LiDAR-Based Deep Loop Closure Detection and Registration using Panoptic Attention", *arXiv: 2209.09699*, 2022.

### 4.1.2 Summary of state of the art

Only a handful of works have proposed to leverage semantic information for large-scale mapping and localization [19, 3], and particularly for loop closure detection. Based on semantic segmentation, SuMa++ [19] filters dynamic objects from a LiDAR-based map and extends the ICP algorithm with additional semantic constraints. While SuMa++ does not utilize semantic information for loop closure detection, RINet [64] explicitly addresses LiDAR-based place recognition via a rotation invariant global descriptor combining semantic and geometric information. For the same task, Kong *et al.* [54] propose to build a graph representation of point clouds, which are enriched by both semantic and instance segmentation and perform graph similarity matching. SA-LOAM [65] integrates a semantic-aided variant of ICP into the popular LOAM pipeline for point cloud registration. To address loop closure, it uses a similar

graph representation as Kong *et al.* [54]. SV-Loop [126] is a loop closure detection method for vision-based SLAM. It separately proposes loop closure candidates based on raw images and panoptic segmentation maps, which are then fused to extract the most feasible candidates. In our approach, we exploit panoptic annotations of point clouds while predicting both loop closure detection and point cloud registration. Additionally, we only utilize them during the training process but not for deployment, making the method more versatile.

### 4.1.3 Description of work performed so far

Our novel PADLoC architecture is built upon our previously proposed LCDNet [14], where instead of using a differentiable approximation of the optimal transport to obtain point matches, we propose to leverage the cross-attention matrices of transformers. The overview of the architecture can be seen in Figure 17.

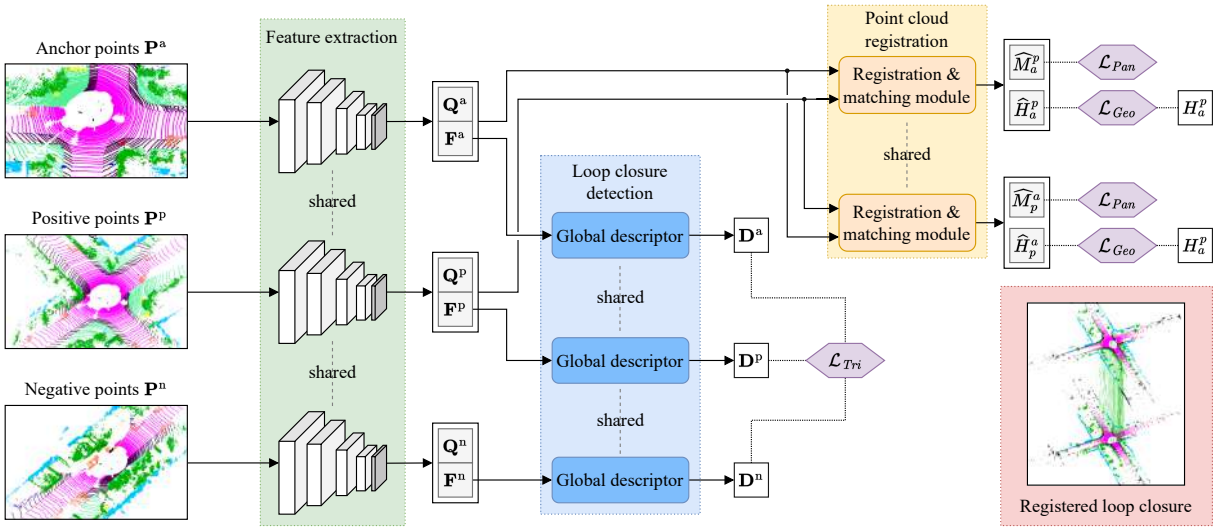



Figure 17: Overview of our proposed PADLoC architecture for joint loop closure detection and point cloud registration. It consists of a shared feature extractor (green) followed by a global descriptor head (blue) for loop closure detection and a registration and matching module (orange) to estimate the 6-DoF transform between two point clouds (red).

**Loop Closure Detection:** The global descriptor module of PADLoC further encodes the previously extracted features to perform loop closure detection. For this task, we employ the NetVLAD layer [1] to convert the feature vectors **F** of the anchor, the positive, and the negative points to their respective final descriptor **D**.

**Point Matching:** The matching module predicts soft correspondences between keypoints of a source point cloud and a target point cloud. Additionally, it outputs projected target coordinates which are linear combinations of the original target coordinates with a one-to-one pairing with the points of the source set and a confidence weight for each of these matches. Inspired by the success of transformers in related tasks, we propose a novel architecture that performs cross-attention directly on the encoder part, obviating the need for a decoder by feeding independent inputs for the queries, keys, and values.

**Point Cloud Registration:** To obtain the final relative transformation from a source point cloud to a target point cloud, we perform a weighted version of the Kabsch-Umeyama algorithm that

Table 9: Comparison of loop closure detection and point cloud registration performance

| | | KITTI Seq. 08 [33] | | | Ford Seq. 01 [92] | | |
|---|---|---|---|---|---|---|---|
| | Method | AP ↑ | $r_{err}$ [°] ↓ | $t_{err}$ [m] ↓ | AP ↑ | $r_{err}$ [°] ↓ | $t_{err}$ [m] ↓ |
| Handcrafted | M2DP [41] | 0.05 | — | — | 0.89 | — | — |
| | Scan Context* [52] | 0.65 | 3.11 | — | <u>0.97</u> | 16.68 | — |
| | LiDAR-Iris* [117] | 0.64 | <u>1.84</u> | — | 0.90 | <u>1.66</u> | — |
| | ISC* [113] | 0.31 | 6.27 | — | 0.62 | 6.15 | — |
| | ICP (pt2pt) [132] | — | 160.63 | 2.41 | — | 9.56 | 2.79 |
| | ICP (pt2pl) [132] | — | 160.73 | 2.49 | — | 9.16 | 2.62 |
| Learning | DCP [116] | — | 46.06 | 2.59 | — | 12.14 | 3.42 |
| | OverlapNet* [18] | 0.32 | 65.45 | — | 0.79 | 9.44 | — |
| | LCDNet [14] | <u>0.76</u> | **0.37** | <u>0.19</u> | <u>0.97</u> | 1.82 | <u>1.44</u> |
| | PADLoC (ours) | **0.81** | **0.37** | **0.16** | **0.98** | **1.50** | <u>1.41</u> |

Comparison of the average precision (AP) for loop closure detection as well as rotation error $r_{err}$ and translation error $t_{err}$ for point cloud registration of PADLoC with previous methods. All learning-based models are trained on the KITTI odometry benchmark dataset. PADLoC uses panoptic annotations from the SemanticKITTI dataset. Methods denoted with * only estimate the yaw between two point clouds instead of a full 6-DoF transformation. Bold and underlined values denote the best and second best scores, respectively.

finds the optimal translation and rotation between two sets of points by minimizing the root mean square error of the point pairs.

**Loss Functions:** Our total loss function consists of a weighted sum of the triplet loss for loop closure detection as well as a geometric loss and the newly proposed panoptic loss for point cloud registration. The triplet loss enforces a small distance between the descriptors of an anchor point cloud and a positive point cloud, i.e., a LiDAR scan of a true loop closure, while increasing the distance between the descriptors of the anchor and a negative point cloud, i.e., a LiDAR scan taken at a different place. We formulate our geometric loss, which compares the predicted relative transformation from the anchor to the positive sample with the ground truth transformation, and an auxiliary matching loss, where we transform the anchor points with the ground truth transformation and project the positive sample. We also formulate a novel panoptic loss as the sum of semantic misclassification losses and the multi-matched objective loss. For the misclassification losses, we treat the matching process as a classification task, where the projected positive points are assigned a semantic class. In our novel multi-matched object loss, we further exploit the instance labels to encourage the network to match entire objects consistently from one point cloud to the other. Furthermore, in our method, we further exploit the instance labels to encourage the network to match entire objects consistently from one point cloud to the other. This is done by penalizing matches of points from a single object in the anchor to multiple objects in the positive sample.

### 4.1.4 Performance evaluation

In order to quantitatively evaluate the performance of our proposed approach, we compute the average precision (AP), the rotation error in degrees, and the translation error in meters. We perform experiments on two publicly available autonomous driving datasets, namely the KITTI odometry benchmark [33] and the Ford campus vision and LiDAR dataset [92]. Additionally, we also present results on a more challenging in-house dataset recorded in Freiburg, Germany.

We compare our proposed method with three of the most recent learning-based methods for loop closure detection namely LCDNet [14], OverlapNet [18], and Deep Closest Point (DCP) [116], as well as previously proposed handcrafted methods M2DP [41], Intensity Scan Context (ISC) [113], Scan Context [52], and LiDAR-Iris [117]. For DCP, we combine the feature extraction module of PADLoC with a full transformer-based matching module based on the authors' code release. For the other methods, we directly use the official code published by the respective authors.

In Table 9, we show the average precision, rotation and translation errors for PADLoC and the baselines. Overall, our method is able to effectively address all opposing challenges as listed in the previous section. Furthermore, it is also shown that PADLoC reaches the highest average precision and lowest translation/rotation errors for the KITTI odometry benchmark [33] and the Ford campus vision and LiDAR dataset [92] compared to previous methods.

### 4.1.5 Future Work

In this work, we proposed the novel PADLoC architecture that is composed of a common feature extractor, a global descriptor as well as a transformer-based registration and matching module. Unlike previous approaches, we feed different inputs as value, query, and key to the transformer encoder exploiting its internal structure. We further introduced a new loss function that leverages ground truth panoptic annotations by penalizing matching points from different semantic classes as well as across multiple objects and validated its positive impact. Through extensive experimental evaluations, we demonstrated the efficacy of PADLoC compared to both handcrafted and learning-based methods. Future work will focus on exploiting panoptic information in an online manner and applying the matching approach of PADLoC to point cloud registration tasks in other domains.

## 4.2 Continual SLAM: Beyond Lifelong Simultaneous Localization and Mapping through Continual Learning

### 4.2.1 Introduction and objectives

An essential task for an autonomous robot, deployed in the open world without prior knowledge about its environment, is to perform SLAM. While classical methods typically rely on handcrafted low-level features that tend to fail under challenging conditions, deep learning-based approaches mitigate such problems due to their ability to learn high-level features. However, they lack the ability to generalize to out-of-distribution data, with respect to the training set. For visual SLAM, such out-of-distribution data can correspond to images sourced from cities in different countries or under substantially different conditions. In the following, we use the term *environment* to refer to a bounded geographical area. While different environments can share the same fundamental structure, e.g., urban areas, their specific characteristics prevent the seamless transfer of learned features, resulting in a domain gap between cities.

In the context of this work, lifelong SLAM considers the long-term operation of a robot in a single environment (see Figure 18). Although this environment can be subject to temporal changes, the robot is constrained to stay within the area borders [55]. Recent works attempt to relax this assumption by leveraging domain adaptation techniques for deep neural networks [133, 67, 68, 72]. While a naive solution for adapting to a new environment is to source additional data, this is not feasible when the goal is to ensure the uninterrupted operation of the robot. Moreover, changes within an environment can be sudden, and data collection and

annotation often come at a high cost. Therefore, adaptation methods should be trainable in an unsupervised or self-supervised manner without the need for ground truth data. As illustrated in Figure 18, the setting addressed in domain adaptation only considers unidirectional knowledge transfer from a single known to a single unknown environment [5] and thus does not represent the open world, where the number of new environments that a robot can encounter is infinite and previously seen environments can be revisited. To address this gap, we take the next step by considering more complex sequences of environments and formulate the novel task of continual SLAM that leverages insights from both continual learning (CL) and lifelong SLAM. We propose a dual-network architecture called CL-SLAM to balance adaptation to new environments and memory retention of preceding environments.

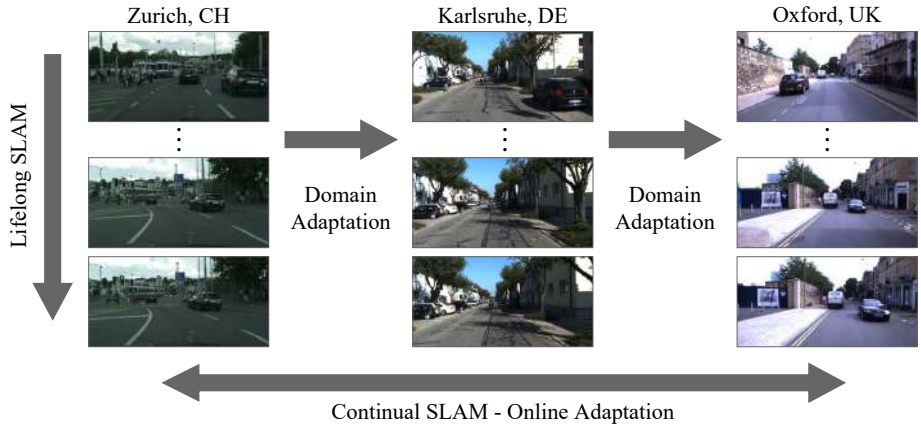



Figure 18: While lifelong SLAM considers the long-term operation of a robot in a single dynamically changing environment, domain adaptation techniques aim toward transferring knowledge gained in one environment to another environment. The newly defined task of continual SLAM extends both settings by requiring omnidirectional adaptation for multiple environments. Agents have to both quickly adapt to new environments and effectively recall knowledge from previously visited environments.

A summary of this work is provided hereafter. The corresponding paper is referenced below and can be found in Appendix 7.7:

- [112] N. Vödisch, D. Cattaneo, W. Burgard, and A. Valada, "Continual SLAM: Beyond Lifelong Simultaneous Localization and Mapping through Continual Learning", *International Symposium of Robotics Research (ISRR)*, 2022.

### 4.2.2 Summary of state of the art

Recently, Luo *et al.* [72] employed a subtask of CL for self-supervised visual odometry and depth estimation, opening a new avenue of research. Online adaptation enables these methods to enhance the trajectory and depth prediction on a test set sourced from a different data distribution than the originally used training set. Both Zhang *et al.* [133] and CoMoDA [59] primarily target the depth estimation task. While Zhang *et al.* propose to learn an adapter to map the distribution of the online data to the one of the training data, CoMoDA updates the internal parameters of the depth and pose networks based on online data and a replay buffer. The work in spirit most similar to ours is done by Li *et al.* [67]. They propose to substitute the standard convolutional layers in the depth and pose networks with convolutional LSTM variants. Then,

the model parameters are continuously updated using only the online data. In subsequent work, Li *et al.* [68] replace the learnable pose network by point matching from optical flow. Note that all existing works purely focus on one-step adaptation, i.e., transferring knowledge gained in one environment to a single new environment. In this paper, we introduce continual SLAM to take the next step by considering more complex deployment scenarios comprising more than two environments and further alternating between them.

### 4.2.3 Description of work performed so far

**Continual SLAM:** As motivated in the introduction, the newly formulated problem of continual SLAM addresses the domain gap caused by deploying a SLAM system to the open world. Ideally, a method addressing this problem should be able to achieve the following goals: 1) quickly adapt to unseen environments while in deployment, 2) leverage knowledge from previously seen environments to speed up the adaptation, and 3) effectively memorize knowledge from previously seen environments to minimize the required adaptation when revisiting them, while mitigating overfitting to any of the environments. Formally, continual SLAM can be defined as a potentially infinite sequence of scenes $\mathscr{S} = (s_1 \rightarrow s_2 \rightarrow \dots)$ from a set of different environments $s_i \in \{E_a, E_b, \dots\}$, where $s$ denotes a scene and $E$ denotes an environment. For a more complete formal definition along with details of our newly proposed metrics, the adaptation quality (AQ) and the retention quality (RQ), we refer to the full paper.

To conclude, we identify the following main challenges: 1) large number of different environments, 2) huge number of chained scenes, 3) scenes can occur in any possible order, and 4) environments can contain multiple scenes. Therefore, following the spirit of continual learning (CL), a continual SLAM algorithm has to balance between short-term adaptation to the current scene and long-term knowledge retention. This trade-off is also commonly referred to as avoiding catastrophic forgetting with respect to previous tasks without sacrificing performance on the new task at hand.

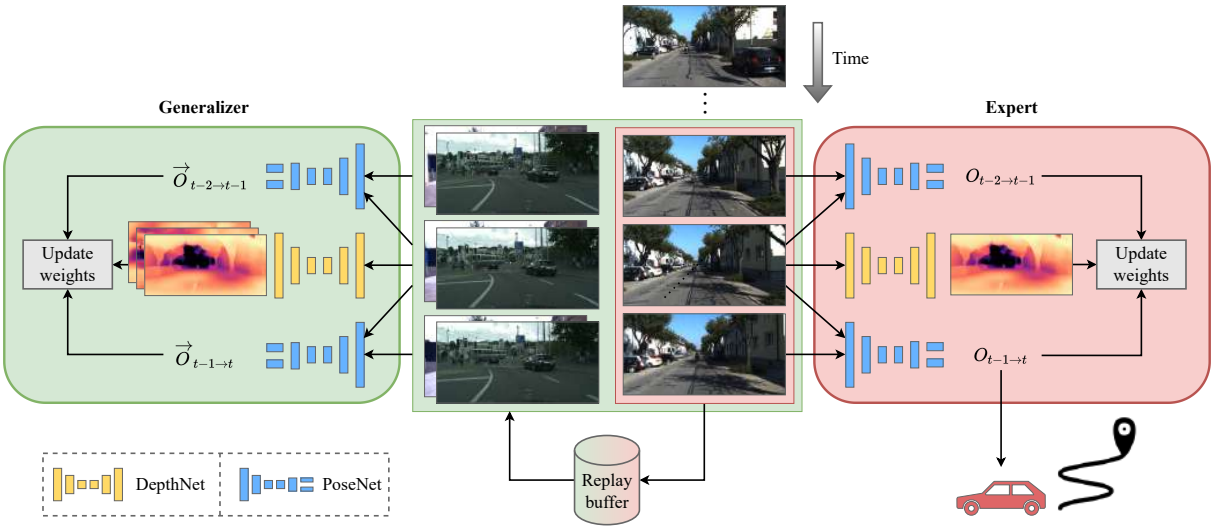



Figure 19: Online adaptation scheme of our proposed CL-SLAM that is constructed as a dual-network architecture including a generalizer (left) and an expert (right). While the expert focuses on the short-term adaptation to the current scene, the generalizer avoids catastrophic forgetting by employing a replay buffer comprising samples from the past and the present.

Table 10: Translation and rotation error for computing the AQ and RQ metrics

| Used for | Previous scenes | Current scene | $\mathscr{B}_{fixed}$ | | $\mathscr{B}_{expert}$ | | $\mathscr{B}_{general}$ | | CL-SLAM | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | $t_{err}$ | $r_{err}$ | $t_{err}$ | $r_{err}$ | $t_{err}$ | $r_{err}$ | $t_{err}$ | $r_{err}$ |
| AQ | $c_t$ | $k_1$ | 130.74 | 26.35 | **2.50** | **0.37** | 7.21 | 1.26 | **2.50** | **0.37** |
| | $c_t$ | $r_1$ | 170.76 | 13.37 | **28.94** | 5.63 | 29.05 | **5.49** | **28.94** | 5.63 |
| | $c_t \rightarrow r_1$ | $k_1$ | – | – | 3.66 | 0.73 | 14.14 | 1.79 | **3.24** | **0.54** |
| | $c_t \rightarrow k_1$ | $r_1$ | – | – | 32.56 | 6.08 | 34.79 | 6.64 | **30.13** | **5.87** |
| RQ | $c_t \rightarrow k_1 \rightarrow r_1$ | $k_2$ | 164.77 | 25.07 | 45.20 | 5.62 | 8.48 | 1.79 | **4.85** | **1.59** |
| | $c_t \rightarrow k_1 \rightarrow r_1 \rightarrow k_2$ | $r_2$ | 200.14 | 28.94 | **15.91** | 4.93 | 16.02 | 4.98 | 20.50 | **4.77** |
| | $c_t \rightarrow k_1$ | $k_2$ | – | – | 15.82 | 2.50 | 9.37 | 2.21 | **7.48** | **1.63** |
| | $c_t \rightarrow k_1 \rightarrow r_1$ | $r_2$ | – | – | 14.89 | 4.62 | **12.24** | **4.38** | 16.41 | 4.58 |

The *previous scenes* denote the scenes that have been used for previous training of the algorithm, the *current scene* denotes the evaluation scene to compute both errors $t_{err}$ in [%] and $r_{err}$ in [°/100m]. $c_t$ refers to the Cityscapes training set. $r_i$ and $k_i$ are sequences from KITTI and the Oxford RobotCar dataset. Bold and underlined values indicate the best and second best scores on each sequence.

**CL-SLAM:** To address continual SLAM in the context of vision-based data, we propose CL-SLAM as shown in Figure 19. The core of CL-SLAM is the dual-network architecture of the visual odometry (VO) model that consists of an *expert* that produces myopic online odometry estimates and a *generalizer* that focuses on the long-term learning across environments. We train both networks in a self-supervised manner where the weights of the expert are updated only based on online data, whereas the weights of the generalizer are updated based on a combination of data from both the online stream and a replay buffer. We use the VO estimates of the expert to construct a pose graph. To reduce drift, we detect global loop closures and add them to the graph, which is then optimized. Finally, we can create a dense 3D map using the depth predicted by the expert and the optimized path. A complete description of the technical approach including details of the model architecture and training scheme can be found in the full paper.

### 4.2.4 Performance evaluation

In order to quantitatively evaluate the performance of our proposed approach, we compute both the adaptation quality (AQ) and the retention quality (RQ) by deploying CL-SLAM and the baseline methods on a fixed sequence of scenes. In particular, we use the official training split of the Cityscapes dataset [22] to initialize the DepthNet and PoseNet. The pre-training step is followed by a total of four scenes of the Oxford RobotCar dataset [74] and the KITTI dataset [33].

$$(c_t \rightarrow k_1 \rightarrow r_1 \rightarrow k_2 \rightarrow r_2), \tag{16}$$

where $c_t$ refers to the Cityscapes training set.

We compare CL-SLAM to three baselines that are inspired by previous works towards online adaptation to a different environment compared to the environment used during training. As previously noted, continual SLAM differs from such a setting in the sense that it considers a sequence of different environments. First, $\mathscr{B}_{expert}$ imitates the strategy employed by Li *et al.* [67], using a single set of network weights that is continuously updated based on the current data. This corresponds to only using the expert network in our architecture without resetting the weights. Second, $\mathscr{B}_{general}$ follows CoMoDA [59] leveraging a replay buffer built from previously seen environments. This method corresponds to only using the generalizer network.

Finally, we compute the error without performing any adaptation, i.e., $\mathscr{B}_{fixed}$ utilizes network weights fixed after the pre-training stage.

In Table 10, we show the translation and rotation errors for our CL-SLAM and the baselines. For the AQ and PQ scores, please refer to the full paper. Overall, our method is able to effectively address all challenges listed in the previous section, due to its dual-network architecture and efficient usage of replay data.

### 4.2.5 Future Work

Future work will focus on transferring the proposed design scheme to more advanced visual odometry methods, e.g., using point matching via optical flow. We further plan to address the currently infinite replay buffer to mitigate the scaling problem, e.g., by storing more abstract representations or keeping only the most representative images.

## 4.3 SLAM for Row Guidance System

### 4.3.1 Introduction, objectives and summary of state of the art

In agriculture, farmers first seed their crops and then conduct plant care operations, e.g., weeding, insect control, fungicides, etc. If the farmer chooses to seed their crop with a tractor instead of the field robot, the location of the plant rows are unknown. Although the positions of the plant rows can be recorded with an RTK GPS tractor, it is very likely (and this is what AGI has observed throughout the years) that the recorded rows and the actual rows won't match due to RTK GPS accuracy and errors. If a farmer then uses the Robotti for a plant care operation, it is highly likely that the implement or the robot will damage the crop.

To handle this problem, AGI tried to create a row guidance system for Robotti that will enable it to follow rows that have not been seeded/planted by itself.

### 4.3.2 Description of work performed so far

Row guidance can be a combination of DL-based methods, classic image analysis and computer vision algorithms and GNSS localization. Using the robot's plan and its GNSS position, the robot starts driving. Using its FrontEye camera, the robot can detect the crop rows and steer the robot based on their actual location. Using its GNSS, the robot records its positions in real time and uploads them to Amazon Web Service (AWS) in order to map its position and path. By doing so, the user is able to 'replay' the recorded path in subsequent operations.

The plant emergent zone Deep Learning model can find the point at which the plant emerges from the soil. This can be used by the row guidance algorithm to better extrapolate where the position of the rows is at. The robot can change its path based on the position of the crop rows. Using the robots GNSS, the path based on the position of the rows is recorded and uploaded to the cloud server in real time.

**Row Guidance Algorithm**

The row guidance algorithm has been improved. The structure of the original row guidance algorithm is shown in Fig. 20 (work performed before the third year of the project). The image is acquired, then the color is segmented, the image is warped so that the plant rows are straight, and the algorithm slices the image and finds a histogram based on the green saturation. From

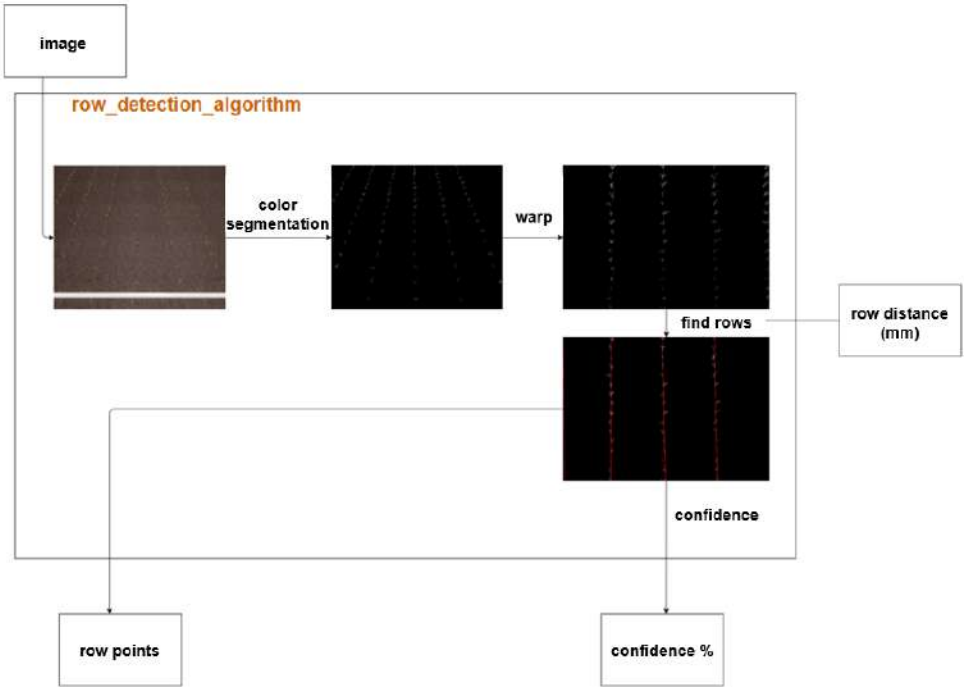



Figure 20: Original structure of the row guidance algorithm.

there, weighted least square regression is used to find the row. The row detection confidence is calculated based on how well the line fits with where the individual plants are at in the image.

As shown in Fig. 21, Pin the new row guidance algorithm (work performed in the third year of the project) part of the row guidance algorithm is the same and part has been improved. The slice image module, the "for loop" over the number of slices and the histogram creation module are the same as in the previous one. The find row position, find confidence, update GT position, find confidence, construct polyline and set row line bound modules have been improved. Originally the user had to provide the row distance value in the system. However, this had the weakness of being highly inflexible, as the row width can vary slightly depending on where the crop emerges from the soil. The new method still takes the row distance value into consideration, but uses it as an indication rather than an absolute. To find the individual plants, confidences for the histograms in the slices are calculated, see Fig. 22. The higher the confidence, the more it is expected the plant is in the crop row. After the confidences are found for the individual plants using the histograms, the row with the highest average confidence is used as the start row (GT, ground truth, position). The user-provided row width is then applied for the remaining rows. In Fig. 22, the colors of the detected plant points (marked as X's) denote their confidence value. The polyline representing the rows is calculated and is then overlayed onto the image, see Fig. 23. Green lines represent a high level of confidence, whereas blue lines represent a low level of confidence.

**Deep Learning – Crop emergence zone**

In 2022, images uploaded using AgroIntelli's CropEye system were used to train a model (YOLOv5) to annotate the plant's emergence zone. It was decided to find the plant emergence zone instead of the emergent point since the latter will often be a very precise location, whereas the zone allows for a degree of imprecision. However, the location of the plant compared to the location of the emergent zone is a significant step up. 4750 images of white cabbages crop

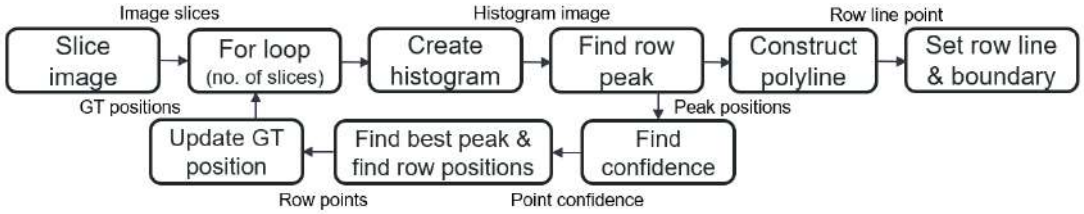



Figure 21: Diagram shows the structure of the new row guidance algorithm. GT = ground truth.

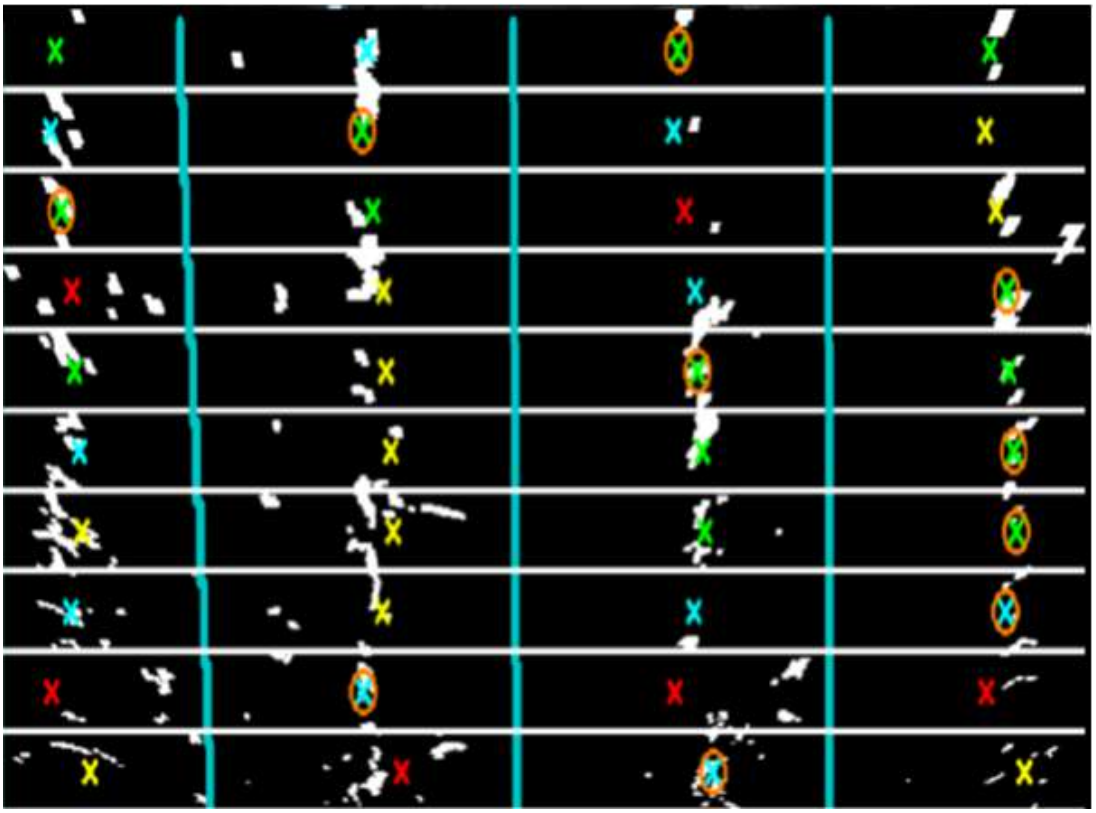


Figure 22: An illustration of the histogram for the different slices. Xs represent the plant points whereas their color represents the range of confidence: green is above 90 percent, turquoise is between 90 and 66 percent, yellow is between 66 and 33 percent, purple is below 33 percent, and red is 0 percent. The blue lines are the row boundary lines. The origin point of the different slices is shown as an orange circle.

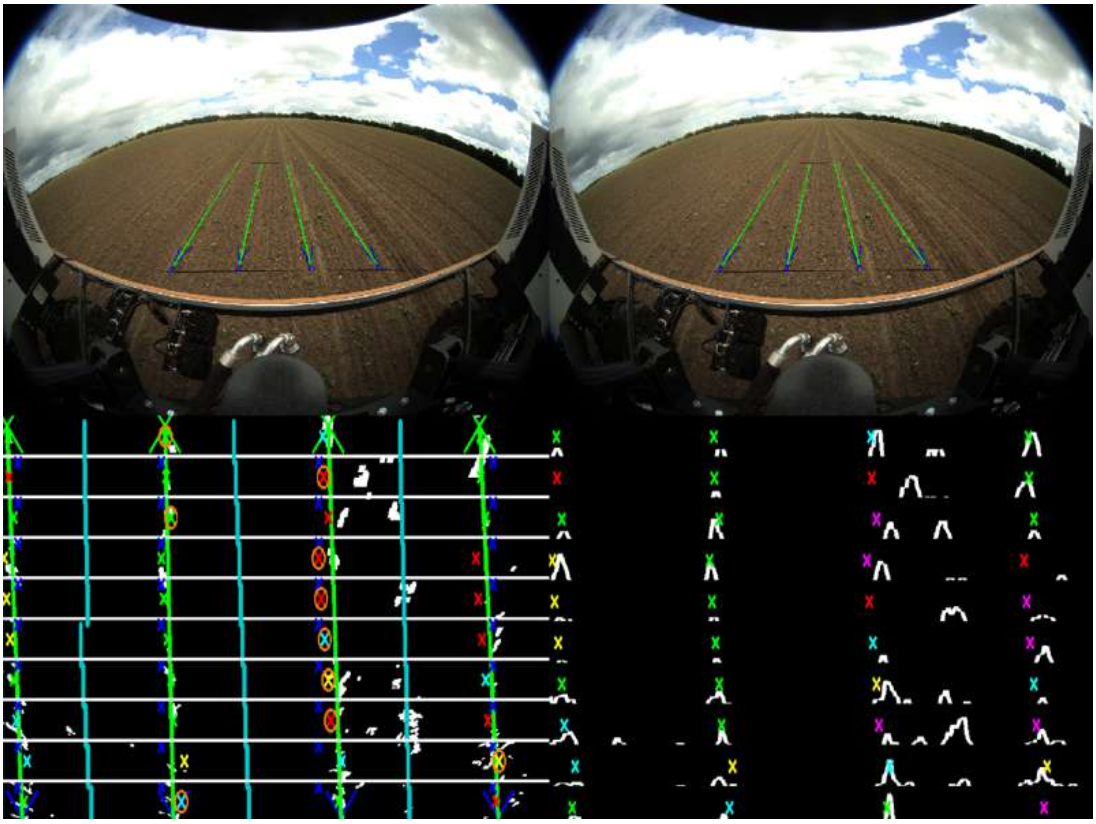


Figure 23: Image shows the process to find the crop rows (both left and right camera images are the same). Left: The white lines represent the cut for the slices. The colored X's represent the confidence range (see figure above for ranges). The blue lines represent the row boundaries. The white patches are where green was found on the image. Right: The colored X's represent the confidence range. The histograms represent where the patches of green are located and their corresponding histogram.

Figure 24: The purple hand drawn circles at the crop emergent point have been annotated after the model has found the cabbages.

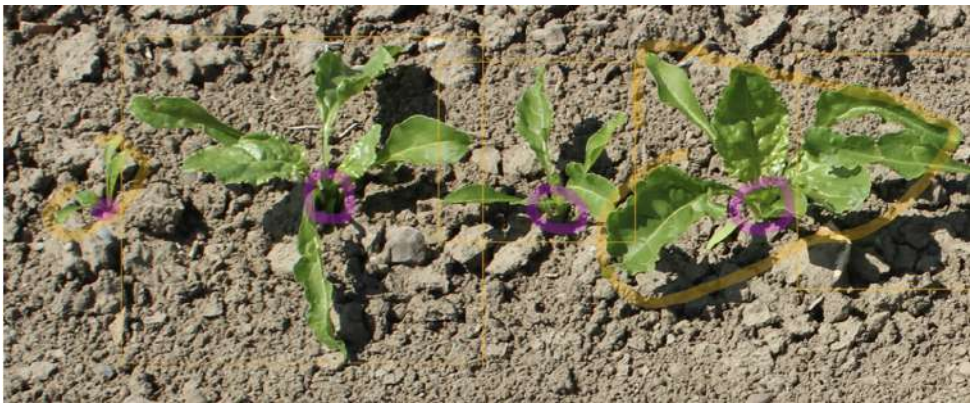


Figure 25: Example of the sugar beet emergent zone annotations.

emergence zone were annotated and used for model training, see Fig. 24. As there can be multiple cabbages in an image, this resulted in 12483 white cabbage bounding boxes and 6640 white cabbage Plant Soil Emerges Zones (PSEZ).

In addition to the cabbages, sugar beets crop emergence zones have also been annotated on 489 images, see Fig. 25. A total of 9208 sugar beet crop emergent zones were annotated from 6 different locations in EU, see Fig. 26.

**Steering, visualization and mapping**

The steering line is found by starting at the center of the image, then for each slice, subtracting the left and right points that mirror each other on the corresponding line. These values are then averaged, then multiplied by the horizontal meter per pixel value. Fig. 27, shows the steering line as calculated, based on the plant's positions in the rows.

As the robot navigates, based on either the FrontEye camera system or the GNSS, the actual driven position is recorded and uploaded to the cloud, creating the map. Currently, this information is used to to calculate the cross-track error, or the difference between the planned path and the actual path. Ideally, the robot should drive as close as possible to the planned path. The row guidance system will use the same recorded information. This information will be the map (collection of paths) of where the robot has driven and, as the crop rows don't move, can be used for future operations.

Figure 26: Locations of where the sugar beet images were acquired in 2022.

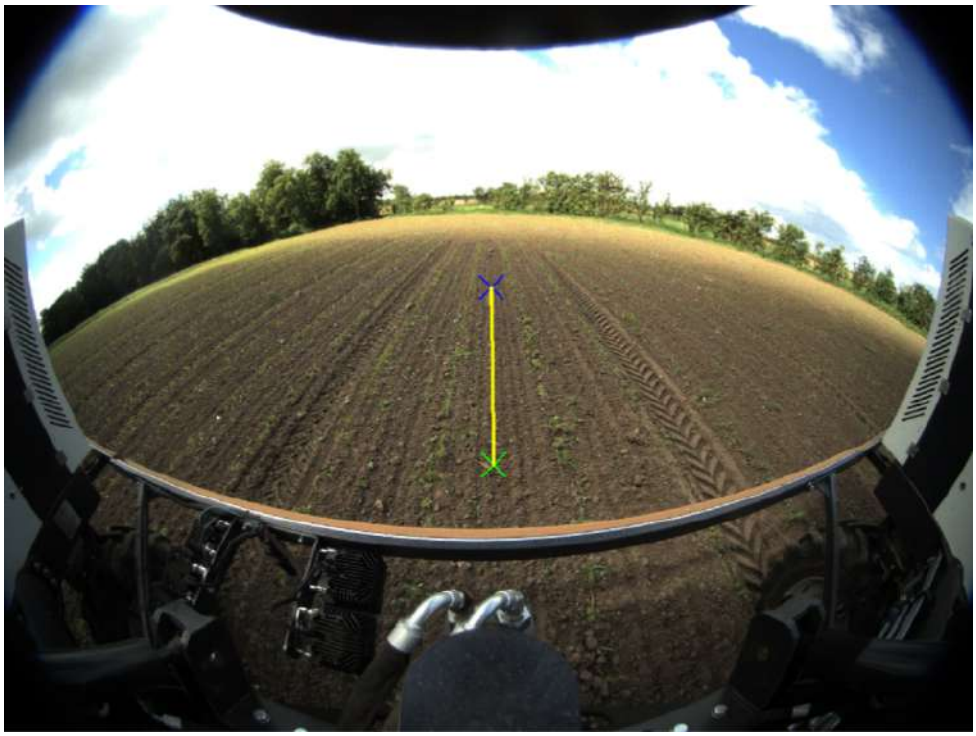


Figure 27: Calculated steering line visualized on the image. The green x is the starting point and the blue x is the steering point.

### 4.3.3 Performance evaluation

To check the performance of the steering line evaluation approach, the steering line will be compared against the actual drive line. All images are recorded with metadata including GPS location. In addition, both the planned path of the robot and the actual path of the robot is recorded in AWS. The row guidance steering line and actual path of the robot will be visualized in order to test if the lines are similar over time.

### 4.3.4 Conclusions and Future Work

**Row guidance system**
The DL models will be added to the row guidance framework as shown in Fig. 21. There are several potential ways for adding these models in the framework. The Crop and Weed model could remove all the weeds from the image, leaving the crop. Then the row guidance algorithm would only be fed with the known locations of the crop, removing or reducing the error caused from the weeds, provided that the DL model will be able to find the crop. This will decrease the sensitivity of the row guidance algorithm to too many weeds. However the algorithm will still be effected by wind and the crop that is laying down. In addition, it is possible that the precision of the row would be affected when the plant reaches a certain size, since this could affect the bounding boxes accuracy However this issue can be eliminated, if the locations of the plants are recorded and mapped when they are very small.

A second, more precise approach, is to use a DL model to find both the plant and the plant emergent zone. However, this requires that the model is robust and accurate enough to find both the crop and the emergent zone. When the zones are known, then the precision is fairly high, as the bounding boxes will be smaller and likely the centers will be more aligned.

**DL for detecting crop emergence zone**
In 2023, the crop emergence zone will be added to AgroIntelli's Crop and Weed model (based on approx. 50k annotated images). Additional images collected by the CropEye camera system will be annotated and used to train and evaluate the model, hoping that the new dataset will increase robustness and will minimize false positives and negatives.

**Steering and mapping**
The current idea for steering the robot is to use the nudge system in Robotti. When the farmer is in the field and can see that the robot is too close to the crop, he is able to nudge the robot to the left or right. Using the same system, the row guidance framework can send the nudge information to the the steering controller, which then adjusts the robot's trajectory as necessary. The mapping system is currently operational, however in the future we will have to add in the route planner website the option to run an operation based on the recorded row guidance map.

# 5 Sensor information fusion

## 5.1 Multimodal fusion framework with robustness extensions.

### 5.1.1 Introduction, objectives and summary of state of the art

Human visual object recognition is often rapid, effortless and largely viewpoint or object orientation independent [34]. However, with the advent of deep neural networks, computer vision algorithms have achieved unprecedented performance and even surpassed human capabilities on tasks including image classification, face identification and object recognition [105]. Despite the accuracy of deep neural networks, their generalisation property across changes in the input distribution e.g., illumination changes and harsh conditions, is not established yet. In a behavioural comparison of humans and well-known deep neural architectures like ResNet-152 [40], classification performance of neural networks seems to decline rapidly with decreasing signal-to-noise ratio under image distortions [34, 35]. In the context of object detection, multi-sensor configurations are known to provide redundancy and often enhance performance of the detection algorithms. Moreover, efficient sensor fusion strategies minimize uncertainties, increase reliability and are crucial in achieving robustness against asymmetric sensor failures. Increasing number of sensors might enhance the performance of detection algorithms, however this comes with a considerable computational/energy cost. This is often not desirable in mobile robotic systems which typically have constraints in terms of computational power and battery consumption. In such cases, intelligent choice/combination of sensors is critical. We build upon a multi-modal fusion strategy and a data augmentation method that were proposed for object detection in harsh lighting conditions [75, 76]. Together, these two methods resulted in a multimodal object detection system that can deal with harsh lighting conditions. In the third year of the project, we have continued this work. In particular, we have extended the data augmentation method to further improve the performance of the multimodal object detection system. That is to say, we have worked on an extension of Random Shadows and Highlights (RSH) to polygon-shaped masks in order to allow for robustness against a larger variety of lighting perturbations and we have also performed more extensive evaluation.

### 5.1.2 Description of work performed so far

Shadows in indoor environments naturally take a shape similar to that of objects casting them. Artificially creating shadows that resemble real-world objects requires scene modeling from a single image or prior knowledge of objects in the image. To solve this problem, we exploit the polygon data extracted from a large-scale object-segmentation dataset, in this case COCO [70]. It contains $886,284$ instances of objects/polygons of 80 object categories. We randomly select, translate, and rotate the polygons and create shadows and highlights to imitate real-world harsh lighting conditions in indoors. This method is named as “Semantic Shadows and Highlights (SSH)”. We also provide a drawing tool for creating polygons, which allows the users to draw and create artificial shadows of any other shape they desire. On the contrary, shadows cast by buildings in the outdoors often take the form of trapezoids. Furthermore, bright light entering through windows or doors inside the buildings also take similar shapes. To imitate outdoor shadows cast by buildings, we propose to create shadows of a trapezoidal shape with its base aligned with the vertical axis of the image. This method is named as “Random Shadows and Highlights (RSH)”.

Table 11: Image classification results of AlexNet on CIFAR-10 dataset with augmentation probability of 0.5. The network is trained individually with each augmentation method and tested against all others individually and combined. The combined tests include all methods except the one being tested. "Avg Acc" is the mean of individual outputs for each method under examination against all augmentation methods.

| Method | Train Acc | Test Acc RGC (0.5) | Test Acc RCJ (0.5) | Test Acc RDI (0.5) | Test Acc RSH (0.5) | Test Acc SSH (0.5) | Avg Acc | Test Acc All (0.5) | TTD |
|---|---|---|---|---|---|---|---|---|---|
| Baseline | 0.991 | 0.764 | 0.644 | 0.574 | 0.622 | 0.564 | 0.634 | 0.258 | 0.171 |
| RGC (0.5) | 0.956 | 0.816 | 0.670 | 0.599 | 0.636 | 0.576 | 0.659 | 0.290 | 0.140 |
| RCJ (0.5) | 0.860 | 0.801 | 0.764 | 0.645 | 0.631 | 0.56 | 0.680 | 0.354 | 0.096 |
| RDI (0.5) | 0.777 | 0.791 | 0.695 | 0.704 | 0.623 | 0.559 | 0.674 | 0.373 | 0.073 |
| RSH (0.5) | 0.874 | 0.772 | 0.660 | 0.588 | 0.777 | 0.615 | 0.682 | 0.345 | 0.097 |
| **SSH** (0.5) | 0.789 | 0.770 | 0.655 | 0.579 | 0.729 | 0.720 | **0.691** | **0.394** | **0.069** |

Table 12: Image classification results of AlexNet on CIFAR-10 dataset with augmentation probability of 1.0. The configuration of the experiments is same as stated in Table 11

| Method | Train Acc | Test Acc RGC (1) | Test Acc RCJ (1) | Test Acc RDI (1) | Test Acc RSH (1) | Test Acc SSH (1) | Avg Acc | Test Acc All (1) | TTD |
|---|---|---|---|---|---|---|---|---|---|
| Baseline | 0.991 | 0.709 | 0.460 | 0.329 | 0.416 | 0.303 | 0.443 | 0.129 | 0.171 |
| RGC (1) | 0.935 | 0.804 | 0.503 | 0.374 | 0.445 | 0.332 | 0.492 | 0.139 | 0.131 |
| RCJ (1) | 0.781 | 0.755 | 0.720 | 0.481 | 0.413 | 0.287 | 0.531 | 0.168 | 0.061 |
| RDI (1) | 0.622 | 0.580 | 0.555 | 0.603 | 0.298 | 0.226 | 0.452 | 0.188 | 0.019 |
| RSH (1) | 0.794 | 0.719 | 0.496 | 0.338 | 0.740 | 0.423 | 0.543 | 0.168 | 0.054 |
| **SSH** (1) | 0.656 | 0.683 | 0.463 | 0.325 | 0.662 | 0.642 | **0.555** | **0.208** | **0.014** |

### 5.1.3 Performance evaluation

We have evaluated RSH and SSH and compared against alternative approaches. We have done this in both detection and classification tasks, as shown in Table 11, Table 12, Table 13 and Table 14. We have also set up manipulation experiments in which we are going to evaluate the effectiveness of RSH and SSH. These experiments include a vision-based box-pushing task (Figure 28a) and and a vision-based pick and place task with language instructions (Figure 28b).

### 5.1.4 Conclusions and Future Work

The utility of deep neural networks in the real world is feasible only if the models are robust against harsh environmental conditions. In the learning phase of vision systems, it is essential to expose the network to the irregularities that images might encounter across unconstrained realistic scenarios. Among the most prevalent concerns in vision models is their sensitivity to changing lighting conditions. Thus, we proposed "Random Shadows and Highlights" which creates explicit shadows and highlights in the form of trapezoids and "Semantic Shadows and

Table 13: Object detection results of DETR on Pascal VOC 2007 dataset with augmentation probability of 0.5. The network is trained individually with each RSH and SSH and tested against both individually. "Avg" is the mean of individual outputs for each method under examination with augmentation applied.

| Method | mAP @ IoU = 0.5 | | | | | TTD |
|---|---|---|---|---|---|---|
| | Train | Test (None) | Test w/ RSH p=0.5 | Test w/ SSH p=0.5 | Avg | |
| baseline | 0.970 | 0.807 | 0.799 | 0.737 | 0.768 | 0.163 |
| RSH p=0.5 | 0.959 | 0.809 | 0.799 | 0.784 | 0.792 | 0.150 |
| **SSH** p=0.5 | 0.961 | 0.810 | **0.807** | **0.804** | **0.806** | 0.151 |

Table 14: Object detection results of DETR on Pascal VOC 2007 dataset with augmentation probability of 1.0. The configuration of the experiments is same as stated in Table 13

| Method | mAP @ IoU = 0.5 | | | | | TTD |
|---|---|---|---|---|---|---|
| | Train | Test (None) | Test w/ RSH p=1 | Test w/ SSH p=1 | Avg | |
| baseline | 0.970 | 0.807 | 0.669 | 0.602 | 0.6355 | 0.163 |
| RSH p=1 | 0.970 | 0.805 | 0.765 | 0.783 | 0.774 | 0.165 |
| **SSH** p=1 | 0.952 | 0.806 | **0.797** | **0.791** | **0.794** | **0.146** |

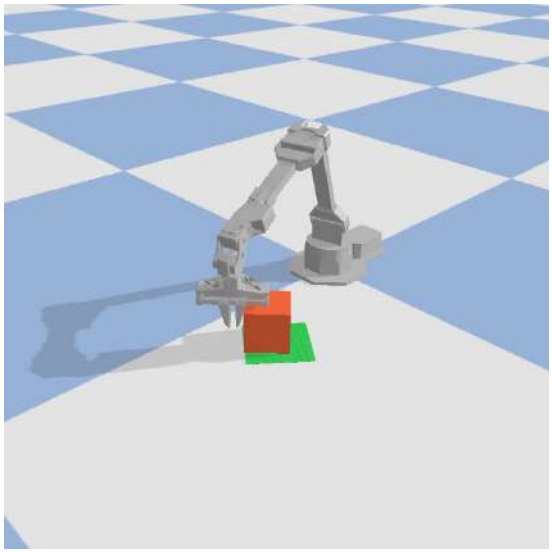
Highlights" which creates shadows in the form of real world objects in the existing image datasets. These augmentation methods challenge the network in the training phase in order to develop immunity against such harsh conditions when deployed in real world applications. For future work, we plan to complete the evaluations in the tasks that are shown in Figure 28a and Figure 28b. Ideally, we would also like to perform experiments with a real robotic setup for the box pushing task. Finally, we plan to shear the polygons by utilizing the concept of perspective projection to make even more realistic shadows. This could possibly further improve the data augmentation method that was proposed for training the multimodal object detector for harsh lighting conditions.

## 5.2 Multimodal Feature Fusion Framework for Manipulation

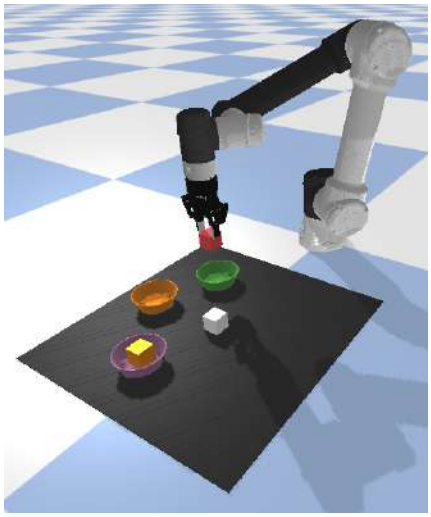
### 5.2.1 Introduction and objectives

A robot, while performing a specific task, often has access to multiple different data sources from its sensors. Such sensors include RGB cameras, lidars, force feedback sensors, microphones, infrared sensors and more. While taking advantage of such diverse data modalities is obviously beneficial, it is a complex task. TAU has previously proposed a general feature fusion framework suitable for these scenarios, with the primary application area being robotic arm manipulation (pick and place, insertion, etc.). The following requirements are set for the framework to adhere to:

- Support for input signals of modalities that are expected to be supported on the robotic arm in an industrial setting (RGB and depth camera feeds, force sensors, self-pose measurements).



(a) Box-pushing task.



(b) Pick and place task.

Figure 28: Figure **(a)** shows a box pushing task in which we plan to evaluate Semantic Shadows and Highlights (SSH). The box pose is not observed directly but has to be estimated from images. Figure **(b)** shows a pick and place task where we plan to evaluate the effectiveness of SSH. Here the task is to execute language commands, such as "Pick the red block and place it in the green bowl".

- Inclusion of baseline feature fusion modules, supporting arbitrary combinations of modalities.
- Support for both task-specific surrogate objectives and input reconstruction objectives.
- Extensibility of the framework with respect to inputs/modalities, fusion approaches and outputs.
- Compactness of the intermediate and final representations, for faster computation.
- Support for robustness: minimizing the impact of damaged or missing data from individual modalities during inference.
- Capability to train on real and/or synthetic data.
- Integration with the simulation environment (Webots) for collecting synthetic data.

More detailed descriptions of the framework itself have been previously provided in Deliverables D4.1 and D4.2. To avoid redundancy, we do not replicate them here.

### 5.2.2 Description of work performed so far

Over the past year the majority of framework elements have been implemented, specifically:

- Input encoders for RGB images, depth (lidar) images, proprioception, force torque sensors.
- Feature fusion modules based on concatenation, MLPs, product-of-experts.

- Surrogate objectives supporting robotic arm insertion tasks, i.e. optical flow reconstruction, future pose prediction.
- Robustness extensions, i.e. input reconstruction decoders, cross-modality compensation module.
- Reinforcement learning pipeline for framework training.
- Simulation environment (Webots) for collecting synthetic data and making demonstrations.

However, some of the other planned steps have run into difficulties and are currently delaying the final integration of the framework into the OpenDR toolkit. Integration of the entire pipeline from the DL models to the simulation environment is not complete, thus the integration with the real robotic arm (which is expected to be a complex process in itself) is pending. Generalizations of the feature fusion module via neural architecture search (NAS), previously described in Deliverable D4.2, do not yet yield improvements over handcrafted baseline designs, in terms of task performance or computation speed. Finally, the processing speed, being acceptable on the powerful desktop system, would currently be impractical in the embedded environments as targeted by OpenDR.

### 5.2.3 Conclusions and Future Work

In the final year of the project we aim to address the shortcomings listed above, drawing on the assistance and expertise of other partners where appropriate. In addition to debugging and speed enhancements, we are working on extending the framework capabilities by adding new input decoders for new and existing modalities. We plan to continue experimenting with NAS for feature fusion optimization, ultimately producing at least one publication regarding our experience. Finally, time permitting, we plan to investigate the active perception possibilities for our multimodal setting and tasks.

# 6 Conclusions

In the third year of the project, the consortium carried out a series of activities following the overall objectives of the project. In the context of work package 4, we focused on objective O1 for providing a modular, open and non-proprietary toolkit for core robotic functionalities enabled by lightweight deep learning, and objective O2 for leveraging AI and cognition in robotics to go from perception to action.

AU worked towards O1a by introducing a novel uncertainty estimation method called Layer Ensembles (Section 2.2) that achieves high uncertainty quality while achieving high speed and lower memory usage and by performing structured pruning (Section 3.2) to 3D Object Detection models for increasing their inference speed. Additionally, AU introduced a Variational Neural Networks (VNNs) implementation in PyTorch and JAX (Section 2.3) that allows for easy experimentation and application of VNNs to the existing projects.

AUTH worked towards towards O2a by developing a novel active perception object recognition approach that provides active perception capabilities for any existing DL-based object detector. Furthermore, AU and AUTH worked towards O1a by proposing a voxel-based 3D Single Object Tracking method (Section 3.1) that operates with real-time speed on embedded GPU platforms and achieves high tracking accuracy under the real-time evaluation scenario.

ALU-FR worked towards objective O2 by proposing PAPS [81], a method for amodal panoptic segmentation (Section 2.1). PAPS enables robots to see "behind" objects by providing pixel-wise semantic annotations for both visible and occluded regions of an image. Furthermore, ALU-FR proposed Batch3DMOT [12] (Section 3.3), an offline 3D tracking framework that follows the tracking-by-detection paradigm and utilizes multiple sensor modalities (camera, LiDAR, radar) to solve a multi-frame, multi-object tracking objective. In addition, ALU-FR proposed PADLoC [2] (Section 4.1), an attention-based loop closure detection and point cloud registration method for LiDAR-SLAM exploiting panoptic annotations during training time, and by introducing the novel task of continual SLAM [112] (Section 4.2). This task combines lifelong SLAM with online domain adaptation to effectively reflect challenges that occur when deploying a SLAM system to the real world.

AGI worked towards building the blocks for a row guidance system, which can correct the path and steering of the agricultural field robot based on the position of the actual crop rows (Section 4.3). Improvements were made to the row guidance algorithm, and DL-based methods will be fully integrated into the system in 2023 to increase the accuracy and precision. Based on the current implementation, the robot is able to upload its actual position to build maps that can be used in consecutive operations.

TUD developed an extension of the lightweight data augmentation method for such harsh lighting conditions called "Semantic Shadows and Highlights" (Section 5.1). The initial results of this method are promising and we plan to further evaluate it in robot manipulation tasks.

TAU worked towards objective O2 by further developing the multimodal feature fusion framework (Section 5.2), incorporating the cross-modality robustness extensions, investigating novel fusion structures by means of neural architecture search, and preparing the integration with simulated and real production environments.

# References

[1] R. Arandjelović, P. Gronat, A. Torii, T. Pajdla, and J. Sivic. NetVLAD: CNN architecture for weakly supervised place recognition. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 40(6):1437–1451, 2018. 39

[2] J. Arce, N. Vödisch, D. Cattaneo, W. Burgard, and A. Valada. Padloc: Lidar-based deep loop closure detection and registration using panoptic attention. *arXiv preprint arXiv:2209.09699*, 2022. 8, 38, 57, 115

[3] A. L. Ballardini, D. Cattaneo, and D. G. Sorrenti. Visual localization at intersections with digital maps. In *Int. Conf. on Robotics and Automation*, pages 6651–6657, 2019. 38

[4] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, and P. H. Torr. Fully-convolutional siamese networks for object tracking. *arXiv:1606.09549*, 2016. 26

[5] B. Bešić, N. Gosala, D. Cattaneo, and A. Valada. Unsupervised domain adaptation for lidar panoptic segmentation. *IEEE Robotics and Automation Letters*, 7(2):3404–3411, 2022. 42

[6] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877, 2017. 20

[7] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. Weight Uncertainty in Neural Networks. 2015. 14, 15, 20

[8] D. S. Bolme, J. R. Beveridge, B. A. Draper, and Y. M. Lui. Visual object tracking using adaptive correlation filters. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2544–2550, 2010. 26

[9] T. Bozinis, N. Passalis, and A. Tefas. Improving visual question answering using active perception on static images. In *Proc. International Conference on Pattern Recognition (ICPR)*, pages 879–884, 2021. 20

[10] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018. 20

[11] G. Brasó and L. Leal-Taixé. Learning a neural solver for multiple object tracking. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, pages 6246–6256, 2020. 34, 35

[12] M. Büchner and A. Valada. 3d multi-object tracking using graph neural networks with cross-edge modality attention. *IEEE Robotics and Automation Letters*, 7(4):9707–9714, 2022. 8, 34, 57, 102

[13] W. Burgard, D. Fox, and S. Thrun. Probabilistic robotics. *The MIT Press*, 2005. 8

[14] D. Cattaneo, M. Vaghi, and A. Valada. LCDNet: Deep loop closure detection and point cloud registration for LiDAR SLAM. *IEEE Transactions on Robotics*, pages 1–20, 2022. 39, 40, 41

[15] D. S. Chaplot, E. Parisotto, and R. Salakhutdinov. Active neural localization. In *International Conference on Learning Representations*, 2018. 8

[16] T. Charnock, L. Perreault-Levasseur, and F. Lanusse. Bayesian neural networks. *arXiv:2006.01490*, 2020. 14, 20

[17] L.-C. Chen, M. Collins, Y. Zhu, G. Papandreou, B. Zoph, F. Schroff, H. Adam, and J. Shlens. Searching for efficient multi-scale architectures for dense image prediction. In *Advances in Neural Information Processing Systems*, pages 8713–8724, 2018. 13

[18] X. Chen, T. Läbe, A. Milioto, T. Röhling, J. Behley, and C. Stachniss. OverlapNet: A siamese network for computing lidar scan similarity with applications to loop closing and localization. *Autonomous Robots*, 2021. 40, 41

[19] X. Chen, A. Milioto, E. Palazzolo, P. Giguère, J. Behley, and C. Stachniss. SuMa++: Efficient LiDAR-based semantic SLAM. In *Int. Conf. on Intelligent Robots and Systems*, pages 4530–4537, 2019. 38

[20] B. Cheng, M. D. Collins, Y. Zhu, T. Liu, T. S. Huang, H. Adam, and L.-C. Chen. Panoptic-deeplab: A simple, strong, and fast baseline for bottom-up panoptic segmentation. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, 2020. 12

[21] H.-k. Chiu, A. Prioletti, J. Li, and J. Bohg. Probabilistic 3d multi-object tracking for autonomous driving. *arXiv preprint arXiv:2001.05673*, 2020. 37

[22] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The Cityscapes dataset for semantic urban scene understanding. In *IEEE Conf. on Computer Vision and Pattern Recognition*, pages 3213–3223, 2016. 44

[23] P. Dendorfer, H. Rezatofighi, A. Milan, J. Shi, D. Cremers, I. Reid, S. Roth, K. Schindler, and L. Leal-Taixé. Mot20: A benchmark for multi object tracking in crowded scenes. *arXiv preprint:2003.09003*, 2020. 34

[24] V. Dwaracherla, X. Lu, M. Ibrahimi, I. Osband, Z. Wen, and B. V. Roy. Hypermodels for exploration. In *ICLR Proceedings*, volume 8, 2020. 14, 15

[25] Z. Fang, S. Zhou, Y. Cui, and S. Scherer. 3d-siamrpn: An end-to-end learning method for real-time 3d single object tracking using raw point cloud. *IEEE Sensors Journal*, 21(4):4995–5011, 2021. 26, 29, 30

[26] P. Follmann, R. König, P. Härtinger, M. Klostermann, and T. Böttger. Learning to see the invisible: End-to-end trainable amodal instance segmentation. In *IEEE Winter Conference on Applications of Computer Vision*, pages 1328–1336, 2019. 14

[27] W. K. Fong, R. Mohan, H. J. Valeria, L. Zhou, H. Caesar, O. Beijbom, and A. Valada. Panoptic nuscenes: A large-scale benchmark for lidar panoptic segmentation and tracking. *IEEE Robotics and Automation Letters*, 2022. 34, 37

[28] D. Frossard and R. Urtasun. End-to-end learning of multi-sensor 3d tracking by detection. In *Int. Conf. on Robotics and Automation*, 2018. 34

[29] Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *JMLR Workshop and Conference Proceedings*, volume 48, pages 1050–1059, 2016. 14, 15, 20

[30] N. Gao, Y. Shan, Y. Wang, X. Zhao, Y. Yu, M. Yang, and K. Huang. Ssap: Single-shot instance segmentation with affinity pyramid. In *Int. Conf. on Computer Vision*, 2019. 12

[31] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? The KITTI vision benchmark suite. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012. 29

[32] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, 2012. 34, 37

[33] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the KITTI vision benchmark suite. In *IEEE Conf. on Computer Vision and Pattern Recognition*, 2012. 40, 41, 44

[34] R. Geirhos, D. H. J. Janssen, H. H. Schütt, J. Rauber, M. Bethge, and F. A. Wichmann. Comparing deep neural networks against humans: object recognition when the signal gets weaker. *CoRR*, abs/1706.06969, 2017. 52

[35] R. Geirhos, C. R. M. Temme, J. Rauber, H. H. Schütt, M. Bethge, and F. A. Wichmann. Generalisation in humans and deep neural networks. *CoRR*, abs/1808.08750, 2018. 52

[36] S. Giancola, J. Zarzar, and B. Ghanem. Leveraging shape completion for 3d siamese tracking, 2019. 26, 29, 30

[37] N. Gosala and A. Valada. Bird's-eye-view panoptic segmentation using monocular frontal view images. *IEEE Robotics and Automation Letters*, 2022. 12

[38] W. K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, 1970. 20

[39] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016. 23

[40] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016. 52

[41] L. He, X. Wang, and H. Zhang. M2DP: A novel 3D point cloud descriptor and its application in loop closure detection. In *Int. Conf. on Intelligent Robots and Systems*, pages 231–237, 2016. 40, 41

[42] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. *arxiv:1707.06168*, 2017. 32

[43] D. Held, S. Thrun, and S. Savarese. Learning to track at 100 fps with deep regression networks. *1604.01802*, 2016. 26

[44] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista. High-speed tracking with kernelized correlation filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(3):583–596, 2015. 26

[45] A. Hornakova, R. Henschel, B. Rosenhahn, and P. Swoboda. Lifted disjoint paths with application in multiple object tracking. In *Int. Conf. on Machine Learning*, pages 4364–4375, 2020. 34

[46] J. V. Hurtado, R. Mohan, W. Burgard, and A. Valada. Mopt: Multi-object panoptic tracking. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshop on Scalability in Autonomous Driving*, 2020. 34

[47] W.-D. Jang, D. Wei, X. Zhang, B. Leahy, H. Yang, J. Tompkin, D. Ben-Yosef, D. Needleman, and H. Pfister. Learning vector quantized shape code for amodal blastomere instance segmentation. *arXiv preprint arXiv:2012.00985*, 2020. 14

[48] S. Jiayao, S. Zhou, Y. Cui, and Z. Fang. Real-time 3d single object tracking with transformer. *IEEE Transactions on Multimedia*, 2022. 27

[49] R. Jonschkowski and O. Brock. Learning state representations with robotic priors. *Autonomous Robots*, 39(3):407–428, 2015. 8

[50] L. Ke, Y.-W. Tai, and C.-K. Tang. Deep occlusion-aware instance segmentation with overlapping bilayers. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, pages 4019–4028, 2021. 14

[51] A. Kim, A. Ošep, and L. Leal-Taixé. Eagermot: 3d multi-object tracking via sensor fusion. *Int. Conf. on Robotics and Automation*, 2021. 34

[52] G. Kim, S. Choi, and A. Kim. Scan Context++: Structural place recognition robust to rotation and lateral variations in urban environments. *IEEE Transactions on Robotics*, 38(3):1856–1874, 2022. 40, 41

[53] D. P. Kingma and M. Welling. Auto-Encoding Variational Bayes. *arXiv:1312.6114*, 2014. 15

[54] X. Kong, X. Yang, G. Zhai, X. Zhao, X. Zeng, M. Wang, Y. Liu, W. Li, and F. Wen. Semantic graph based place recognition for 3D point clouds. In *Int. Conf. on Intelligent Robots and Systems*, pages 8216–8223, 2020. 38, 39

[55] H. Kretzschmar, G. Grisetti, and C. Stachniss. Lifelong map learning for graph-based slam in static environments. *KI - Künstliche Intelligenz*, 24(3):199–206, 2010. 41

[56] M. Kristan, J. Matas, A. Leonardis, T. Vojir, R. Pflugfelder, G. Fernandez, G. Nebehay, F. Porikli, and L. Cehovin. A novel performance evaluation methodology for single-target trackers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(11):2137–2155, 2016. 29

[57] H. kuang Chiu, J. Li, R. Ambrus, and J. Bohg. Probabilistic 3d multi-modal, multi-object tracking for autonomous driving. *Int. Conf. on Robotics and Automation*, 2021. 37

[58] S. Kullback and R. A. Leibler. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79 – 86, 1951. 15, 18

[59] Y. Kuznietsov, M. Proesmans, and L. Van Gool. CoMoDA: Continuous monocular depth adaptation using past experiences. In *IEEE Winter Conf. on Applications of Computer Vision*, 2021. 42, 44

[60] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom. PointPillars: Fast Encoders for Object Detection from Point Clouds. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2019. 8, 27

[61] J. Lee, Y. Bahri, R. Novak, S. S. Schoenholz, J. Pennington, and J. Sohl-Dickstein. Deep neural networks as gaussian processes. In *6th ICLR Proceedings*, 2018. 15

[62] B. Li, W. Wu, Q. Wang, F. Zhang, J. Xing, and J. Yan. Siamrpn++: Evolution of siamese visual tracking with very deep networks. *arXiv:1812.11703*, 2018. 26

[63] B. Li, J. Yan, W. Wu, Z. Zhu, and X. Hu. High performance visual tracking with siamese region proposal network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8971–8980, 2018. 26

[64] L. Li, X. Kong, X. Zhao, T. Huang, W. Li, F. Wen, H. Zhang, and Y. Liu. RINet: Efficient 3D lidar-based place recognition using rotation invariant neural network. *IEEE Robotics and Automation Letters*, 7(2):4321–4328, 2022. 38

[65] L. Li, X. Kong, X. Zhao, W. Li, F. Wen, H. Zhang, and Y. Liu. SA-LOAM: Semantic-aided LiDAR SLAM with loop closure. In *Int. Conf. on Robotics and Automation*, pages 7627–7634, 2021. 38

[66] M. Li, Y.-X. Wang, and D. Ramanan. Towards streaming perception. In *European Conference on Computer Vision*, pages 473–488. Springer, 2020. 26, 30

[67] S. Li, X. Wang, Y. Cao, F. Xue, Z. Yan, and H. Zha. Self-supervised deep visual odometry with online adaptation. In *IEEE Conf. on Computer Vision and Pattern Recognition*, 2020. 41, 42, 44

[68] S. Li, X. Wu, Y. Cao, and H. Zha. Generalizing to the open world: Deep visual odometry with online adaptation. In *IEEE Conf. on Computer Vision and Pattern Recognition*, pages 13179–13188, 2021. 41, 43

[69] Y. Liao, J. Xie, and A. Geiger. KITTI-360: A novel dataset and benchmarks for urban scene understanding in 2d and 3d. *arXiv preprint arXiv:2109.13410*, 2021. 14

[70] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014. 52

[71] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *Proc. European Conference on Computer Vision*, pages 21–37, 2016. 23

[72] H. Luo, Y. Gao, Y. Wu, C. Liao, X. Yang, and K.-T. Cheng. Real-time dense monocular SLAM with online adapted depth prediction network. *IEEE Transactions on Multimedia*, 21(2):470–483, 2019. 41, 42

[73] D. J. C. Mackay. Probable networks and plausible predictions — a review of practical bayesian methods for supervised neural networks. *Network: Computation in Neural Systems*, 6(3):469–505, 1995. 14, 20

[74] W. Maddern, G. Pascoe, C. Linegar, and P. Newman. 1 year, 1000km: The Oxford RobotCar dataset. *Int. Journal of Robotics Research*, 36(1):3–15, 2017. 44

[75] O. Mazhar, R. Babuska, and J. Kober. Gem: Glare or gloom, i can still see you - end-to-end multi-modal object detection. *IEEE Robotics and Automation Letters*, 6(4):6321–6328, 2021. Accepted Author Manuscript. 52

[76] O. Mazhar and J. Kober. Random shadows and highlights: A new data augmentation method for extreme lighting conditions. *arXiv preprint arXiv:2101.05361*, 2021. 52

[77] O. Michel. Cyberbotics ltd. webots™: professional mobile robot simulation. *International Journal of Advanced Robotic Systems*, 1(1):5, 2004. 21

[78] M. Mittal, R. Mohan, W. Burgard, and A. Valada. Vision-based autonomous uav navigation and landing for urban search and rescue. *arXiv preprint arXiv:1906.01304*, 2019. 33

[79] M. Mittal, A. Valada, and W. Burgard. Vision-based autonomous landing in catastrophe-struck environments. *arXiv preprint arXiv:1809.05700*, 2018. 11

[80] R. Mohan and A. Valada. Efficientps: Efficient panoptic segmentation. *Int. Journal of Computer Vision*, 129(5):1551–1579, 2021. 12

[81] R. Mohan and A. Valada. Amodal panoptic segmentation. *arXiv preprint arXiv:2202.11542*, 2022. 11, 12, 14, 57, 68

[82] R. Mohan and A. Valada. Perceiving the invisible: Proposal-free amodal panoptic segmentation. *IEEE Robotics and Automation Letters*, 7(4):9302–9309, 2022. 7, 11, 12

[83] B. Nanay. The importance of amodal completion in everyday perception. *i-Perception*, 9(4), 2018. 11

[84] I. Oleksiienko and A. Iosifidis. Analysis of voxel-based 3d object detection methods efficiency for real-time embedded systems. In *International Conference on Emerging Techniques in Computational Intelligence*, pages 59–64, 2021. 32

[85] I. Oleksiienko and A. Iosifidis. Layer Ensembles. *arXiv:2210.04882*, 2022. 15, 79

[86] I. Oleksiienko, P. Nousi, N. Passalis, A. Tefas, and A. Iosifidis. Vpit: Real-time embedded single object 3d tracking using voxel pseudo images. *arXiv:2206.02619*, 2022. 8, 26, 91

[87] I. Oleksiienko, D. T. Tran, and A. Iosifidis. Variational neural networks. *arxiv:2207.01524*, 2022. 7, 15, 20, 85

[88] I. Oleksiienko, D. T. Tran, and A. Iosifidis. Variational neural networks implementation in pytorch and jax. *Software Impacts*, 14:100431, 2022. 7, 20

[89] I. Osband, J. Aslanides, and A. Cassirer. Randomized prior functions for deep reinforcement learning. In *NeurIPS*, volume 31, pages 8626–8638, 2018. 7, 14, 15, 16, 20

[90] I. Osband, Z. Wen, M. Asghari, M. Ibrahimi, X. Lu, and B. V. Roy. Epistemic Neural Networks. 2021. 15, 17, 20

[91] R. Padilla, S. L. Netto, and E. A. Da Silva. A survey on performance metrics for object-detection algorithms. In *Proc. International Conference on Systems, Signals and Image Processing*, pages 237–242, 2020. 20

[92] G. Pandey, J. R. McBride, and R. M. Eustice. Ford campus vision and lidar data set. *The International Journal of Robotics Research*, 30(13):1543–1552, 2011. 40, 41

[93] N. Passalis and A. Tefas. Pseudo-active vision for improving deep visual perception through neural sensory refinement. In *Proc. IEEE International Conference on Image Processing (ICIP)*, pages 2763–2767, 2021. 20

[94] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. 20

[95] L. Porzi, S. R. Bulo, A. Colovic, and P. Kontschieder. Seamless scene segmentation. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, pages 8277–8286, 2019. 12

[96] J. Pöschmann, T. Pfeifer, and P. Protzel. Factor graph based 3d multi-object tracking in point clouds. In *Int. Conf. on Intelligent Robots and Systems*, pages 10343–10350, 2020. 37

[97] H. Qi, C. Feng, Z. Cao, F. Zhao, and Y. Xiao. P2b: Point-to-box network for 3d object tracking in point clouds. *arXiv:2005.13888*, 2020. 26, 29, 30, 31

[98] L. Qi, L. Jiang, S. Liu, X. Shen, and J. Jia. Amodal instance segmentation with kins dataset. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, pages 3014–3023, 2019. 14

[99] I. Radosavovic, R. P. Kosaraju, R. Girshick, K. He, and P. Dollár. Designing network design spaces. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, pages 10428–10436, 2020. 13

[100] N. Radwan, W. Burgard, and A. Valada. Multimodal interaction-aware motion prediction for autonomous street crossing. *The International Journal of Robotics Research*, 39(13):1567–1598, 2020. 33

[101] N. Radwan, A. Valada, and W. Burgard. Vlocnet++: Deep multitask learning for semantic visual localization and odometry. *IEEE Robotics and Automation Letters*, 3(4):4407–4414, 2018. 38

[102] D. R. Rico Jonschkowski and O. Brock. Differentiable particle filters: End-to-end learning with algorithmic priors. In *Proceedings of Robotics: Science and Systems*, 2018. 8

[103] J. Shan, S. Zhou, Z. Fang, and Y. Cui. Ptt: Point-track-transformer module for 3d single object tracking in point clouds. 2021. 26, 27, 29, 30, 31

[104] S. Shi, X. Wang, and H. Li. PointRCNN: 3d object proposal generation and detection from point cloud. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, pages 770–779, 2019. 37

[105] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016. 52

[106] K. Sirohi, R. Mohan, D. Büscher, W. Burgard, and A. Valada. EfficientLPS: Efficient LiDAR panoptic segmentation. *arXiv preprint arXiv:2102.08009*, 2021. 12

[107] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. 15

[108] S. Tang, M. Andriluka, B. Andres, and B. Schiele. Multiple people tracking by lifted multicut and person re-identification. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, 2017. 34

[109] A. Valada, A. Dhall, and W. Burgard. Convoluted mixture of deep experts for robust semantic segmentation. In *IEEE/RSJ International conference on intelligent robots and systems (IROS) workshop, state estimation and terrain perception for all terrain mobile robots*, 2016. 11

[110] M. Valdenegro-Toro. Deep sub-ensembles for fast uncertainty estimation in image classification. *arxiv:1910.08168*, 2019. 15, 16

[111] F. R. Valverde, J. V. Hurtado, and A. Valada. There is more than meets the eye: Self-supervised multi-object detection and tracking with sound by distilling multimodal knowledge. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11612–11621, 2021. 34

[112] N. Vödisch, D. Cattaneo, W. Burgard, and A. Valada. Continual slam: Beyond lifelong simultaneous localization and mapping through continual learning. In *Int. Symposium of Robotics Research*, 2022. 8, 38, 42, 57, 124

[113] H. Wang, C. Wang, and L. Xie. Intensity Scan Context: Coding intensity and geometry relations for loop closure detection. In *Int. Conf. on Robotics and Automation*, pages 2095–2101, 2020. 40, 41

[114] J. Wang, K. Sun, T. Cheng, B. Jiang, C. Deng, Y. Zhao, D. Liu, Y. Mu, M. Tan, X. Wang, et al. Deep high-resolution representation learning for visual recognition. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 43(10):3349–3364, 2020. 12

[115] S. Wang and C. C. Fowlkes. Learning optimal parameters for multi-target tracking with contextual interactions. *Int. Journal of Computer Vision*, 122(3):484–501, 2016. 34

[116] Y. Wang and J. Solomon. Deep Closest Point: Learning representations for point cloud registration. In *Int. Conf. on Computer Vision*, pages 3522–3531, 2019. 40, 41

[117] Y. Wang, Z. Sun, C.-Z. Xu, S. E. Sarma, J. Yang, and H. Kong. LiDAR Iris for loop-closure detection. In *Int. Conf. on Intelligent Robots and Systems*, pages 5769–5775, 2020. 40, 41

[118] Y. Wen, D. Tran, and J. Ba. Batchensemble: An alternative approach to efficient ensemble and lifelong learning. In *ICLR*, volume 8, 2020. 15

[119] X. Weng, J. Wang, D. Held, and K. Kitani. 3d multi-object tracking: A baseline and new evaluation metrics. In *Int. Conf. on Intelligent Robots and Systems*, pages 10359–10366, 2020. 34, 37

[120] X. Weng, Y. Wang, Y. Man, and K. M. Kitani. GNN3DMOT: Graph neural network for 3d multi-object tracking with 2d-3d multi-feature learning. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, pages 6498–6507, 2020. 34, 35

[121] A. G. Wilson and P. Izmailov. Bayesian Deep Learning and a Probabilistic Perspective of Generalization. 2020. 14, 20

[122] Y. Xiao, Y. Xu, Z. Zhong, W. Luo, J. Li, and S. Gao. Amodal segmentation based on visible region segmentation and shape prior. In *AAAI Conference on Artificial Intelligence*, 2021. 14

[123] T.-J. Yang, M. D. Collins, Y. Zhu, J.-J. Hwang, T. Liu, X. Zhang, V. Sze, G. Papandreou, and L.-C. Chen. Deeperlab: Single-shot image parser. *arXiv preprint arXiv:1902.05093*, 2019. 13

[124] T. Yin, X. Zhou, and P. Krahenbuhl. Center-based 3d object detection and tracking. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, pages 11784–11793, 2021. 34, 37

[125] F. Yu, H. Chen, X. Wang, W. Xian, Y. Chen, F. Liu, V. Madhavan, and T. Darrell. Bdd100k: A diverse driving dataset for heterogeneous multitask learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2636–2645, 2020. 14

[126] Z. Yuan, K. Xu, B. Deng, X. Zhou, P. Chen, and Y. Ma. SV-Loop: Semantic-visual loop closure detection with panoptic segmentation. In *2021 IEEE 6th International Conference on Signal and Image Processing (ICSIP)*, pages 245–250, 2021. 38, 39

[127] J.-N. Zaech, A. Liniger, D. Dai, M. Danelljan, and L. Van Gool. Learnable online graph representations for 3d multi-object tracking. *IEEE Robotics and Automation Letters*, pages 1–1, 2022. 34

[128] A. Zamir, A. Dehghan, and M. Shah. Gmcp-tracker: Global multi-object tracking using generalized minimum clique graphs. *Europ. Conf. on Computer Vision*, 2012. 34

[129] J. Zarzar, S. Giancola, and B. Ghanem. Efficient bird eye view proposals for 3d siamese tracking. *arXiv:1903.10168*, 2020. 27, 29, 30

[130] J. Zhang, L. Tai, M. Liu, J. Boedecker, and W. Burgard. Neural slam: Learning to explore with external memory. *arXiv preprint arXiv:1706.09520*, 2017. 8

[131] Y. Zhang, P. Sun, Y. Jiang, D. Yu, Z. Yuan, P. Luo, W. Liu, and X. Wang. Bytetrack: Multi-object tracking by associating every detection box. *arXiv preprint arXiv:2110.06864*, 2021. 34, 36

[132] Z. Zhang. Iterative point matching for registration of free-form curves and surfaces. *Int. Journal of Computer Vision*, 13(2):119–152, 1994. 40

[133] Z. Zhang, S. Lathuilière, E. Ricci, N. Sebe, and J. Yang. Online depth learning against forgetting in monocular videos. In *IEEE Conf. on Computer Vision and Pattern Recognition*, 2020. 41, 42

[134] C. Zheng, X. Yan, J. Gao, W. Zhao, W. Zhang, Z. Li, and S. Cui. Box-aware feature enhancement for single object tracking on point clouds. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 13199–13208, 2021. 26, 30

[135] B. Zhu, Z. Jiang, X. Zhou, Z. Li, and G. Yu. Class-balanced grouping and sampling for point cloud 3d object detection. *arXiv preprint arXiv:1908.09492*, 2019. 37

[136] H. Zou, J. Cui, X. Kong, C. Zhang, Y. Liu, F. Wen, and W. Li. F-siamese tracker: A frustum-based double siamese network for 3d single object tracking. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8133–8139. IEEE, 2020. 27

[137] J. Zürn, W. Burgard, and A. Valada. Self-supervised visual terrain classification from unsupervised acoustic feature learning. *IEEE Transactions on Robotics*, 37(2):466–481, 2020. 11

# 7 Appendix

## 7.1 Perceiving the Invisible: Proposal-Free Amodal Panoptic Segmentation

The appended paper [81] follows.

# Perceiving the Invisible: Proposal-Free Amodal Panoptic Segmentation

Rohit Mohan and Abhinav Valada

***Abstract*—Amodal panoptic segmentation aims to connect the perception of the world to its cognitive understanding. It entails simultaneously predicting the semantic labels of visible scene regions and the entire shape of traffic participant instances, including regions that may be occluded. In this work, we formulate a proposal-free framework that tackles this task as a multi-label and multi-class problem by first assigning the amodal masks to different layers according to their relative occlusion order and then employing amodal instance regression on each layer independently while learning background semantics. We propose the PAPS architecture that incorporates a shared backbone and an asymmetrical dual-decoder consisting of several modules to facilitate within-scale and cross-scale feature aggregations, bilateral feature propagation between decoders, and integration of global instance-level and local pixel-level occlusion reasoning. Further, we propose the amodal mask refiner that resolves the ambiguity in complex occlusion scenarios by explicitly leveraging the embedding of unoccluded instance masks. Extensive evaluation on the BDD100K-APS and KITTI-360-APS datasets demonstrate that our approach set the new state-of-the-art on both benchmarks.**

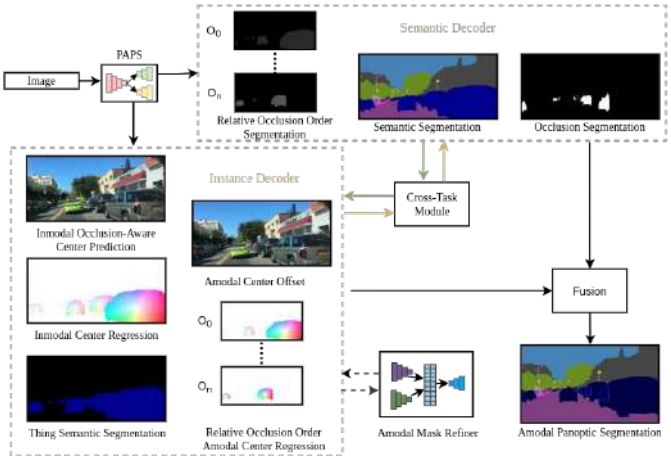



Fig. 1: Overview of our proposed PAPS architecture for amodal panoptic segmentation. Our model predicts multiple outputs from both the semantic and instance decoder. We then fuse the instance-agnostic semantic labels and foreground masks obtained from the segmentation heads with class-agnostic amodal instances that are obtained from the rest of the heads by grouping and majority voting to yield the final amodal panoptic segmentation output.

## I. Introduction

The ability to perceive the entirety of an object irrespective of partial occlusion is known as amodal perception. This ability enables our perceptual and cognitive understanding of the world [1]. The recently introduced amodal panoptic segmentation task [2] seeks to model this ability in robots. The goal of this task is to predict the pixel-wise semantic segmentation labels of the visible amorphous regions of *stuff* classes (e.g., road, vegetation, sky, etc.), and the instance segmentation labels of both the visible and occluded countable object regions of *thing* classes (e.g., cars, trucks, pedestrians, etc.). In this task, each pixel can be assigned more than one class label and instance-ID depending on the visible and occluded regions of objects that it corresponds to, i.e. it allows multi-class and multi-ID predictions. Further, for each segment belonging to a *thing* class, the task requires the knowledge of its visible and occluded regions.

The existing amodal panoptic segmentation approach [2] and baselines [2] follow the proposal-based architectural topology. Proposal-based methods tend to generate overlapping inmodal instance masks as well as multiple semantic predictions for the same pixel, one originating from the instance head and the other from the semantic head, which gives rise to a conflict when fusing the task-specific predictions. This problem is typically tackled using cumbersome heuristics for fusion, requiring multiple sequential processing steps in the pipeline which also tends to favor the amodal instance segmentation branch. On the other hand, proposal-free methods have been more effective in addressing this problem in the closely related panoptic segmentation task [3]–[5] by directly predicting non-overlapping segments. In this work, we aim to alleviate this problem by introducing the first proposal-free framework called Proposal-free Amodal Panoptic Segmentation (PAPS) architecture to address the task of amodal panoptic segmentation. Importantly, to facilitate multi-class and multi-ID predictions, our PAPS decomposes the amodal masks of objects in a given scene into several layers based on their relative occlusion ordering in addition to conventional instance center regression for visible object regions of the scene referred to as inmodal instance center regression. Hence, the network can focus on learning the non-overlapping segments present within each layer. Fig. 1 illustrates an overview of our approach.

Further, amodal panoptic segmentation approaches tend to predict the amodal masks of *thing* class objects by leveraging occlusion features that are conditioned on features of the visible regions. Although it is effective when objects are only partially occluded, it fails in the presence of heavy occlusion as the area of the visible region is reduced. Motivated by humans whose amodal perception is not only based on visible and occlusion cues but also their experience in the world, we propose the amodal mask refiner module to model this capability using explicit memory. This module first predicts an embedding that represents the unoccluded object regions and correlates it with the amodal features generated using either a proposal-free or proposal-based method to complement the lack of visually conditioned occlusion features. We also demonstrate that our amodal mask refiner can be readily incorporated into a variety

Department of Computer Science, University of Freiburg, Germany.
This work was funded by the European Union's Horizon 2020 research and innovation program under grant agreement No 871449-OpenDR.
Supplementary material available on arXiv.

of existing architectures to improve performance.

An interesting aspect of proposal-free methods is that the two sub-tasks, namely, semantic segmentation and instance center regression, are complementary in nature. We leverage this to our benefit and propose a novel cross-task module to bilaterally propagate complementary features between the two sub-tasks decoders for their mutual benefit. Moreover, as rich multi-scale features are important for reliable instance center prediction, we propose the context extractor module that enables within-scale and cross-scale feature aggregation. Finally, to exploit informative occlusion features that play a major role in the amodal mask segmentation quality [2], [6], we incorporate occlusion-aware heads in our PAPS architecture to capture local pixel-wise and global instance-level occlusion information. We present extensive quantitative and qualitative evaluations of PAPS on the challenging BDD100K-APS and KITTI-360-APS datasets, which shows that it achieves state-of-the-art performance. Additionally, we present comprehensive ablation studies to demonstrate the efficacy of our proposed architectural components and we make the models publicly available at http://amodal-panoptic.cs.uni-freiburg.de.

## II. Related Work

Although the amodal panoptic segmentation task [2] is relatively new, the inmodal variant called panoptic segmentation has been extensively studied. We first briefly discuss the methods for panoptic segmentation followed by amodal panoptic segmentation.

*Panoptic Segmentation*: We can categorize existing methods into top-down and bottom-up approaches. Top-down approaches [7]–[10] follow the topology of employing task-specific heads, where the instance segmentation head predicts bounding boxes of objects and its corresponding mask, while the semantic segmentation head outputs the class-wise dense semantic predictions. Subsequently, the outputs of these heads are fused by heuristic-based fusion modules [9], [11]. On the other hand, bottom-up panoptic segmentation methods [4], [5] first perform semantic segmentation, followed by employing different techniques to group [12]–[14] *thing* pixels to obtain instance segmentation. In this work, we follow the aforementioned schema with instance center regression to obtain the panoptic variant of our proposed architecture. Our proposed network modules enrich multi-scale features by enabling feature aggregation from both within-scales and cross-scales. Additionally, our cross-task module facilitates the propagation of complementary features between the different decoders for their mutual benefit.

*Amodal Panoptic Segmentation*: Mohan *et al.* [2] propose several baselines for amodal panoptic segmentation by replacing the instance segmentation head of EfficientPS [9], a top-down panoptic segmentation network, with several existing amodal instance segmentation approaches. EfficientPS employs a shared backbone comprising of an encoder and the 2-way feature pyramid in conjunction with a Mask R-CNN based instance head and a semantic segmentation head, whose outputs are fused to yield the panoptic segmentation prediction. The simple baseline, Amodal-EfficientPS [2], extends EfficientPS with an additional amodal mask head and relies implicitly on the network to capture the relationship between the occluder and occludee. ORCNN [15] further extends it with an invisible mask prediction head to explicitly learn the feature propagation from inmodal mask to amodal mask. Subsequently, ASN [6] employs an occlusion classification branch to model global features and uses a multi-level coding block to propagate these features to the individual inmodal and amodal mask prediction heads. More recently, Shape Prior [16] focuses on leveraging shape priors using a memory codebook with an autoencoder to further refine the initial amodal mask predictions. Alternatively, VQ-VAE [17] utilizes shape priors through discrete shape codes by training a vector quantized variational autoencoder. BCNet [18] seeks to decouple occluding and occluded object instances boundaries by employing two overlapping GCN layers to detect the occluding objects and partially occluded object instances. The most recent, APSNet [2] which is the current state-of-the-art top-down approach focuses on explicitly modeling the complex relationships between the occluders and occludees. To do so, APSNet employs three mask heads that specialize in segmenting visible, occluder, and occlusion regions. It then uses a transformation block with spatio-channel attention for capturing the underlying inherent relationship between the three heads before computing the final outputs. In this work, we present the first bottom-up approach that learns the complex relationship between the occluder and occludee by focusing on learning the relative occlusion ordering of objects. We also employ an occlusion-aware head to explicitly incorporate occlusion information and an amodal mask refiner that aims to mimic the ability of humans by leveraging prior knowledge on the physical structure of objects for amodal perception.

## III. Methodology

In this section, we first describe our PAPS architecture and then detail each of its constituting components. Fig. 2 illustrates the network which follows the bottom-up topology. It consists of a shared backbone followed by semantic segmentation and amodal instance segmentation decoders. The outputs of the decoders are then fused during inference to yield the amodal panoptic segmentation predictions. PAPS incorporates several novel network modules to effectively capture multi-scale features from within-layers and cross-layers, to enable bilateral feature propagation between the task-specific decoders and exploit local and global occlusion information. Further, it incorporates our amodal mask refiner that embeds unoccluded inmodal instance masks to refine the amodal features.

### A. PAPS Architecture

*1) Backbone:* The backbone is built upon HRNet [19] which specializes in preserving high-resolution information throughout the network. It has four parallel outputs with a scale of $\times 4$, $\times 8$, $\times 16$ and $\times 32$ downsampled with respect to the input, namely, $\mathrm{B}_4$, $\mathrm{B}_8$, $\mathrm{B}_{16}$, and $\mathrm{B}_{32}$, as shown in Fig. 2. We then upsample the feature maps to $\times 4$ and concatenate the representations of all the resolutions resulting in $\mathrm{C}_4$, followed by reducing the channels to $256$ with a $1 \times 1$ convolution. Lastly, we aggregate multi-scale features by downsampling high-resolution representations to multiple levels and process each level with a $3 \times 3$ convolution layer ($\mathrm{P}_4$, $\mathrm{P}_8$, $\mathrm{P}_{16}$, $\mathrm{P}_{32}$).

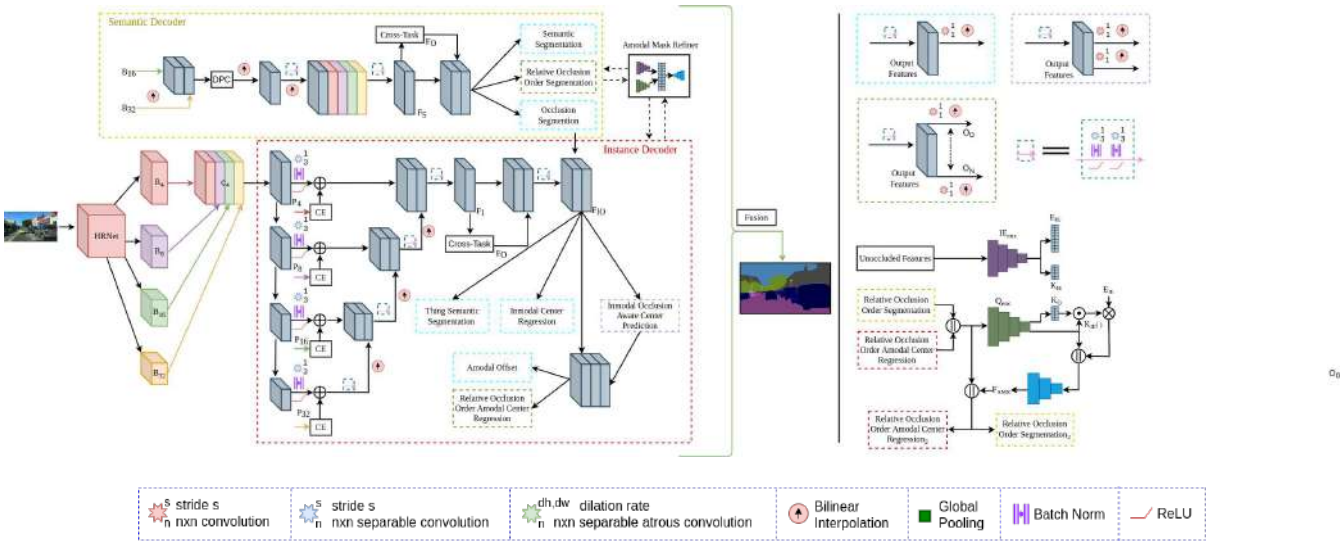



Fig. 2: Illustration of our proposed PAPS architecture consisting of a shared backbone and asymmetric dual-decoder followed by a fusion module that fuses the outputs of the multiple heads of the decoder to yield the amodal panoptic segmentation output. The semantic decoder (yellow-green) and the instance decoder (dark-red) boxes show the topologies of the dual-decoder employed in our architecture. The black-box shows the architecture of our proposed context extractor module. The amodal mask refiner module exploits features from both the decoders to improve amodal masks with embedding correlation.

*2) Context Extractor:* The multi-scale representations from the backbone are computed over all four scales which we refer to as cross-scale features. The way these cross-scale features are computed (concatenation, reduction, and downsampling) leads to a limited exploration for multi-scale features at a given individual scale resolution. Since rich multi-scale representations are crucial for the instance decoder's performance, we seek to enhance the cross-scale features with within-scale contextual features. To do so, we design a lightweight module called the context extractor which is based on the concept of spatial pyramid pooling and is known for efficiently capturing multi-scale contexts from a fixed resolution. We use the context extractor module at each scale ($B_4$, $B_8$, $B_{16}$, $B_{32}$) , and add its output to $P_4$, $P_8$, $P_{16}$, and $P_{32}$, respectively. The proposed context extractor module shown in Fig. S.1 in the supplementary material, employs two $1\times1$ convolutions, two $3\times3$ depth-wise atrous separable convolutions with a dilation rate of $(1, 6)$ and $(3, 1)$, respectively, and a global pooling layer. The output of this module consists of $256$ channels, where $128$ channels are contributed by the $1 \times 1$ convolution and four $32$ channels come from each of the two $3\times3$ depth-wise atrous separable convolutions and its globally pooled outputs. We evaluate the benefits of the aforementioned module in the ablation study presented in Sec. IV-D1.

*3) Cross-Task Module:* The sub-tasks, semantic segmentation and amodal instance center regression, are both distinct recognition problems and yet closely related. The intermediate feature representations of each task-specific decoder can capture complementary features that can assist the other decoder to improve its performance. We propose the cross-task module to enable bilateral feature propagation between the decoders to mutually benefit each other. Given feature inputs $F_I$ and $F_S$ from the two decoders, we fuse them adaptively by employing cross-attention followed by self-attention as

$$F_R = (1 - g_1(F_S)) \cdot F_I + (1 - g_2(F_I)) \cdot F_S, \quad (1)$$

$$F_O = g_3(F_R) \cdot F_R, \quad (2)$$

where $g_1(\cdot)$, $g_2(\cdot)$, and $g_3(\cdot)$ are functions to compute feature confidence score of $F_S$ and $F_I$. These functions consist of a global pooling layer, followed reducing the channels from $256$ to $64$ using a $1 \times 1$ convolution. Subsequently, we employ another $1 \times 1$ convolution with $256$ output channels to remap from the lower dimension to a higher dimension and apply a sigmoid activation to obtain the feature confidence scores. $F_O$ is the output of the cross-task module. The cross-attention mechanism in this module enables $F_I$ and $F_S$ to adaptively complement each other, whereas the following self-attention mechanism enables enhancing the highly discriminative complementary features. The ablation study presented in Sec. IV-D1 shows the influence in performance due to this module.

*4) Semantic Decoder:* The semantic decoder takes $B_{32}$, $B_{16}$, $C_4$ feature maps and the output of cross-task module as its input. First, the $B_{32}$ feature maps are upsampled ($\times16$) and concatenated with $B_{16}$ and are fed to the dense prediction cell (DPC) [20]. The output of DPC is then upsampled ($\times8$) and passed through two sequential $3 \times 3$ depth-wise separable convolutions. Subsequently, we again upsample ($\times4$) and concatenate it with $C_4$. We then employ two sequential $3 \times 3$ depth-wise separable convolutions and feed the output ($F_S$) to the cross-task module. Further, we concatenate $F_S$ with the output of the cross-task module ($F_O$) and feed it to the multiple heads in the semantic decoder.

We employ three heads, namely, relative occlusion order segmentation ($L_{roo}$), semantic segmentation ($L_{ss}$), and occlusion segmentation ($L_{os}$), towards the end of our semantic decoder. The relative occlusion order segmentation head predicts foreground mask segmentation for $O_N$ layers. The masks of each layer are defined as follows: All unoccluded class-agnostic *thing* object masks belong to layer $0$ ($O_0$). Next, layer $1$ ($O_1$) comprises amodal masks of any occluded object that are occluded by layer $0$ objects but not occluded by any other occluded object. Next, layer $2$ ($O_2$) consists of amodal masks of any occluded object, not in the previous layers that

are occluded by layer 1 objects but not occluded by any other occluded objects that are not part of previous layers and so on. Fig. 3 illustrates the separation of *thing* amodal object segments into relative occlusion ordering layers. This separation ensures each *thing* amodal object segment belongs to a unique layer without any overlaps within that layer. We use the binary cross-entropy loss ($L_{roo}$) to train this head. Next, the semantic segmentation head predicts semantic segmentation of both *stuff* and *thing* classes, and we employ the weighted bootstrapped cross-entropy loss [21] ($L_{ss}$) for training. Lastly, the occlusion segmentation head predicts whether a pixel is occluded in the context of *thing* objects and we use the binary cross-entropy loss ($L_{occ}$) for training. The overall semantic decoder loss is given as

$$L_{sem} = L_{ss} + L_{os} + L_{roo}. \quad (3)$$

The predictions from all the heads of the semantic decoder are used in the fusion module to obtain the final amodal panoptic segmentation prediction.

*5) Instance Decoder:* The instance decoder employs a context encoder at each scale ($B_{32}$, $B_{16}$, $B_8$, $B_4$) and adds the resulting feature maps to $P_{32}$, $P_{16}$, $P_8$, and $P_4$, respectively. Then, beginning from ($\times$32), the decoder repeatedly uses a processing block consisting of two sequential $3 \times 3$ depth-wise separable convolutions, upsamples it to the next scale ($\times$16), and concatenates with the existing features of the next scale until $\times$4 feature resolution is obtained ($F_I$). The $F_I$ is then fed to the cross-task module. The cross-task output $F_O$ is concatenated with $F_I$ and is processed by two sequential $3 \times 3$ depth-wise separable convolutions. Subsequently, the features from the occlusion segmentation head of the semantic decoder are concatenated to incorporate explicit pixel-wise local occlusion information referred to as $F_{IO}$ features.

The instance decoder employs five prediction heads. The inmodal occlusion-aware center prediction head consists of two prediction branches, one for predicting the center of mass heatmap of inmodal *thing* object instances ($L_{icp}$) and the other for predicting whether the heatmap is occluded ($L_{ico}$). For the former, we use the Mean Squared Error (MSE) loss ($L_{icp}$) to minimize the distance between the 2D Gaussian encoded groundtruth heatmaps and the predicted heatmaps, for training. For the latter, we use binary cross-entropy loss ($L_{ico}$) for training. Following, the *thing* semantic segmentation ($L_{tss}$) head predicts $N_{thing}$+1 classes, where $N_{thing}$ is the total number of *thing* semantic classes and the '+1' class predicts all *stuff* classes as a single class. This head is trained with the weighted bootstrapped cross-entropy loss [21] ($L_{tss}$). Next, the inmodal center regression ($L_{icr}$) head predicts the offset from each pixel location belonging to *thing* classes to its corresponding inmodal object instance mass center. We use the $L_1$ loss for training this head ($L_{icr}$). All the aforementioned heads take $F_{IO}$ features as input.

The remaining heads of the instance decoder are referred to as the amodal center offset ($L_{aco}$) and relative occlusion order amodal center regression ($L_{rooacr}$). The amodal center offset head predicts the offset from each inmodal object instance center to its corresponding amodal object instance center. Whereas, the relative occlusion ordering amodal center regression head, for each relative occlusion ordering layer, predicts the offset from each pixel location belonging to *thing* classes of the layer to its corresponding amodal object instance mass center. Here, the layers of relative occlusion ordering are defined similarly as in the semantic decoder. Further, we concatenate $F_{IO}$ with features of inmodal occlusion-aware center prediction head to incorporate object-level global occlusion features before feeding it to the aforementioned heads. Finally, we use $L_1$ loss to train both the heads ($L_{aco}$, $L_{rooacr}$). The overall loss for the instance decoder is

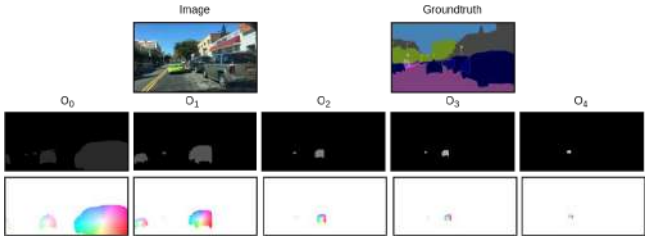


Fig. 3: Groundtruth examples for relative occlusion order segmentation (top-row) and instance center regression (bottom-row) consisting of layer from $O_0$ to $O_5$. Best viewed at $\times$4 zoom.

$$L_{inst} = L_{tss} + L_{ico} + \alpha L_{icp} + \beta(L_{icr} + L_{aco} + L_{rooacr}), \quad (4)$$

where the loss weights $\alpha = 200$ and $\beta = 0.01$.

Note that we learn amodal center offset instead of the amodal center itself to have a common instance-ID that encapsulates both the amodal and inmodal masks.

*6) Amodal Mask Refiner:* We propose the amodal mask refiner module to model the ability of humans to leverage priors on complete physical structures of objects for amodal perception, in addition to visually conditioned occlusion cues. This module builds an embedding that embeds the features of the unoccluded object mask and correlates them with the generated amodal features to complement the lack of visually conditioned occlusion features. The amodal mask refiner shown in Fig. 2 consists of two encoders, unoccluded feature embeddings, and a decoder. We employ the RegNet [22] topology with its first and last stages removed as the two encoders with feature encoding resolution of $\times$16 downsampled with respect to the input. The two encoders are an inmodal embedding encoder ($IE_{enc} \in \mathbb{R}^{(H/16)\times(W/16)\times C}$) that encodes unoccluded objects features and a query encoder ($Q_{enc} \in \mathbb{R}^{(H/16)\times(W/16)\times C}$) that encodes the amodal features, where $H$ and $W$ are the height and width of the input image and $C$ is the feature dimension which is set to 64. Subsequently, an embedding matrix $E_{IE} \in \mathbb{R}^{N\times D}$ embeds the $IE_{enc}$ encoding to create the embedding of unoccluded object masks. Further, to extract the mask embedding information from $E_{IE}$, we compute two key matrices, namely, $K_{IE} \in \mathbb{R}^{N\times D}$ matrix and $K_Q \in \mathbb{R}^{1\times D}$ matrix, from $IE_{enc}$ and $Q_{enc}$ encodings, respectively. Here, $N = 128$ and $D = [(H/16) \times (W/16) \times C]$.

Next, we compute the inner product of $K_{IE}$ and $K_Q$ followed by a softmax and take the inner product of the resulting probability and $E_{IE}$. We then rearrange this output into $(H/16) \times (W/16) \times C$ shape and concatenate it with $Q_{enc}$ and feed it to the decoder. The decoder employs repeated blocks of two $3 \times 3$ depth-wise separable convolutions, followed by a bilinear interpolation to upsample by a factor of 2 until the upsampled output resolution is $\times$4 downsampled with respect to the input. We refer to this output as $F_{AMR}$. The resulting features enrich the amodal features of occluded objects with similar

unoccluded object features, thereby enabling our network to predict more accurate amodal masks.

The amodal mask refiner takes two inputs, namely, the amodal features and the features of the unoccluded objects. The input amodal features are obtained by concatenating the output features (Fig. 2) of relative occlusion ordering heads of the semantic and instance decoders. To compute the features of the unoccluded object, we first perform instance grouping using predictions of the inmodal occlusion-aware, inmodal center regression, and thing semantic segmentation heads to obtain the inmodal instance masks. We then discard all the occluded inmodal instances to generate an unoccluded instance mask. Next, we multiply the aforementioned mask with the output of the second layer of the inmodal center regression head to compute the final unoccluded object features. Finally, the amodal mask refiner outputs $\mathrm{F}_{\mathrm{AMR}}$ which is then concatenated with the amodal features. We employ two similar heads as relative occlusion ordering amodal center regression and segmentation that takes the aforementioned concatenated features as input. We use the same loss functions and loss weights for training the heads as described in Sec. III-A5.

*7) Inference:* We perform a series of steps during inference to merge the outputs of the semantic and instance decoders to yield the final amodal panoptic segmentation. We begin with computing the semantic segmentation prediction and the *thing* foreground mask. To do so, we duplicate the void class logit of the *thing* semantic segmentation head logits $N_{stuff}$-times, such that its number of channels transforms from $1 + N_{thing}$ to $N_{stuff} + N_{thing}$. We then add it to the logits of the semantic segmentation head and employ a softmax followed by an argmax function to obtain the final semantic segmentation prediction. Subsequently, we assign $0$ to all the *stuff* classes and $1$ to all the *thing* classes to obtain the *thing* foreground mask. Next, we obtain the inmodal center point predictions by employing a keypoint-based non-maximum suppression [5] and confidence thresholding $(0.1)$ to filter out the low confidence predictions while keeping only the top-k $(200)$ highest confidence scores on the heatmap prediction of inmodal occlusion-aware center prediction head. We then obtain the amodal center points predictions by applying the corresponding offsets from the amodal instance head to the inmodal center point predictions. We obtain the class-agnostic instance-IDs and the inmodal instance mask using simple instance grouping [5] with the inmodal center prediction and the *thing* foreground mask. Further, we compute semantic labels for each instance-ID by the majority vote of the corresponding predicted semantic labels with its inmodal instance masks.

Now, for each instance-ID, we have its semantic label, inmodal mask, and the amodal center prediction. We compute the relative occlusion order segmentation masks for each layer by applying a threshold of $0.5$ on the outputs of the relative occlusion ordering segmentation head connected to the amodal mask refiner. We then assign the instance-ID to its corresponding relative occlusion ordering layer by checking if the corresponding amodal center lies within the segmentation mask of the layer in question. Finally, we again use the simple instance grouping at each of the relative occlusion ordering layers. For all instance-IDs belonging to a layer, we apply the instance grouping using its amodal instance center and regression along with the corresponding segmentation mask to compute the amodal mask. In the end, for each *thing* object, we have its unique instance-ID, semantic label, inmodal, and amodal mask along with *stuff* class semantic predictions from the semantic segmentation prediction. We obtain the visible attribute of the amodal mask directly from the inmodal mask and obtain the occluded attributes of the amodal mask by removing the inmodal mask segment from the amodal mask.

## IV. Experimental Evaluation

In this section, we describe the datasets that we benchmark on in Sec. IV-A and the training protocol in Sec. IV-B. We then present extensive benchmarking results in Sec. IV-C, followed by a detailed ablation study on the architectural components in Sec. IV-D and qualitative comparisons in Sec. IV-E. We use the standard Amodal Panoptic Quality (APQ) and Amodal Parsing Coverage (APC) metrics [2] to quantify the performance.

### A. Datasets

*KITTI-360-APS* [2] provides amodal panoptic annotations for the KITTI-360 [23] dataset. It consists of $9$ sequences of urban street scenes with annotations for $61{,}168$ images. The sequence numbered $10$ of the dataset is treated as the validation set. This dataset comprises $7$ *thing* classes, namely, car, pedestrians, cyclists, two-wheeler, van, truck, and other vehicles. Further, the dataset consists of $10$ *stuff* classes. These stuff classes are road, sidewalk, building, wall, fence, pole, traffic sign, vegetation, terrain, and sky.

*BDD100K-APS* [2] extends the BDD100K [24] dataset with amodal panoptic annotations for $15$ of its sequences consisting of $202$ images per sequence. The training and validation set consists of $12$ and $3$ sequences, respectively. Pedestrian, car, truck, rider, bicycle, and bus are the $6$ *thing* classes. Whereas, road, sidewalk, building, fence, pole, traffic sign, fence, terrain, vegetation, and sky are the $10$ *stuff* classes

### B. Training Protocol

All our models are trained using the PyTorch library on $8$ NVIDIA TITAN RTX GPUs with a batch size of $8$. We train our network in two stages, with a crop resolution of $376 \times 1408$ pixels and $448 \times 1280$ pixels for the KITTI-360-APS and BDD100K-APS datasets, respectively. For each stage, we use the Adam optimizer with a poly learning rate schedule, where the initial learning rate is set to $0.001$. We train our model for $300$K iterations for the KITTI-360-APS dataset and $70$K iterations for the BDD100K-APS dataset, while using random scale data augmentation within the range of $[0.5, 2.0]$ with flipping for each stage. We use $N = 8$ for relative occlusion order layers. We first train the model without the amodal mask refiner, followed by freezing the weights of the architectural components from the previous stage and train only the amodal mask refiner.

### C. Benchmarking Results

In this section, we present results comparing the performance of our proposed PAPS architecture against current state-of-the-art amodal panoptic segmentation approaches. We report the

TABLE I: Comparison of amodal panoptic segmentation benchmarking results on the KITTI-360-APS and BDD100K-APS validation set. Subscripts $S$ and $T$ refer to *stuff* and *thing* classes respectively. All scores are in [%].

| Model | KITTI-360-APS | | | | | | BDD100K-APS | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | APQ | APC | $\text{APQ}_S$ | $\text{APQ}_T$ | $\text{APC}_S$ | $\text{APC}_T$ | APQ | APC | $\text{APQ}_S$ | $\text{APQ}_T$ | $\text{APC}_S$ | $\text{APC}_T$ |
| Amodal-EfficientPS | 41.1 | 57.6 | 46.2 | 33.1 | 58.1 | 56.6 | 44.9 | 46.2 | 54.9 | 29.9 | 64.7 | 41.4 |
| ORCNN [15] | 41.1 | 57.5 | 46.2 | 33.1 | 58.1 | 56.6 | 44.9 | 46.2 | 54.9 | 29.9 | 64.7 | 41.5 |
| BCNet [18] | 41.6 | 57.9 | 46.2 | 34.4 | 58.1 | 57.6 | 45.2 | 46.4 | 55.0 | 30.7 | 64.7 | 42.1 |
| VQ-VAE [17] | 41.7 | 58.0 | 46.2 | 34.6 | 58.1 | 57.8 | 45.3 | 46.5 | 54.9 | 30.8 | 64.7 | 42.2 |
| Shape Prior [16] | 41.8 | 58.2 | 46.2 | 35.0 | 58.1 | 58.2 | 45.4 | 46.6 | 55.0 | 31.0 | 64.8 | 42.6 |
| ASN [6] | 41.9 | 58.2 | 46.2 | 35.2 | 58.1 | 58.3 | 45.5 | 46.6 | 55.0 | 31.2 | 64.8 | 42.7 |
| APSNet [2] | 42.9 | 59.0 | 46.7 | 36.9 | 58.5 | 59.9 | 46.3 | 47.3 | 55.4 | 32.8 | 65.1 | 44.5 |
| PAPS (Ours) | **44.6** | **61.4** | **47.5** | **40.1** | **59.2** | **64.7** | **48.7** | **50.4** | **56.5** | **37.1** | **66.4** | **51.6** |

APQ and APC metrics of the existing state-of-the-art methods directly from the published manuscript [2]. Tab. I presents the benchmarking results on both datasets.

We observe that our proposed PAPS architecture achieves the highest APQ and APC scores compared to APSNet and other baselines on both datasets. The improvement of $1.7\%$-$2.7\%$ in both the metrics can be attributed to the various proposed components of our architecture. For *stuff* segmentation, the complementary features from the cross-task module aid in better distinguishing *stuff* and *thing* classes, while the high resolution features with the long-range contextual features help in finer segmentation of the boundaries. Consequently, we observe an improvement of $0.7\%$-$1.3\%$ in the *stuff* components of the metrics for both datasets. The *thing* components of the metrics achieve an improvement of $3.2\%$-$7.1\%$ which can be attributed to the synergy of several factors. The context extractor and the cross-task modules provide richer multi-scale representations along with complementary semantic decoder features. This enables reliable segmentation of far-away small-scale instances. Further, the incorporation of local and object-level global occlusion information from the instance and semantic decoder heads enables explicit amodal reasoning capabilities. We also believe that the relative occlusion ordering layers force the network to capture the complex underlying relationship of objects to one another in the context of occlusions. Lastly, the amodal mask refiner module with its transformation of amodal features with unoccluded object mask embeddings improves the quality of large occlusion area segmentation as observed from the higher improvement in APC than the APQ metric. Overall, PAPS establishes the new state-of-the-art on both the amodal panoptic segmentation benchmarks.

### D. Ablation Study

In this section, we first study the improvement due to the various architectural components that we propose in our PAPS and study the generalization ability of the amodal mask refiner by incorporating it in various proposal-based methods. We then evaluate the performance of PAPS for panoptic segmentation and amodal instance segmentation tasks.

*1) Detailed Study on the PAPS Architecture:* In this section, we quantitatively evaluate the influence of each proposed architectural component in PAPS, on the overall performance. Here, the addition of modules to the architecture of the base model M1 in the incremental form is performed according to their description in Sec. III. Tab. II presents results from this experiment. We begin with the model M1 which employs a semantic decoder as described in Sec. III-A4 without any cross-task module and occlusion segmentation head and is similar to [5] with amodal capabilities. For the instance decoder, it employs the aforementioned semantic decoder with the heads described in Sec. III-A5 without occlusion-awareness of center and *thing* semantic segmentation. In the M2 model, we replace the instance decoder architecture with that described in Sec. III-A5 without the cross-task module and the same heads as the M1 model. The improvement in performance shows the importance of multi-scale features from cross-layers for amodal instance center regression. In the M3 model, we add the *thing* segmentation head to the instance decoder whose output is used during inference as described in Sec. III-A7. The improvement achieved indicates that the two decoders capture diverse representations of *thing* classes which further improves the performance.

TABLE II: Evaluation of various architectural components of our proposed PAPS model. The performance is shown for the models trained on the BDD100K-APS dataset and evaluated on the validation set. Subscripts $S$ and $T$ refer to *stuff* and *thing* classes respectively. All scores are in [%].

| Model | APQ | APC | $\text{APQ}_S$ | $\text{APQ}_T$ | $\text{APC}_S$ | $\text{APC}_T$ |
|---|---|---|---|---|---|---|
| M1 | 45.6 | 46.9 | 55.8 | 30.4 | 65.7 | 42.2 |
| M2 | 45.9 | 47.1 | 55.8 | 31.0 | 65.7 | 42.7 |
| M3 | 46.1 | 47.2 | 55.9 | 31.3 | 65.8 | 42.9 |
| M4 | 46.3 | 47.3 | 55.9 | 31.9 | 65.8 | 43.3 |
| M5 | 46.7 | 47.7 | 56.3 | 32.4 | 66.2 | 43.9 |
| M6 | 47.4 | 48.5 | **56.5** | 33.7 | **66.4** | 45.8 |
| M7 (PAPS ) | **48.7** | **50.4** | **56.5** | **37.1** | **66.4** | **51.6** |

In the M4 model, we add the context extractor module. The higher increase in $\text{APQ}_T$ compared to $\text{APC}_T$ indicates that the multi-scale features obtained from the aggregation of within-scales and cross-scales layers are much richer in the representation capacity, thereby improving the detection of small far away objects. Building upon M4, in the M5 model, we add the cross-task module. The increase in both *stuff* and *thing* components of the metrics demonstrates that the two decoders learn complementary features which when propagated bidirectionally is mutually beneficial for each of them. In the M6 model, we add the occlusion segmentation head and occlusion awareness to the inmodal center prediction head. We observe an improvement of $1.3\%$-$1.9\%$ in *thing* components of the metrics demonstrating that the incorporation of occlusion information is integral for good amodal mask segmentation. Lastly, in the M7 model, we add the amodal mask refiner. The substantial improvement of $3.4\%$ and $5.8\%$ in $\text{APQ}_T$ and $\text{APC}_T$, respectively, demonstrates the efficacy of our proposed module. We note that the improvement in $\text{APC}_T$ is higher than $\text{APQ}_T$ indicating

TABLE III: Evaluation of various propsal-based amodal panoptic segmentation approaches with our proposed amodal mask refiner. The performance is shown for the models trained on the BDD100K-APS dataset and evaluated on the validation set. Subscript $T$ refer to *thing* classes. All scores are in [%].

| Model | Amodal Mask Refiner | APQ | APC | $APQ_T$ | $APC_T$ |
|---|---|---|---|---|---|
| ORCNN [15] | | 44.9 | 46.2 | 29.9 | 41.4 |
| BCNet [18] | | 45.2 | 46.4 | 30.7 | 42.1 |
| ASN [6] | | 45.5 | 46.6 | 31.2 | 42.7 |
| APSNet [2] | | 46.3 | 47.3 | 32.8 | 44.5 |
| ORCNN [15] | ✓ | 45.3 | 46.6 | 30.9 | 42.8 |
| BCNet [18] | ✓ | 46.3 | 47.8 | 33.2 | 46.4 |
| ASN [6] | ✓ | 46.7 | 48.1 | 34.4 | 47.1 |
| APSNet [2] | ✓ | **47.5** | **48.9** | **35.9** | **49.2** |

TABLE IV: Performance comparison of panoptic segmentation on the Cityscapes validation set. – denotes that the metric has not been reported for the corresponding method. All scores are in [%].

| Network | PQ | SQ | RQ | $PQ_T$ | $PQ_S$ | AP | mIoU |
|---|---|---|---|---|---|---|---|
| Panoptic FPN [25] | 58.1 | – | – | 52.0 | 62.5 | 33.0 | 75.7 |
| UPSNet [11] | 59.3 | 79.7 | 73.0 | 54.6 | 62.7 | 33.3 | 75.2 |
| DeeperLab [21] | 56.3 | – | – | – | – | – | – |
| Seamless [7] | 60.3 | – | – | 56.1 | 63.3 | 33.6 | 77.5 |
| SSAP [4] | 61.1 | – | – | 55.0 | – | – | – |
| AdaptIS [3] | 62.0 | – | – | 58.7 | 64.4 | 36.3 | 79.2 |
| Panoptic-DeepLab [5] | 63.0 | – | – | – | – | 35.3 | 80.5 |
| EfficientPS [9] | 63.9 | 81.5 | 77.1 | **60.7** | 66.2 | **38.3** | 79.3 |
| PAPS (ours) | **64.3** | **82.1** | **77.3** | 60.1 | **67.3** | 37.2 | **80.8** |

that the increase in segmentation quality of objects with larger occlusion areas is relatively higher than the smaller areas. This result precisely demonstrates the utility of our proposed amodal mask refiner, validating our idea of using embeddings of non-occluded object masks to supplement the amodal features with correlation for mid-to-heavy occlusion cases.

*2) Generalization of amodal mask refiner:* In this section, we study the generalization ability of our proposed amodal mask refiner by incorporating it in existing proposal-based amodal panoptic segmentation approaches. To do so, we adapt the amodal mask refiner by removing all downsampling layers in the encoders and upsampling layers from its decoder, to make it compatible with proposal-based approaches. We add an occlusion classification branch in the amodal instance head of all the proposal-based methods similar to ASN [6] and add another identical amodal mask head. The output of the fourth layer of the amodal mask head of each method is considered as the amodal features input. For the non-occluded object features, we multiply the output of the occlusion classification branch with the output of the fourth layer of the inmodal mask head. We feed the amodal features and non-occluded object features to the amodal mask refiner, followed by concatenating its output with the amodal features. Subsequently, we feed these concatenated features to the newly added amodal mask head. To train the networks, we use the same two-stage procedure described in Sec. IV-B and the training protocol described in [2].

Tab. III presents the results from this experiment. We observe a considerable improvement in the performance of all the proposal-based methods demonstrating the effectiveness and the ease of integration into existing architectures. Moreover, the improvement achieved for APSNet is higher than ORCNN indicating that the performance can vary depending on the quality of the inmodal and amodal feature representations in the network.

TABLE V: Amodal instance segmentation results on the KINS dataset. All scores are in [%].

| Model | $Amodal_{AP}$ | $Inmodal_{AP}$ |
|---|---|---|
| ORCNN [15] | 29.0 | 26.4 |
| VQ-VAE [17] | 31.5 | – |
| Shape Prior [16] | 32.1 | 29.8 |
| ASN [6] | 32.2 | 29.7 |
| APSNet [2] | 35.6 | 32.7 |
| PAPS (Ours) | **37.4** | **33.1** |

*3) Panoptic Segmentation Results on Cityscapes Dataset:* In this section, we evaluate the performance of our proposed PAPS for panoptic segmentation on the Cityscapes [26] dataset. In the architecture, we remove the amodal mask refiner, occlusion segmentation, amodal center offset, relative occlusion order segmentation, and amodal center regression heads as they only contribute to obtaining the amodal masks. We train our network with a learning rate $lr = 0.001$ for 90K iterations using the Adam optimizer. We report the Panoptic Quality (PQ), Segmentation Quality (SQ) and Recognition Quality (RQ) metrics on the validation set of Cityscapes for single-scale evaluation in Tab. IV. For the sake of completeness, we also report the Average Precision (AP), and the mean Intersection-over-Union (mIoU) scores. We observe that PAPS achieves the highest PQ score of $64.3\%$ which is $1.3\%$ and $0.4\%$ higher than the state-of-the-art Panoptic-DeepLab and EfficientPS, respectively. The improvement achieved over Panoptic-DeepLab demonstrates the efficacy of our proposed modules and architectural design choices.

*4) Performance on KINS Dataset:* We benchmark the performance of our proposed PAPS architecture on the KINS [6] amodal instance segmentation benchmark. This benchmark uses the Average Precision (AP) metric for evaluating both amodal and inmodal segmentation. We train our network with a learning rate $lr = 0.001$ for $40$K iterations using the Adam optimizer. We use the same validation protocols as [6]. Tab. V presents results in which our proposed PAPS outperforms the state-of-the-art APSNet by $1.8\%$ and $0.4\%$ for amodal AP and inmodal AP, respectively, establishing the new state-of-the-art on this benchmark. The large improvement in the $\text{Amodal}_{\text{AP}}$ compared to the $\text{Inmodal}_{\text{AP}}$ indicates refining amodal masks with unoccluded object embeddings is an effective strategy.

### *E. Qualitative Evaluations*

In this section, we qualitatively compare the amodal panoptic segmentation performance of our proposed PAPS architecture with the previous state-of-the-art APSNet. Fig. 4 presents the qualitative results. We observe that both approaches are capable of segmenting partial occlusion cases. However, our PAPS outperforms APSNet under moderate to heavy occlusion cases such as cluttered cars and pedestrians. In Fig. 4(a) the faraway cars on the right are detected more reliably by our network along with their amodal mask segmentations demonstrating the positive effects of within-scales and cross-scales multi-scale features and the occlusion aware heads. In Fig. 4(b), our model successfully predicts the amodal masks of heavily occluded pedestrians and cars. This demonstrates the utility of our amodal mask refiner module. By relying on the unoccluded mask features, PAPS is able to make a coarse estimate of

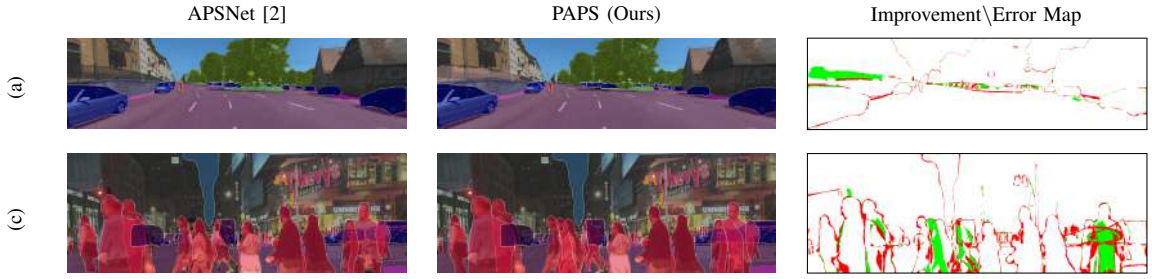



Fig. 4: Qualitative amodal panoptic segmentation results of our proposed PAPS network in comparison to the state-of-the-art APSNet [2] on (a) KITTI-360-APS and (b) BDD100K-APS datasets. We also show the Improvement\Error Map which denotes the pixels that are misclassified by PAPS in red and the pixels that are misclassified by APSNet but correctly predicted by PAPS in green.

the object's amodal masks. Furthermore, PAPS achieves more accurate segmentation of the challenging thin *stuff* classes such as poles and fences.

## V. Conclusion

In this work, we presented the first proposal-free amodal panoptic segmentation architecture that achieves state-of-the-art performance on both the KITTI-360-APS and BDD100K-APS datasets. To facilitate learning proposal-free amodal panoptic segmentation, our PAPS network learns amodal center offsets from the inmodal instance center predictions while decomposing the scene into different relative occlusion ordering layers such that there are no overlapping amodal instance masks within a layer. It further incorporates several novel network modules to capture within-layer multi-scale features for richer multi-scale representations, to enable bilateral propagation of complementary features between the decoders for their mutual benefit, and to integrate global and local occlusion features for effective amodal reasoning. Furthermore, we proposed the amodal mask refiner module that improves the amodal segmentation performance of occluded objects for both proposal-free and proposal-based architectures. Additionally, we presented detailed ablation studies and qualitative evaluations highlighting the improvements that we make to various core network modules of our amodal panoptic segmentation architectures. Finally, we have made the code and models publicly available to accelerate further research in this area.

## References


[1] B. Nanay, "The importance of amodal completion in everyday perception," *i-Perception*, vol. 9, no. 4, 2018.

[2] R. Mohan and A. Valada, "Amodal panoptic segmentation," *arXiv preprint arXiv:2202.11542*, 2022.

[3] K. Sofiiuk, O. Barinova, and A. Konushin, "Adaptis: Adaptive instance selection network," in *Int. Conf. on Computer Vision*, 2019, pp. 7355–7363.

[4] N. Gao, Y. Shan, Y. Wang, X. Zhao, Y. Yu, M. Yang, and K. Huang, "Ssap: Single-shot instance segmentation with affinity pyramid," in *Int. Conf. on Computer Vision*, 2019.

[5] B. Cheng, M. D. Collins, Y. Zhu, T. Liu, T. S. Huang, H. Adam, and L.-C. Chen, "Panoptic-deeplab: A simple, strong, and fast baseline for bottom-up panoptic segmentation," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, 2020.

[6] L. Qi, L. Jiang, S. Liu, X. Shen, and J. Jia, "Amodal instance segmentation with kins dataset," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, 2019, pp. 3014–3023.

[7] L. Porzi, S. R. Bulo, A. Colovic, and P. Kontschieder, "Seamless scene segmentation," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, 2019, pp. 8277–8286.

[8] N. Gosala and A. Valada, "Bird's-eye-view panoptic segmentation using monocular frontal view images," *IEEE Robotics and Automation Letters*, 2022.

[9] R. Mohan and A. Valada, "Efficientps: Efficient panoptic segmentation," *Int. Journal of Computer Vision*, vol. 129, no. 5, pp. 1551–1579, 2021.

[10] K. Sirohi, R. Mohan, D. Büscher, W. Burgard, and A. Valada, "Efficientlps: Efficient lidar panoptic segmentation," *IEEE Transactions on Robotics*, 2021.

[11] Y. Xiong, R. Liao, H. Zhao, R. Hu, M. Bai, E. Yumer, and R. Urtasun, "Upsnet: A unified panoptic segmentation network," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, 2019, pp. 8818–8826.

[12] B. Leibe, A. Leonardis, and B. Schiele, "Combined object categorization and segmentation with an implicit shape model," in *Workshop on statistical learning in computer vision, ECCV*, 2004.

[13] J. Uhrig, E. Rehder, B. Fröhlich, U. Franke, and T. Brox, "Box2pix: Single-shot instance segmentation by assigning pixels to object boxes," in *IEEE Intelligent Vehicles Symposium*, 2018, pp. 292–299.

[14] F. R. Valverde, J. V. Hurtado, and A. Valada, "There is more than meets the eye: Self-supervised multi-object detection and tracking with sound by distilling multimodal knowledge," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 11 612–11 621.

[15] P. Follmann, R. König, P. Härtinger, M. Klostermann, and T. Böttger, "Learning to see the invisible: End-to-end trainable amodal instance segmentation," in *IEEE Winter Conference on Applications of Computer Vision*, 2019, pp. 1328–1336.

[16] Y. Xiao, Y. Xu, Z. Zhong, W. Luo, J. Li, and S. Gao, "Amodal segmentation based on visible region segmentation and shape prior," in *AAAI Conference on Artificial Intelligence*, 2021.

[17] W.-D. Jang, D. Wei, X. Zhang, B. Leahy, H. Yang, J. Tompkin, D. Ben-Yosef, D. Needleman, and H. Pfister, "Learning vector quantized shape code for amodal blastomere instance segmentation," *arXiv preprint arXiv:2012.00985*, 2020.

[18] L. Ke, Y.-W. Tai, and C.-K. Tang, "Deep occlusion-aware instance segmentation with overlapping bilayers," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, 2021, pp. 4019–4028.

[19] J. Wang, K. Sun, T. Cheng, B. Jiang, C. Deng, Y. Zhao, D. Liu, Y. Mu, M. Tan, X. Wang, *et al.*, "Deep high-resolution representation learning for visual recognition," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 43, no. 10, pp. 3349–3364, 2020.

[20] L.-C. Chen, M. Collins, Y. Zhu, G. Papandreou, B. Zoph, F. Schroff, H. Adam, and J. Shlens, "Searching for efficient multi-scale architectures for dense image prediction," in *Advances in Neural Information Processing Systems*, 2018, pp. 8713–8724.

[21] T.-J. Yang, M. D. Collins, Y. Zhu, J.-J. Hwang, T. Liu, X. Zhang, V. Sze, G. Papandreou, and L.-C. Chen, "Deeperlab: Single-shot image parser," *arXiv preprint arXiv:1902.05093*, 2019.

[22] I. Radosavovic, R. P. Kosaraju, R. Girshick, K. He, and P. Dollár, "Designing network design spaces," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, 2020, pp. 10 428–10 436.

[23] Y. Liao, J. Xie, and A. Geiger, "KITTI-360: A novel dataset and benchmarks for urban scene understanding in 2d and 3d," *arXiv preprint arXiv:2109.13410*, 2021.

[24] F. Yu, H. Chen, X. Wang, W. Xian, Y. Chen, F. Liu, V. Madhavan, and T. Darrell, "Bdd100k: A diverse driving dataset for heterogeneous multitask learning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 2636–2645.

[25] A. Kirillov, R. Girshick, K. He, and P. Dollár, "Panoptic feature pyramid networks," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, 2019, pp. 6399–6408.

[26] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, 2016, pp. 3213–3223.

# Perceiving the Invisible: Proposal-Free Amodal Panoptic Segmentation

## - Supplementary Material -

Rohit Mohan and Abhinav Valada

In this supplementary material, we provide additional ablation studies on the proposed architectural components and the illustration of the context extractor module.

### S.1. Ablation Study

In this section, we first study the importance of the various components of our proposed cross-task module. Subsequently, we study the influence of the number of relative occlusion ordering layers on the performance of our network. For all the experiments, we train our PAPS network without the amodal mask refiner on the BDD100K-APS dataset and evaluate it on the validation set. We use APQ and APC metrics as the principal evaluation criteria for all the experiments performed in this section.

#### A. Evaluation of the Cross-Task Module

In this section, we evaluate our proposed architecture of the cross-task module to enable bilateral propagation of features between the task-specific decoders. For this experiment, we use the PAPS architecture without the amodal mask refiner, similar to model M6 in Sec. IV-D. Tab. S.1 presents results from this experiment. We begin with model M61 which does not use the cross-task module. In model M62, we concatenate outputs of the opposite decoder as $F_O$. For the instance decoder, $F_O = F_S$ where $F_S$ are the output features of the semantic decoder. For the semantic decoder, $F_O = F_I$ where $F_I$ are the output features of the semantic decoder. The improvement in the performance shows the utility of propagating features between the task-specific decoders. In the model M63, we define $F_O$ as the summation of the task-specific decoder features given as

$$F_O = F_I + F_S. \tag{1}$$

We observe a drop in performance for model M63 compared to both model M61 and model M62 indicating that the use of summation fails to capture complementary features and at the same time affects learning the relevant primary features of the decoders themselves. In model M64, we employ self-attention given by

$$F_R = F_I + F_S, \tag{2}$$

$$F_O = g_3(F_R) \cdot F_R, \tag{3}$$

where $g_3(\cdot)$ is the function to compute the confidence scores of $F_R$. This model achieves improved performance over both model M62 and model M63 demonstrating that the attention mechanisms are beneficial for learning complementary features. As a next step, we employ self-attention to each individual task-specific decoder features in model M65 and define $F_O$ as

$$F_O = g_1(F_I) \cdot F_I + g_2(F_S) \cdot F_S, \tag{4}$$

where $g_1(\cdot)$ and $g_2(\cdot)$ are the functions to compute the confidence scores. Model M65 achieves a score lower than Model M64 and similar to Model M62. This indicates that applying self-attention to each input of the cross-task module effectively reduces them to be similar to a summation operation. Hence, in Model M66, we employ cross-attention in $F_O$ as follows

$$F_O = (1 - g_1(F_S)) \cdot F_I + (1 - g_2(F_I)) \cdot F_S. \tag{5}$$

This model achieves a performance similar to Model M64 demonstrating that cross-attention is equally important as self-attention. Lastly, we use our proposed cross-attention followed by self-attention cross-task configuration (Eq. (1) and Eq. (2)), which yields the highest overall improvement. Consequently, from this experiment, we infer that cross-attention enables learning of adaptive complementary decoder features, whereas the following self-attention enables enhancement of these highly discriminative complementary features.

TABLE S.1: Ablation study on various configurations of our proposed cross-task head. The performance is shown for the models trained on the BDD100K-APS dataset and evaluated on the validation set. Subscripts $S$ and $T$ refer to *stuff* and *thing* classes respectively. All scores are in [%].

| Model | APQ | APC | $\text{APQ}_S$ | $\text{APQ}_T$ | $\text{APC}_S$ | $\text{APC}_T$ |
|---|---|---|---|---|---|---|
| M61 | 46.9 | 48.1 | 56.1 | 33.2 | 66.0 | 45.2 |
| M62 | 47.0 | 48.1 | 56.2 | 33.3 | 66.1 | 45.3 |
| M63 | 46.7 | 48.0 | 55.9 | 32.9 | 65.9 | 45.1 |
| M64 | 47.1 | 48.2 | 56.3 | 33.4 | 66.3 | 45.4 |
| M65 | 47.0 | 48.1 | 56.2 | 33.3 | 66.1 | 45.3 |
| M66 | 47.1 | 48.2 | 56.3 | 33.4 | 66.3 | 45.4 |
| M67 | **47.4** | **48.5** | **56.5** | **33.7** | **66.4** | **45.8** |

#### B. Detailed Study on the Relative Occlusion Ordering Layers

In this section, we study the effects of the number of relative occlusion ordering layers on the performance of our proposed architecture. Similar to Sec. S.1-A, for this experiment we use the PAPS architecture without the amodal mask refiner module. Tab. S.2 shows results from this experiment. We begin with $N = 4$ where $N$ is the number of relative occlusion ordering layers. The model achieves an improved score of $45.4\%$ and $46.6\%$ in APQ and APC, respectively compared to the baselines. This indicates that with four relative occlusion ordering layers, we can encapsulate sufficient object instances present in a given scene. Next, we use $N = 6$ and obtain a

TABLE S.2: Influence on varying the number of layers of the relative occlusion ordering layers. The performance is shown for the models trained on the BDD100K-APS dataset and evaluated on the validation set. $N$ is the number of layers, subscripts $S$ and $T$ refer to *stuff* and *thing* classes respectively. All scores are in [%].

| N | APQ | APC | $\text{APQ}_S$ | $\text{APQ}_T$ | $\text{APC}_S$ | $\text{APC}_T$ |
|---|---|---|---|---|---|---|
| 4 | 45.4 | 46.6 | 56.1 | 29.3 | 65.9 | 41.1 |
| 6 | 46.8 | 47.8 | 56.3 | 32.6 | 66.2 | 44.3 |
| 8 | **47.4** | **48.5** | **56.5** | **33.7** | **66.4** | **45.8** |
| 10 | **47.4** | **48.5** | **56.5** | **33.7** | **66.4** | **45.8** |
| 12 | **47.4** | **48.5** | **56.5** | **33.7** | **66.4** | **45.8** |

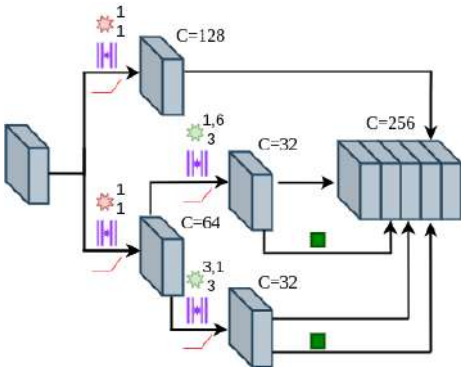



Fig. S.1: Topology of our proposed context extractor module.

significant improvement in the *thing* components of the metrics. Subsequently, we train the model with $N = 8$ which yields a lower performance in the metrics compared to $N = 6$. This indicates that $N = 6$ covers the majority of object instances in a given scene throughout the dataset. We then train the network with $N = 10$ and $N = 12$. These models do not achieve any improvement over the model with $N = 8$ layers demonstrating that with eight relative occlusion ordering layers, we can encapsulate the maximal number of object instances in the dataset.

## S.2. Context Extractor

Our proposed context extractor module enriches cross-scale features with within-scale contextual features, resulting in a rich multi-scale representation. This yields an improvement in performance for the instance decoder of our PAPS architecture as shown in Sec. IV-D-B. Fig. S.1 illustrates the architecture of the context extractor module. It splits the input into two parallel branches and employs two $1 \times 1$ convolutions. One of the branches is further subdivided into two parallel branches. Here, each branch uses a $3 \times 3$ depth-wise atrous separable convolutions with a dilation rate of $(1, 6)$ and $(3, 1)$, respectively. These branches are again subdivided into two parallel branches each. In each of these two parallel branches, one branch employs a global pooling layer. Finally, all the outputs of the remaining parallel branches are concatenated. Please note that each of the convolutions is followed by batch normalization and ReLU activation function.

## 7.2 Layer Ensembles

The appended paper [85] follows.

# LAYER ENSEMBLES

*Illia Oleksiienko and Alexandros Iosifidis*

Department of Electrical and Computer Engineering, Aarhus University, Denmark
{io,ai}@ece.au.dk

## ABSTRACT

Deep Ensembles, as a type of Bayesian Neural Networks, can be used to estimate uncertainty on the prediction of multiple neural networks by collecting votes from each network and computing the difference in those predictions. In this paper, we introduce a novel method for uncertainty estimation called Layer Ensembles that considers a set of independent categorical distributions for each layer of the network, giving many more possible samples with overlapped layers, than in the regular Deep Ensembles. We further introduce Optimized Layer Ensembles with an inference procedure that reuses common layer outputs, achieving up to 19× speed up and quadratically reducing memory usage. We also show that Layer Ensembles can be further improved by ranking samples, resulting in models that require less memory and time to run while achieving higher uncertainty quality than Deep Ensembles.

***Index Terms*—** Deep Ensembles, Bayesian neural networks, uncertainty estimation, uncertainty quality

## 1. INTRODUCTION

Uncertainty estimation in neural networks is an important task for critical problems, such as autonomous driving, medical image analysis, or other problems where silent failures of machine learning systems can lead to high-cost damages or endanger lives. Bayesian Neural Networks (BNNs) [1, 2, 3] provide a tool to estimate prediction uncertainty by exploiting a distribution over the network weights and sampling a set of models with slightly different predictions for a given input. This difference in the predictions expresses the uncertainty of the network, while the mean of all predictions is used as the prediction of the network. The selection of the adopted distribution affects the computational requirements and statistical quality of the network, with Gaussian distribution resulting in Bayes By Backpropagation (BBB) [4] and Hypermodel [5] methods, Bernoulli distribution in Monte Carlo Dropout (MCD) [6], and Categorical distribution in Deep Ensembles [7].

This work has received funding from the European Union's Horizon 2020 research and innovation programme (grant agreement No 871449 (OpenDR)).

We introduce Layer Ensembles, which consider a set of weight options for each layer that are sampled using independent Categorical distributions, resulting in a high number of models that can have common layer samples. We show that Layer Ensembles achieve better uncertainty quality than Deep Ensembles for the same number of parameters, and they allow to dynamically change the number of samples to keep the best ratio between the uncertainty quality and time cost.

## 2. RELATED WORK

Output uncertainty estimation is usually done by approximating expectation and covariance of outputs using the Monte Carlo integration with a limited number of weight samples, which can be simplified to running the network few times with different samples of random variables and then computing the mean and variance of the output vectors. Epistemic Neural Networks (ENNs) [8] propose a framework to estimate an uncertainty quality of a model by generating a synthetic dataset and training a Neural Network Gaussian Process (NNGP) [9] on it that represents a true predictive distribution. The model of interest is then evaluated by the KL-divergence [10] between the true predictive distribution from NNGP and the predictive distribution of the model of interest.

Monte Carlo Dropout (MCD) [6], instead of only using Dropout [11] layers as a form of regularization during training to avoid overtrusting particular neurons, it also keeps these Dropout layers during inference. This has the effect of adopting a Bernoulli distribution of weights and sampling different models from this distribution. Bayes By Backpropagation (BBB) [4] considers a Gaussian distribution over a network's weights, which is estimated using the reparametrization trick [12] that allows to use regular gradient computation.

Variational Neural Networks (VNNs) [13] can be considered in the same group as MCD and BBB from the Bayesian Model Averaging perspective, where sampled models can lie in the same loss-basin and be similar, i.e., describing the problem from the same point of view, as explained in [2]. VNNs consider a Gaussian distribution over each layer's predictions, that is parametrized by the outputs of the corresponding sublayers. Hypermodels [5] consider an additional hypermodel $\theta = g_\nu(z)$ to generate parameters of a base model $f_\theta(x)$ using a random variable $z \sim \mathcal{N}(0, I)$ as an input to the hypermodel.

Deep Ensembles [7] have a better uncertainty quality than all other discussed methods and can be viewed as a BNN with a Categorical distribution over weights, with the ideal number of weight samples equal to the number of ensembles. The addition of prior untrained models to Deep Ensembles, as described in [7], improves the uncertainty quality of the network. Deep Sub-Ensembles [14] split the neural network into two parts, where the first part contains only a single trunk network, and the second part is a regular Deep Ensemble network that operates on features generated by the trunk network. This reduces the memory and computational load, compared to the Deep Ensembles, and provides a trade-off between the uncertainty quality and resource requirements. Batch Ensembles [15] optimize Deep Ensembles by using all weights in a single matrix operation and using Hadamard product instead of matrix multiplication that increases inference speed and reduces memory usage.

## 3. LAYER ENSEMBLES

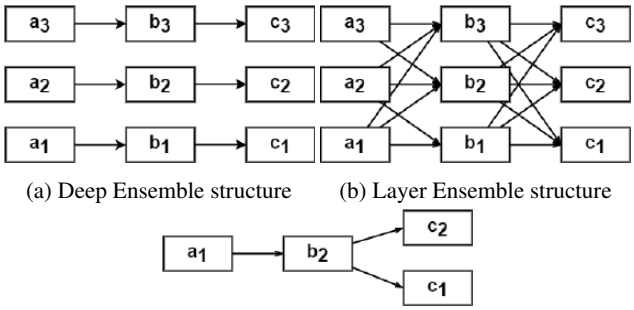
We consider a neural network $F(x, w)$ with $N$ layers that takes $x$ as input and is parametrized by the weights $w$. A Deep Ensemble network is formed by $K$ identical networks with $N$ layers each, where the corresponding weights of each network, i.e., $w_i,\ i \in [1, K]$, are trained independently. We formulate Layer Ensembles as a stochastic neural network $F(x, w)$ with $N$ layers $\mathrm{LE}_i(x, w_q^i),\ i \in [1, K],\ q \in [1, N]$ and $K$ weight options for each layer:

$$w_q^i \sim \mathrm{Categorical}(K). \tag{1}$$

This results in the same memory structure as for Deep Ensembles, with $KN$ weight sets for a network of $N$ layers and $K$ ensembles. However, Layer Ensembles allow for connections between the layers of different ensembles, by sampling different layer options to form a network in the ensemble. This greatly increases the number of possible different weight samples, while those can contain identical subnetworks. This can be used to speed up the inference of a set of sampled layers. Fig. 1 illustrates how the same memory structure of the ensembles is used in Deep Ensembles (Fig. 1a) and Layer Ensembles (Fig. 1b).

Training of Layer Ensembles is done the same way as for Deep Ensembles, by using a regular loss function and averaging over the predictions of different weight samples. The number of weight samples for Deep Ensembles is usually equal to the number of networks $K$, meaning that all the ensembles are used in the prediction process. The same strategy is not required for Layer Ensembles, as each layer option can be included in multiple networks. This means that one can select a few layer options per inference and expect that, with the sufficient amount of training steps, all the layer weights will be trained.

Following [7], we consider an output of a prior untrained Layer Ensemble network added to the output of the trained network, using the same draws from the random distributions for both untrained and trained networks. Experiments show that the addition of prior networks improves the uncertainty quality of Layer Ensembles by a factor of 2 for each number of ensembles that was tested.




(a) Deep Ensemble structure (b) Layer Ensemble structure




(c) Two samples of a Layer Ensemble network with common first two layer options

**Fig. 1**: Example structures of (a) Deep Ensembles and (b) Layer Ensembles for a 3-layer network with 3 ensembles ($N = 3$, $K = 3$). While the memory structure remains identical, Layer Ensembles have many more options for sampling that can be optimized considering the common layers in samples. Layer Ensembles with common layers earlier in the architecture lead to faster inference (c).

Layer Ensembles can be used to define Deep Ensembles [7] and Deep Sub-Ensembles [14] as special cases by sampling specific layers or number of ensembles to be used. Considering a Layer Ensemble network with $N$ layers and $K$ ensembles for each layer, we can sample $K^N$ possible models. Sampling $K$ layers with none of the layer options used in two different networks corresponds to a Deep Ensemble network. By selecting different number of ensembles per layer, we can achieve Deep Sub-Ensembles by using one ensemble for the first $T$ layers and $K$ ensembles for the remaining $N - T$ layers, resulting in a single trunk network and in an ensembled tail network. Experimenting with the number of ensembles for each layer can result in interesting new methods for specific analysis problems and is a direction for future work.

### 3.1. Inference Optimization

Layer Ensembles can reuse outputs of identical subnetworks processing the input $x$ when they are used in different networks. Fig. 1c shows an example of two Layer Ensembles where the first two layers are identical, and only the last layer has different weights. Instead of computing $c_2(b_2(a_1(x)))$ and $c_1(b_2(a_1(x)))$ independently, one can compute the common layer $V = b_2(a_1(x))$ first, and then $c_2(V)$ and $c_1(V)$.

Algorithm 1 implements an Optimized Layer Ensembles (OLE) function that recursively computes the output of a Layer Ensemble network for a set of sorted layer samples. Layer samples are represented as a set of selected options

**Algorithm 1** Optimized Layer Ensembles

**Require:** Network $F(x)$, list of sorted samples $S$, layer index $i$, input $x$

**function** OLE($F, S_i, i, x$)
  result $\leftarrow$ []
  $s_{i+1} \leftarrow$ []
  **if** $i = \text{size}(s)$ **then return** $[x]$ ▷ Final layer computed
  **end if**
  $s_l \leftarrow S_i[0]$
  $l \leftarrow F[i][s_l](x)$ ▷ First sampled option for layer $i$
  **for** $t \in [0..\text{size}(S_i)]$ **do** ▷ For each sample
    **if** $S_i[t] \neq s_l$ **then**
      result = result $\cup$ OLE($F, s_{i+1}, i+1, l$)
      $s_l \leftarrow S_i[t]$
      ▷ Next sampled option for layer $i$
      $l \leftarrow F[i][s_l](x)$
      $s_{i+1} \leftarrow$ []
    **end if**
    ▷ Update sub-samples list for input $l$
    $s_{i+1} \leftarrow s_{i+1} \cup s_i[t][1:]$
  **end for**
  result = result $\cup$ OLE($F, s_{i+1}, i+1, l$)
  **return** result
**end function**
**return** OLE($F, S, 0, x$)

for each layer, such as $[1, 2, 2]$ and $[1, 2, 1]$ for the model in Fig. 1c. These samples are sorted in ascending order by the first-most values, while using later indices in case of identical previous values. This allows to have the most overlapping samples in a sequence, giving the possibility to optimize layer executions, as a layer option should be called only once for the same input and used by all samples that share it. After the current layer option is used, there is no need to keep its output in memory anymore, as it will never be used later. The results of the OLE function is an array of outputs for all runs using this layer, which means that in order to run a full set of samples the OLE function needs to be called with network function $F$, samples list $S$, layer index $i = 0$, and the input to the network $x$.

## 4. UNCERTAINTY QUALITY EXPERIMENTS

We implement Layer Ensembles inside the Epistemic Neural Networks (ENNs) [8] framework to estimate the uncertainty quality of this method and to compare it with Deep Ensembles. ENNs consider a regression task $y = f(x) + \epsilon$ and generates a synthetic dataset $D_T = \{(x, y)_t \text{ for } t \in [0, T-1]\}$, where $x$ is a $D_x$-dimensional input vector, $y$ is an output scalar, $\epsilon \sim \mathcal{N}(0, \sigma^2)$ is a random noise, and $T = D_x\lambda$ is a dataset size. A Neural Network Gaussian Process (NNGP) [9] is trained on the dataset to represent the true model of the data. For each data point, a model of interest should provide a prediction $\mu$ and an uncertainty in that prediction $\sigma^2$, which is modeled by a one-dimensional Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$. Given the predictions from both the true NNGP model and a model of interest $M$, the uncertainty quality score $Q(M)$ is computed as:

$$
\begin{aligned}
Q(M) &= \frac{1}{T}\sum_{t=0}^{T-1} \text{KL}(\mathcal{N}_M \parallel \mathcal{N}_{\text{NNGP}}), \\
\mathcal{N}_M &= \mathcal{N}(\mathbb{E}[M(x_t)], \text{Var}[M(x_t)]), \\
\mathcal{N}_{\text{NNGP}} &= \mathcal{N}(\mathbb{E}[\text{NNGP}(x_t)], \text{Var}[\text{NNGP}(x_t)]),
\end{aligned}
\tag{2}
$$

where $M$ and NNGP are the model of interest and the true NNGP model, respectively, and KL is a Kullback–Leibler divergence function [10].

We implement Layer Ensembles inside the ENNs JAX repository [16] and follow the same experimental parameters, including $D_x \in \{10, 100, 1000\}$, $\lambda \in \{1, 10, 100\}$, and $\epsilon \in \{0.01, 0.1, 1\}$. The experiments are repeated with all combinations of $(D_x, \lambda, \epsilon)$ parameters and with 10 different random seeds. The results are then averaged, computing the mean and variance of uncertainty quality scores for each method. Fig. 4 provides a comparison between Deep Ensembles and Layer Ensembles for different number of ensembles and number of samples. Layer Ensembles start to achieve good uncertainty quality with only 3 ensembles and outperform Deep Ensembles for the same number of ensembles used. This means that the memory footprint is much lower for Layer Ensembles.

### 4.1. Layer sample ranking

Since there are many possible layer samples for Layer Ensembles, it is not always feasible to use all possible networks when using Layer Ensembles. One way to decrease the computational cost is to randomly select fewer layer samples, resulting in slightly lower quality of uncertainty, as shown in Fig. 4. Another option is to rank layer samples based on uncertainty quality on the validation set and use the best layer samples when using a particular number of samples.

Let us consider the full set of layer samples $S = \{s_j | j \in [1, J]\}$, where $J$ is a number of all combinations that can be computed by multiplying all layer-wise numbers of ensembles. To reduce the computational load of layer sample ranking, we introduce an iterative process of selecting best layer samples by starting from a single layer sample with the best mean error $s^1$:

$$
s^1 = \underset{s_j}{\text{argmax}}\ Q(M^{\{s_j\}}), \tag{3}
$$

where $Q(\cdot)$ is the uncertainty quality score function, $M^{\{s_j\}}$ is the Layer Ensemble model applied to a set of layer samples, containing a single sample $s_j$. Given a set of optimal layer samples $S^P = \{s_i^P | i \in [1..P]\}$ of size $P$, the next layer sample set is created by finding the best addition to the already

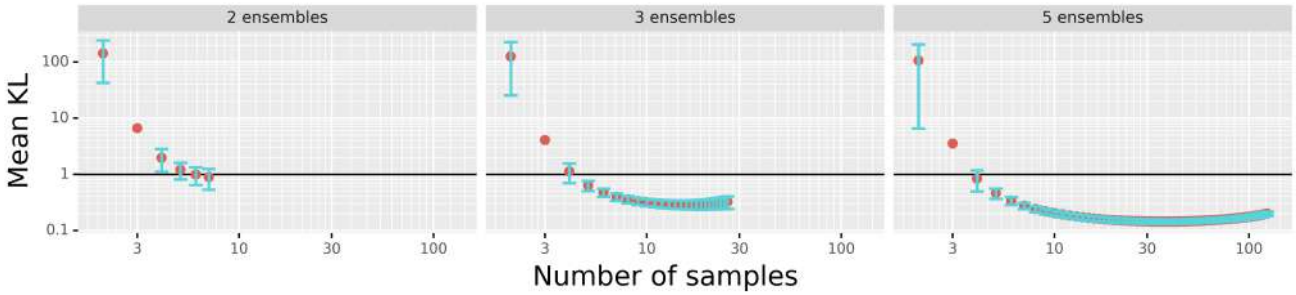



**Fig. 2**: Comparison of mean KL values with 1 STD range for Layer Ensembles with different number of ensembles and sampled layers, averaged across all experiment parameters. The best layer samples are selected based on the validation set and evaluated on the test set.

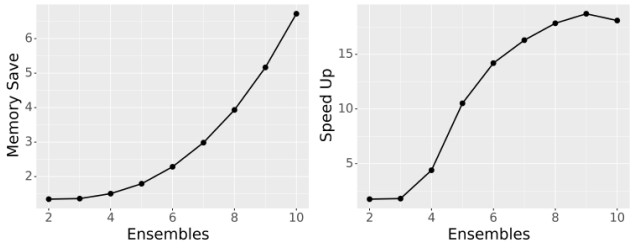



**Fig. 3**: Speed up and memory saved during inference of Optimized Layer Ensembles, compared to the regular Layer Ensembles for different number of ensembles of a 4-layer CNN for MNIST classification. This excludes memory used for the ML framework and model weights.

existing set:

$$
\begin{aligned}
s^{P+1} &= \underset{s_j}{\operatorname{argmax}}\ Q(M^{S^P \cup s_j}), \\
S^{P+1} &= S^P \cup s^{P+1}.
\end{aligned}
\tag{4}
$$

Fig. 2 illustrates the uncertainty quality results of each best layer sample set of Layer Ensembles. With a number of ensembles higher than 2, the optimal uncertainty quality is increased up to a certain number of layer samples and starts decreasing after that. This means that it is beneficial for both inference speed and uncertainty quality to not use all the available layer samples. Layer Ensembles with 5 ensembles achieves the best uncertainty quality at 36 layer samples, which is much lower than the number of 125 possible layer samples. Even at 20 layer samples, the uncertainty quality is just 6% lower, but it is still 2 times better than the Deep Ensembles with 30 ensembles, while it uses 6 times less memory, and it is at least 1.5× faster, as the speed can be improved by Optimized Layer Ensembles inference procedure but depends on the overlap between layer samples.

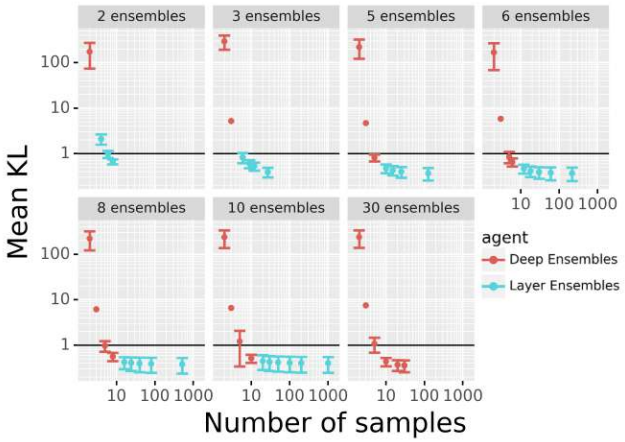



**Fig. 4**: Comparison of mean KL values with 1 STD range for Deep Ensembles and Layer Ensembles with random unique layer samples, averaged across all experiment parameters.

## 5. CONCLUSIONS

In this paper, we proposed a novel uncertainty estimation method called Layer Ensembles, which corresponds to a Bayesian Neural Network with independent Categorical distribution over weights of each layer. We showed that Layer Ensembles use parameters more effectively than Deep Ensembles and provide a flexible way to balance between inference time and model uncertainty quality. We showed that the inference of Layer Ensembles can be optimized by performing the same computations once, which increases the inference speed by up to 19 times and reduces memory usage quadratically. Finally, we proposed a layer sample ranking system that allows to select best layer samples based on the combined uncertainty quality, leading to a high increase in uncertainty quality and reducing both memory and time requirements.

## 6. REFERENCES


[1] David J C Mackay, "Probable networks and plausible predictions — a review of practical bayesian methods for supervised neural networks," *Network: Computation in Neural Systems*, vol. 6, no. 3, pp. 469–505, 1995.

[2] Andrew Gordon Wilson and Pavel Izmailov, "Bayesian Deep Learning and a Probabilistic Perspective of Generalization," in *NeurIPS*, 2020, vol. 33.

[3] Tom Charnock, Laurence Perreault-Levasseur, and François Lanusse, "Bayesian neural networks," *arXiv:2006.01490*, 2020.

[4] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra, "Weight Uncertainty in Neural Networks," in *JMLR Workshop and Conference Proceedings*, 2015, vol. 37, pp. 1613–1622.

[5] Vikranth Dwaracherla, Xiuyuan Lu, Morteza Ibrahimi, Ian Osband, Zheng Wen, and Benjamin Van Roy, "Hypermodels for exploration," in *ICLR Proceedings*, 2020, vol. 8.

[6] Yarin Gal and Zoubin Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *JMLR Workshop and Conference Proceedings*, 2016, vol. 48, pp. 1050–1059.

[7] Ian Osband, John Aslanides, and Albin Cassirer, "Randomized prior functions for deep reinforcement learning," in *NeurIPS*, 2018, vol. 31, pp. 8626–8638.

[8] Ian Osband, Zheng Wen, Mohammad Asghari, Morteza Ibrahimi, Xiyuan Lu, and Benjamin Van Roy, "Epistemic Neural Networks," *arXiv:2107.08924*, 2021.

[9] Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S. Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein, "Deep neural networks as gaussian processes," in *6th ICLR Proceedings*, 2018.

[10] S. Kullback and R. A. Leibler, "On Information and Sufficiency," *The Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79 – 86, 1951.

[11] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014.

[12] Diederik P Kingma and Max Welling, "Auto-encoding variational bayes," in *ICLR Proceedings*, 2014, vol. 2.

[13] Illia Oleksiienko, Dat Thanh Tran, and Alexandros Iosifidis, "Variational neural networks," *arxiv:2207.01524*, 2022.

[14] Matias Valdenegro-Toro, "Deep sub-ensembles for fast uncertainty estimation in image classification," *arxiv:1910.08168*, 2019.

[15] Yeming Wen, Dustin Tran, and Jimmy Ba, "Batchensemble: An alternative approach to efficient ensemble and lifelong learning," in *ICLR*, 2020, vol. 8.

[16] Ian Osband, Zheng Wen, Mohammad Asghari, Morteza Ibrahimi, Xiyuan Lu, and Benjamin Van Roy, "Github code for epistemic neural networks," 2021.

## 7.3 Variational Neural Networks

The appended paper [87] follows.

# Variational Neural Networks

Illia Oleksiienko*, Dat Thanh Tran† and Alexandros Iosifidis*, *Senior Member, IEEE*

***Abstract*—Bayesian Neural Networks (BNNs) provide a tool to estimate the uncertainty of a neural network by considering a distribution over weights and sampling different models for each input. In this paper, we propose a method for uncertainty estimation in neural networks called Variational Neural Network that, instead of considering a distribution over weights, samples outputs of each layer from a Gaussian distribution, parametrized by the predictions of mean and variance sub-layers. In uncertainty quality estimation experiments, we show that VNNs achieve better uncertainty quality than other single-bin Bayesian Model Averaging methods, such as Monte Carlo Dropout or Bayes By Backpropagation methods.**

***Index Terms*—Bayesian Neural Networks, Bayesian Deep Learning, Uncertainty Estimation**

## I. Introduction

The ability to estimate the uncertainty of prediction in neural networks provides advantages in using high-performing models in real-world problems, as it enables higher-level decision-making to consider such information in further actions. To do so, one needs the neural network to accompany its output with a measurement of its corresponding uncertainty for each input it processes. Several approaches have been introduced to this end, with Bayesian Neural Networks (BNNs) [1]–[3] providing an elegant framework for estimating uncertainty of a neural network by introducing a probability distribution over its weights and sampling different models that are meant to describe the input from different points of view. This allows to determine inputs for which the network predictions are different, leading to a measurement of the network uncertainty in its outputs. Such an approach usually comes with an increased computational cost, but may be valuable for tasks where prediction errors result in high losses.

The choice of the prior weights probability distribution function influences the statistical quality of the model and the computational resources needed to use such neural networks, which creates a possibility to explore different approaches to BNNs by using Gaussian [4], Bernoulli [5], Categorical [6] or other distributions. Sampling from the posterior distribution can be difficult, due to the complex nature of it. This leads to methods that avoid direct computation of the posterior, such as Markov Chain Monte Carlo (MCMC) [7], which constructs a Markov chain of samples $S_i$ that are distributed following the desired posterior, or Variational Inference [8], which scales better than MCMC and aims to estimate a parametrized distribution that should be close to the exact posterior. How close the distributions are is computed using the Kullback-Leibler (KL) divergence [9], but it still requires the exact posterior. This is overcome by computing an Evidence Lower Bound (ELBO) instead and optimizing it with Stochastic Variational Inference (SVI) [10].

We introduce Variational Neural Networks (VNNs) which do not consider a distribution over weights, but define sub-layers to generate parameters for the output distribution of the layer. To keep computational and memory resource usage practical, we consider a Gaussian distribution with learnable mean and variance. This is achieved by using two instances of the same regular layer like convolutional or linear with different weights, and using their predictions from the inputs as means and variances of the Gaussian distribution over the outputs. We provide a neural network formulation that describes both related BNNs and the proposed VNNs in a unified manner, and show that VNNs, while being in the same group, as Monte Carlo Dropout (MCD) [5] and Bayes By Backprop (BBB) [4] from the Bayesian Model Averaging (BMA) perspective [2], achieve better uncertainty quality and retain it with the increasing data dimensionality.

## II. Bayesian Neural Networks

BNNs [1]–[3] consider a distribution over their weights $p(w|a)$ and a distribution over their hyperparameters $p(a)$. A predictive distribution over an output $y$ for a data point $x$ can be obtained by integrating over all possible hyperparameters and model weights, i.e.:

$$p(y|x) = \int\int p(y|x,w)\, p(w|a)\, p(a)\, da\, dw. \tag{1}$$

Given a dataset $D = (X_t, Y_t)$, where $X_t$ and $Y_t$ are the sets of inputs $\{x\}$ and targets $\{y\}$, the distribution of weights can be derived from Bayes' theorem as $p(w|D,a) = \frac{p(Y_t|X_t,w)p(w|a)}{p(D)}$, and the corresponding predictive distribution has a form

$$p(y|x,D) = \int\int p(y|x,w)\, p(w|D,a)\, p(a)\, da\, dw. \tag{2}$$

Classical neural networks can be viewed as BNNs with $p(a) = \delta(a - \hat{a})$ and $p(w|D,a) = \delta(w - \hat{w}_a)$ [3], where $\hat{a}$ are the selected model hyperparameters, $\hat{w_a}$ are the weights, optimized by training the model, and $\delta(x)$ is the Dirac delta function which has values 0 everywhere except at $x = 0$ where it equals to 1. In this case, the predictive distribution becomes

$$\begin{aligned} p(y|x,D) &= \int\int p(y|x,w)\, p(w|D,a)\, p(a)\, da\, dw \\ &= \int\int p(y|x,w)\, \delta(w - \hat{w}_a)\, \delta(a - \hat{a})\, da\, dw \\ &= p(y|x,\hat{w}_{\hat{a}}), \end{aligned} \tag{3}$$

which is a distribution dictated by a loss of the network.

This work has received funding from the European Union's Horizon 2020 research and innovation programme (grant agreement No 871449 (OpenDR)).

I. Oleksiienko and A. Iosifidis are with the Department of Electrical and Computer Engineering, Aarhus University, Denmark (e-mail: {io,ai}@ece.au.dk). D. T. Tran is with the Department of Computing Sciences, Tampere University, Finland (e-mail: thanh.tran@tuni.fi).

When training classical neural networks, hyperparameters are considered fixed at point $\hat{a}$ and weights are optimized either by maximum likelihood estimation (MLE), i.e.:

$$\begin{aligned} w_{\text{mle}} &= \underset{w}{\text{argmax}}\Big[\log\, p(D|w,\hat{a})\Big] \\ &= \underset{w}{\text{argmax}}\left[\sum_i \log\, p(Y_i|X_i,w,\hat{a})\right], \end{aligned} \tag{4}$$

or by maximum a posteriori (MAP), i.e.:

$$\begin{aligned} w_{\text{map}} &= \underset{w}{\text{argmax}}\Big[\log\, p(w|D,\hat{a})\Big] \\ &= \underset{w}{\text{argmax}}\Big[\log\, p(D|w,\hat{a}) + \log\, p(w)\Big], \end{aligned} \tag{5}$$

where $\log\, p(w)$ is a regularization term.

Due to the complexity of neural networks, direct computation of $w_{\text{mle}}$ or $w_{\text{map}}$ cannot be achieved, and therefore approximate methods are used to find these values. The most popular process to estimate the weight values is the Backpropagation algorithm [11], where an initial randomly selected $w$ is updated following the direction of negative gradient of the loss function with respect to $w$.

## III. Related Works

The use of BNNs in real-world applications is limited due to the complex nature of the possible prior and predictive distributions. Therefore, simplified versions are used. Assumptions that are proposed in different methods below aim to reduce memory, inference and training time, but they come with the cost of reducing the statistical quality of the resulting models. This problem is further discussed in Section V.

MCD [5] considers a neural network with Dropout [12] added to each layer. The Dropout layer effectively turns off random neurons of the layer by multiplying connection weights with a random binary mask, sampled from a Bernoulli distribution. This allows to avoid overfitting specific neurons. After training, Dropout is replaced by an identity function and all neurons are used for inference. Instead of replacing Dropout with identity, MCD uses it during inference leading to a stochastic model. The model uncertainty for an input is computed by performing inference multiple times and computing mean and variance of predictions. BBB [4] samples model parameters from a Gaussian distribution and trains it using regular Backpropagation. By doing so, the family of models with different weights is sampled from the learned distribution, and the uncertainty of the network is computed as the variation in predictions of different samples.

Ensembles of neural networks [6] can also be used for uncertainty estimation. Ensembles are trained in parallel for the same task, but with different random seeds, which results in different weight initialization and training order. Outputs from members of an ensemble will vary, and this can be used to improve performance by taking an average of their predictions, or to estimate uncertainty by computing the variance of their outputs. Such an approach can be viewed as a BNN with a categorical distribution over weights that randomly selects one of the trained model weights. Hypermodels [13] use an additional model $\theta = g_\nu(\mathbf{z})$ to generate parameters for a base model $f_\theta(x)$. Linear Hypermodels set $g_\nu(\mathbf{z}) = a + B\mathbf{z}$, $\mathbf{z} \sim \mathcal{N}(0, I)$. Using different samples of $\mathbf{z}$, one can sample different model parameters and estimate uncertainty in the same way as for the aforementioned methods.

## IV. Variational Neural Networks

As introduced in Section II, a neural network is described by its weights $w$ and hyperparameters $\alpha$. Hyperparameters include the structure of the network, i.e., the type and number of layers, their size and connections, etc. Usually, we limit the hyperparameters by defining some of them in the beginning, e.g., by selecting that we want to use a Convolutional Neural Network (CNN). This is a reasonable approach, as it is impractical to iterate through all possible types and structures of networks during training. We are using the neural network formulation $\text{NN}(x) := F^\Lambda(x, w)$, where $\text{NN}(x)$ is a neural network applied to an input $x$, $w$ are the trained weights, $F$ is a neural network function that incorporates structure and other hyperparameters inside it, and $\Lambda$ is a set of layer implementations, which are used by $F$ to process layer inputs.

In case of CNNs, $\Lambda = \{\text{Conv2D}(x, w), \text{FC}(x, w)\}$ results in a regular CNN, where $\text{Conv2D}(x, w)$ is a 2D convolutional layer function and $\text{FC}(x, w)$ is a fully connected layer function. If we select $\Lambda = \{\text{Conv2D}(x, w_c \sim \mathcal{N}(\mu_c, \Sigma_c)), \text{FC}(x, w_l \sim \mathcal{N}(\mu_l, \Sigma_l))\}$ with layer weights sampled from a corresponding Gaussian distribution, then the resulting network is a BBB CNN. Such neural network formulation allows to accurately describe all the discussed uncertainty estimation methods, as well as the proposed VNNs.

### A. Variational Layer

We define a Variational Layer (VL) that takes an input $x$ and weights $w$ as

$$\begin{aligned} &\text{VL}(x, w) = \alpha_\mathcal{N}(f(x, w)), \\ &f(x, w) \sim \mathcal{N}\Big(\alpha_\mu(L(x, \mu)), diag[(\alpha_\sigma(L(x, \sigma)))^2]\Big), \\ &w = (\mu, \sigma), \end{aligned} \tag{6}$$

where $L(x, w)$ is a regular neural network layer, such as fully connected, convolutional or a recurrent layer. $L(x, \mu)$ and $L(x, \sigma)$ represent instances of the same layer with different values of parameters and corresponding activation functions $\alpha_\mu(\cdot)$ and $\alpha_\sigma(\cdot)$. The activation function $\alpha_\mathcal{N}(\cdot)$ can be used to apply nonlinearity to the randomly sampled values $f(x, w)$. By selecting which of $\alpha_\mu(\cdot)$, $\alpha_\sigma(\cdot)$, $\alpha_\mathcal{N}(\cdot)$ are set to identity and which are set to actual activation functions (such as the Rectified Linear Unit (ReLU)) one can create networks that are described by different mathematical models. In the following, we show how different selections can lead to specific types of uncertainties, i.e., epistemic and aleatoric uncertainties [14].

Training of VNNs is performed by averaging outputs from different network passes for the same sample and, thanks to the reparametrization trick [15], the regular Backpropagation algorithm is applied. Networks can also be trained with a single pass, which results in the same training procedure as for classical neural networks. The models are trained with the usual loss functions that are suitable for the task.

### B. Output uncertainty estimation

Estimation of prediction uncertainties in VNNs and BNNs can be done following the same formulation, but it obtained from different characteristics of these methods. Below, we first describe how BNNs can be reformulated by splitting the parametrized distribution over weights into isolated parameters and a non-parametric distribution, and then show that this formulation can be applied to VNNs.

Following [16], we consider a neural network $F(x, w)$ with a parametric distribution over weights $q_m(w)$. We assume the choice of $q_m(w)$ in a form

$$w = Q(m, z), \qquad w \sim q_m(w), \qquad z \sim p(z), \tag{7}$$

where $p(z)$ is a non-parametric distribution and $Q(\cdot)$ applies a deterministic transformation, parametrized by $m$, to a non-parametric random variable $z$. Such formulation is suitable for every uncertainty estimation method described in Section III. For BBB models, $Q(\cdot)$ is defined as

$$Q(m, z) = \mu + \sigma^2 z, \qquad z \sim \mathcal{N}(0, I), \qquad m = (\mu, \sigma), \tag{8}$$

where we break down a parametric Gaussian distribution $\mathcal{N}(\mu, \sigma I)$ into two parts: a parametric deterministic transformation $z \longrightarrow \mu + \sigma^2 z$ and a non-parametric random variable $z \sim \mathcal{N}(0, I)$.

Defining an epistemic index $z \sim p(z)$ [16], we can formulate a deterministic neural network $F_d(\cdot)$ function that takes a draw of a random non-parametric variable $z$, instead of using $F(\cdot)$ with a complex distribution over $w$:

$$F_d(x, m, z) := F(x, w), \qquad w = Q(m, z), \qquad z \sim p(z) \tag{9}$$

With this formulation, a predictive distribution (2) for fixed hyperparameters is defined by splitting $w$ into $m$ and $z$ as follows [2], [16]:

$$\begin{aligned} p(y|x, D) &= \int p(y|x, w)\, q_m(w|D)\, dw = \int p(y|x, m, z)\, p(z)\, dz, \\ \mathbb{E}[y] &= \int y\, p(y|x, D) dy \approx \frac{1}{T} \sum_i^T F_d(x, m, z_i), \\ \mathrm{Cov}[y] &= \int (y - E[y])(y - E[y])^T p(y|x, D)\, dy, \\ &\approx \frac{1}{T} \sum_i^T (E[y] - F_d(x, m, z_i))(E[y] - F_d(x, m, z_i))^T, \end{aligned} \tag{10}$$

where expectation and variance are computed using Monte Carlo integration, which can be viewed as an approximation of $p(z)$ with $\sum_{i=0}^{T} \frac{\delta(z - z_i)}{T}$, $z_i \sim p(z)$, $i \in 1, \ldots, T$ [2]. Variance of the outputs is computed by taking main diagonal values of the $\mathrm{Cov}[y]$ representing the uncertainty of the model.

VNNs, despite not having a direct distribution over weights, can also be formulated as a deterministic function $F_d(x, w, z)$ with a variational index $z \sim p(z)$. This is done by describing the output Gaussian distribution $\mathcal{N}\Big(\alpha_\mu(L(x, \mu)), diag[(\alpha_\sigma(L(x, \sigma)))^2]\Big)$ of a VL as a linear transformation of a unit Gaussian $\alpha_\mu(L(x, \mu)) + diag[(\alpha_\sigma(L(x, \sigma)))^2]\mathcal{N}(0, I)$.

### C. Epistemic uncertainty

Epistemic uncertainty describes the lack of knowledge of the model and can be improved by providing a better model structure, better dataset or improved training procedure, while aleatoric uncertainty describes the uncertainty in data due to noise in data perceiving process or domain shift [17], [18]. Usually, the epistemic uncertainty in BNNs is modeled by assuming a distribution over weights and fixed hyperparameters. The use of unfixed hyperparameters leads to the Hierarchical Bayes approach [19], where the epistemic uncertainty is represented by both hyperparameters and weight distributions.

Given the fact that model parameters and structure are usually separated, the predictive distribution equation (2) holds only in the case where the hyperparameters' influence is limited to the training procedure. If the model structure is included in hyperparameters, then:

$$p(y|x, D) = \int \int p(y|x, w, a)\, p(w|D, a)\, p(a)\, da\, dw, \tag{11}$$

where the probability of a prediction for a selected model, depends on both weights and hyperparameters. Following this approach, the predictive distribution of VNN (10) can be interpreted as a predictive distribution, computed for a Hierarchical BNN with a unit Gaussian distribution over hyperparameters $z$ and a Dirac delta distribution over weights:

$$\begin{aligned} p(y|x, D) &= \int \int p(y|x, w, z)\, p(w|D, z)\, p(z)\, dz\, dw \\ &= \int \int p(y|x, w, z)\, \delta(w - \hat{w})\, p(z)\, dz\, dw \\ &= \int p(y|x, \hat{w}, z)\, p(z)\, dz. \end{aligned} \tag{12}$$

This formulation shows that the use of the variational index $z$ models the epistemic uncertainty in VNNs.

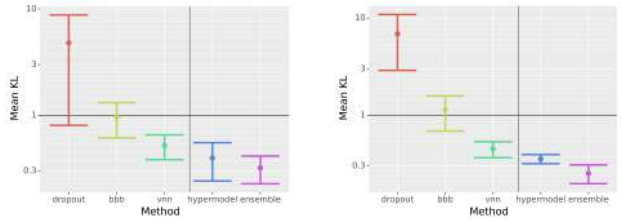
### D. Aleatoric uncertainty

Fully connected and convolutional layers can be described as the operation $L(x, \lambda) = W_\lambda x + b_\lambda$, where $\lambda = (W_\lambda, b_\lambda)$, $W_\lambda$ and $b_\lambda$ are weights and biases of the layer, and a corresponding activation function $\alpha_\lambda(\cdot)$ can be applied to the output of $L(x, \lambda)$.

Consider a Variational Layer with $L(x, \sigma) = \Sigma$, $\Sigma \in \mathbb{R}$, which can be directly achieved by setting $W_\sigma = 0, b_\sigma = \Sigma$ and setting the corresponding activation function $\alpha_\sigma(\cdot)$ to identity. Applying the formulation of fully connected and convolutional layers to $f(x, w)$ (6) and using the reparametrization trick [15], we can reformulate it as follows:

$$\begin{aligned} \epsilon &\sim \mathcal{N}(0, I), \\ f(x, w) &= \alpha_\mu(W_\mu x + b_\mu) + \Sigma\epsilon = L(x, \mu) + \epsilon_\sigma, \\ \epsilon_\sigma &\sim \mathcal{N}(0, \Sigma I). \end{aligned} \tag{13}$$

In this formulation, $\epsilon_\sigma$ models the aleatoric uncertainty [17] for the next subnetwork, which takes outputs of the current layer as inputs and cannot improve this uncertainty by improving the model.




(a) All parameters (b) $D_x = 1000$

Fig. 1: Comparison of mean KL value with 1 STD range for each method averaged across different experiment parameters.

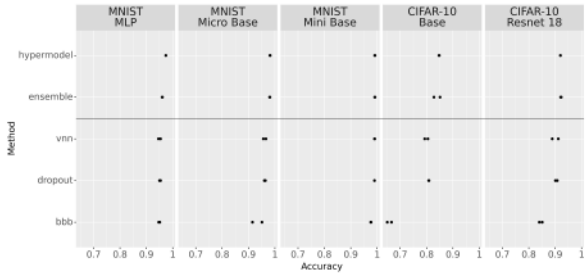



Fig. 2: Comparison of classification accuracy on MNIST and CIFAR-10 datasets with different model architectures.

## V. Experiments

A recently proposed framework called Epistemic Neural Networks [16] aims to provide a possibility to rank BNNs based on their ability to accurately estimate output uncertainty. This is done by first generating a synthetic dataset $D_T = \{(x, y)_t \text{ for } t \in [0, T-1]\}$ for a simple regression task $y = f(x) + \epsilon$, where $y$ is an output scalar, $x$ is an input data point with $D_x$ number of dimensions, $\epsilon$ is a random variable sampled from a Gaussian distribution $\mathcal{N}(0, \sigma^2)$ representing an aleatoric uncertainty. The dataset size $T$ is determined as $T = D_x\lambda$, where $\lambda$ is a hyperparameter, meaning that more data points are created for a higher dimensionality of $x$. The dataset is used to train a Neural Network Gaussian Process (NNGP) [20] and an uncertainty estimation model of interest. NNGP serves as an ideal probabilistic model for this data, and a predictive distribution of a selected uncertainty estimation model should be as close as possible to the predictive distribution of the NNGP model. The above process is used to create two datasets, one used for training the uncertainty estimation model and one (test set) used to evaluate the uncertainty estimation performance. Following [16], random noise is added to the data belonging to the training set, as it has been shown to increase the uncertainty estimation performance, which is measured by computing the KL-divergence between the true posterior $\mathcal{N}(\mu_{\text{GP}}, k_{\text{GP}})$ and a model predictive distribution $\mathcal{N}(\mu_B, k_B)$. Lower values of KL-divergence represent better uncertainty quality for a selected model, and therefore can be used to rank different approaches for uncertainty estimation.

We implement VNNs inside the ENN's JAX implementation [21] to reproduce results for BBB [4], MCD [5], Ensemble [6], Hypermodel [13] and compare them with VNN. The code is available at [22]. We follow the original framework parameters and repeat experiments with the following options: $D_x \in \{10, 100, 1000\}$, $\lambda \in \{1, 10, 100\}$, and $\epsilon \in \{0.01, 0.1, 1\}$. Each model is trained with 10 different random seeds, and the resulting KL value is the average of individual runs. The average KL values for all experiment parameters are given in Fig. 1a and for the highest input dimension value $D_x = 1000$ are given in Fig. 1b. VNN has better uncertainty quality than BBB and MCD, but it is outperformed by Hypermodel and Ensemble. This can be explained by the difference in BMA for Deep Ensembles and Variation Inference methods, as explained in [2]. The weight probability distribution can be split into basins, where models sampled from the same basin are too similar and will describe the problem from the same point of view, resulting in multiple entries of actually identical model in the prediction voting. Deep Ensembles and Hypermodels avoid this problem by not having a single anchor point with small weight deviations, and therefore having high chances of converging trained models into different basins. This means that VNN has a higher chance than Ensemble to have its samples in a single basin, placing it in the same group as BBB and MCD. Additionally, with bigger data dimensionality $D_x$, MCD and BBB achieve worse results, while VNN, Ensemble and Hypermodel perform better.

We further perform experiments on image classification. We train the same methods for image classification tasks on MNIST [23] and CIFAR-10 [24] datasets. To show the influence of model architecture on the performance, we use a set of architectures $\{F_i\}$ and train each method with the selected architecture $F$. We select a Base architecture to have 3 convolutional and 1 linear layer for MNIST, and 6 convolutional and 1 linear layer for CIFAR-10. Mini and Micro Base architectures have the same layer structure as the Base one, but a lower number of channels in each layer. MLP architecture consists of 3 fully connected layers. We also use Resnet-18 [25] architecture for experiments on CIFAR-10. For each method, we train models with different hyperparameter values and select the best two models for comparison. The results of classification experiments are given in Fig. 2 and are roughly following the results of uncertainty quality estimation experiments.

## VI. Conclusion

We proposed Variational Neural Networks that consider a Gaussian distribution over outputs of each layer, the mean and variance of which are generated by the corresponding sub-layers, and evaluated their uncertainty estimation quality within the Epistemic Neural Networks framework. Experiments show that, despite having similar properties of Bayesian Model Averaging to Monte Carlo Dropout and Bayes By Backpropagation, where sampled models are close resulting in similar models' points of view, VNNs achieve better uncertainty quality which is retained when data dimensionality is increased, in contrast to Monte Carlo Dropout and Bayes By Backpropagation methods.

## Acknowledgment

We thank Dr. Martin Magris for helpful discussions and feedback.

## REFERENCES


[1] D. J. C. Mackay, "Probable networks and plausible predictions — a review of practical bayesian methods for supervised neural networks," *Network: Computation in Neural Systems*, vol. 6, no. 3, pp. 469–505, 1995.

[2] A. G. Wilson and P. Izmailov, "Bayesian Deep Learning and a Probabilistic Perspective of Generalization," *arXiv:2002.08791*, 2020.

[3] T. Charnock, L. Perreault-Levasseur, and F. Lanusse, "Bayesian neural networks," *arXiv:2006.01490*, 2020.

[4] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight Uncertainty in Neural Networks," *arXiv:1505.05424*, 2015.

[5] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," *arxiv:1506.02142*, 2016.

[6] I. Osband, J. Aslanides, and A. Cassirer, "Randomized prior functions for deep reinforcement learning," *arXiv:1806.03335*, 2018.

[7] W. K. Hastings, "Monte Carlo sampling methods using Markov chains and their applications," *Biometrika*, vol. 57, no. 1, pp. 97–109, 1970.

[8] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe, "Variational inference: A review for statisticians," *Journal of the American Statistical Association*, vol. 112, no. 518, pp. 859–877, 2017.

[9] S. Kullback and R. A. Leibler, "On Information and Sufficiency," *The Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79 – 86, 1951.

[10] M. Hoffman, D. M. Blei, C. Wang, and J. Paisley, "Stochastic variational inference," *arxiv:1206.7051*, 2012.

[11] H. J. Kelley, "Gradient theory of optimal flight paths," *Ars Journal*, vol. 30, no. 10, pp. 947–954, 1960.

[12] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014.

[13] V. Dwaracherla, X. Lu, M. Ibrahimi, I. Osband, Z. Wen, and B. V. Roy, "Hypermodels for exploration," *arXiv:2006.07464*, 2020.

[14] A. Kendall and Y. Gal, "What uncertainties do we need in bayesian deep learning for computer vision?" in *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc., 2017.

[15] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv:1312.6114*, 2014.

[16] I. Osband, Z. Wen, M. Asghari, M. Ibrahimi, X. Lu, and B. V. Roy, "Epistemic Neural Networks," *arXiv:2107.08924*, 2021.

[17] E. Hüllermeier and W. Waegeman, "Aleatoric and epistemic uncertainty in machine learning: an introduction to concepts and methods," *Mach. Learn.*, vol. 110, no. 3, pp. 457–506, 2021.

[18] J. Gawlikowski, C. R. N. Tassi, M. Ali, J. Lee, M. Humt, J. Feng, A. M. Kruspe, R. Triebel, P. Jung, R. Roscher, M. Shahzad, W. Yang, R. Bamler, and X. X. Zhu, "A survey of uncertainty in deep neural networks," *arxiv:2107.03342*, 2021.

[19] G. M. Allenby and P. E. Rossi, "Hierarchical bayes models," *The handbook of marketing research: Uses, misuses, and future advances*, pp. 418–440, 2006.

[20] J. Lee, Y. Bahri, R. Novak, S. S. Schoenholz, J. Pennington, and J. Sohl-Dickstein, "Deep neural networks as gaussian processes," *arXiv:1711.00165*, 2018.

[21] I. Osband, Z. Wen, M. Asghari, M. Ibrahimi, X. Lu, and B. V. Roy, "Github code for epistemic neural networks," 2021.

[22] I. Oleksiienko, D. T. Tran, and A. Iosifidis, "Variational neural networks implementation in pytorch and jax," *Software Impacts*, vol. 14, p. 100431, 2022.

[23] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[24] A. Krizhevsky, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.

[25] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *arXiv:1512.03385*, 2015.

## 7.4 VPIT: Real-time Embedded Single Object 3D Tracking Using Voxel Pseudo Images

The appended paper[86] follows.

# VPIT: Real-time Embedded Single Object 3D Tracking Using Voxel Pseudo Images

Illia Oleksiienko*, Paraskevi Nousi†, Nikolaos Passalis†, Anastasios Tefas† and Alexandros Iosifidis*
*Department of Electrical and Computer Engineering, Aarhus University, Denmark
†Department of Informatics, Aristotle University of Thessaloniki, Greece
{io, ai}@ece.au.dk {paranous, passalis, tefas}@csd.auth.gr

***Abstract*—In this paper, we propose a novel voxel-based 3D single object tracking (3D SOT) method called Voxel Pseudo Image Tracking (VPIT). VPIT is the first method that uses voxel pseudo images for 3D SOT. The input point cloud is structured by pillar-based voxelization, and the resulting pseudo image is used as an input to a 2D-like Siamese SOT method. The pseudo image is created in the Bird's-eye View (BEV) coordinates, and therefore the objects in it have constant size. Thus, only the object rotation can change in the new coordinate system and not the object scale. For this reason, we replace multi-scale search with a multi-rotation search, where differently rotated search regions are compared against a single target representation to predict both position and rotation of the object. Experiments on KITTI Tracking dataset show that VPIT is the fastest 3D SOT method and maintains competitive Success and Precision values. Application of a SOT method in a real-world scenario meets with limitations such as lower computational capabilities of embedded devices and a latency-unforgiving environment, where the method is forced to skip certain data frames if the inference speed is not high enough. We implement a real-time evaluation protocol and show that other methods lose most of their performance on embedded devices, while VPIT maintains its ability to track the object.**

***Index Terms*—3D tracking, single object tracking, voxels, pillars, pseudo images, real-time neural networks, embedded deep learning (DL)**

## I. Introduction

With the rise of robotics usage in real-world scenarios, there is a need to develop methods for understanding the 3D world in order to allow a robot to interact with objects. To this end, efficient methods for 3D object detection, tracking, and active perception are needed. Such methods provide the main source of information for scene understanding, as having high-quality object detection and tracking outputs increases chances for a successful interaction with surroundings.

While for 2D perception tasks mainly cameras are used, there is a variety of sensors that can be used for 3D perception tasks, ranging from inexpensive solutions like single or multiple cameras to more costly ones like Lidar or Radar. Lidar is currently a well-adopted choice for 3D perception methods as it creates a set of 3D points forming a point cloud taken by shooting laser beams in multiple directions and counting the time needed for a reflection to be sensed. Point cloud data, despite being sparse and irregular, contains much more information needed for 3D perception compared to images, and is more robust to changes in weather and lightning conditions. 3D object detection and tracking methods using point clouds provide the best combination of accuracy and inference speed, as can be seen in the KITTI leaderboard [1].

Lidars usually operate at 10-20 FPS, and therefore, a method receiving point cloud data as input can be called real-time if its inference speed is at the rate of the Lidar's data generation, as it will not be able to receive more data to process. Even when the method cannot benefit from the FPS, higher than the Lidar's frame rate, perception methods are usually paired with another method that makes use of the perception results, as in planning tasks. This means that the saved processing time in between Lidar's frames can be used to analyze the results of the tracking without affecting its performance. In case of insufficient total frame rate, that can happen due to a slower computing system and higher additional computational load, tracking methods can suffer from dropped frames, that cannot be processed because the system is busy processing a previous data frame. This will result in worse tracking results and, in case of high drop rate, it may make the method unusable.

3D SOT, in contrast to the multiple 3D object tracking (3D MOT) task, focuses on tracking a single object of interest with a given initial frame position. This task lies between object detection and multiple object tracking tasks, as the latter one requires objects to be detected first, and then to associate each of them to a previously observed object. SOT methods do not rely on object detection, but they try to find the object offset on a new frame either by using correlation filters [2], [3], or deep learning models that regress to the predicted object offset [4], or use a Siamese approach to find the position with the highest similarity [5]–[8], or use voting [9], [10].

Application of tracking methods for real-world tasks meets a problem where, due to the model's latency, not all data frames can be processed [11], [12]. For single-frame tasks, such as object detection, this problem only influences the latency of the predictions and not their quality, but in tracking, the connection between consecutive frames is important. This means that dropping frames can result in wrong associations for MOT or in a lost object for SOT. Methods like those mentioned above do not take into consideration the limited computational capabilities of embedded computing devices, which are typically used in robotics applications, e.g., self-driving cars, due to a lower power consumption that allows the robotic system to stay active for a longer time. These devices

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871449 (OpenDR).

have a different architecture of CPU-GPU communication than desktop computers and high-end workstations. Thus, depending on the CPU-GPU workload balance and communication protocol of a robotic system, some methods can be better suited for achieving real-time operation.

In this paper, we present a novel method for 3D SOT receiving voxel pseudo images as an input, as opposed to point-based models proposed for 3D SOT. Voxel pseudo images are created in Bird's-eye View (BEV) coordinates, and since the objects in such projection do not change size, we remove the standard multiscale search process followed by 2D SOT methods. We propose the use of multi-rotation search, where differently rotated search regions are compared against a single target representation to predict both the position and rotation of the object. We show that adopting such an approach leads to fast tracking with competitive performance compared to the point-based methods. This opens a possibility to adapt 2D SOT methods for a 3D task with minor changes. Considering real-world limitations, we follow [11] and implement a benchmark for real-time Lidar-based 3D SOT on embedded devices, showcasing the performance drop of the fastest methods when implemented in a real-world scenario. We compare VPIT using the real-time benchmark with the fastest methods P2P [9] and PTT [10] and show that those methods lose most of their performance on embedded devices, which indicates that other methods that are $2\times$ or more slower than these will not be able to track at all.

The remainder of the paper is organized as follows: Section II provides a description of the related works. Section III describes the proposed method, along with the proposed training and inference processes. Section IV provides evaluation results on high-end and embedded devices with consideration of real-time requirements. Section V concludes the paper and formulates directions for future work.

## II. Related Works

SOT in 3D is commonly formulated as an extension of 2D SOT. In both cases, an initial position of the object of interest is given, and the method needs to predict the position of the object in all future frames. The main difference between 2D and 3D SOT rises from the type of data used as input. Camera-based 3D perception methods commonly achieve poor performance due to the increased difficulty of extracting correct spatial information from cameras, and therefore most of the 3D object tracking methods use point cloud data or a combination of point clouds and camera images. Point cloud sparsity does not allow using straightforward extensions of 2D SOT methods based on regular CNNs.

SC3D [13] uses a Siamese approach by encoding a target point cloud shape in the initial frame and searching for regions in a new frame with the smallest cosine distance between target and search encodings. After finding a new location, the new object shape is combined with the previous one to increase the quality of comparison in future frames. It achieves good tracking results, but its operation is computationally expensive. Assuming that an object should not move far away from its current location in consecutive frames, one can define a search region inside its neighborhood for searching it in a new frame. This is commonly done by expanding the region around the position of the target.

P2B [9] uses a point-wise network to process points in target and search regions to create a similarity map and find potential target centers. These regions are then used by a voting algorithm to find the best position candidate.

BAT [14] is a 3D tracking method based on P2B, which uses Box Cloud representations as point features, i.e., a representation which depicts the distances between the points of an object and the center and corners of its 3D bounding box.

3D-SiamRPN [5] uses a Siamese point-wise network to create features for target and search point clouds. It then uses a cross-correlation algorithm to find points of the target in a new frame. An additional region proposal subnetwork is used to regress the final bounding box.

The F-siamese tracker [15] aims to fuse RGB and point cloud information for 3D tracking and applies a 2D Siamese model to generate 2D proposals from an RGB image, which are then used for 3D frustum generation. The proposed frustums are processed with a 3D Siamese model to get the 3D object position.

Point-Track-Transformer (PTT) [10], [16] creates a transformer module for point-based SOT methods and employs it based on a P2B model, leading to an increased performance.

3D Siam-2D [17] uses two Siamese networks, one that creates fast 2D proposals in BEV space, and another one that uses projected 2D proposals to identify which of the proposals belong to the object of interest. The loss of information that occurs in BEV projection affects the performance of the 2D Siamese network. Although voxel pseudo images lie on the BEV space, they do not have this problem, as each pixel of this image incorporates information about the points in a corresponding voxel via a small neural network, and not the projection alone.

Recent works on 3D SOT focus on improving speed as well as tracking performance, but most do not take into consideration the limited computational capabilities of embedded devices, like the NVIDIA Jetson series, which are commonly used in robotics applications. Large delays in processing incoming frames can lead to dropped frames, which can quickly lead to performance degradation of traditional trackers as larger and larger offsets must be predicted as time progresses. In 2D tracking, this concept has been investigated, for example in the real-time experiments of the VOT benchmarks [18], and more recently in [11] for streaming perception tasks in general. In this work, we extend this notion to 3D SOT, and design VPIT to be used efficiently and effectively on embedded devices. Furthermore, to the best of our knowledge, VPIT is the first tracker to use a siamese architecture on point cloud pseudo images directly, while using varying rotations of the target area to find the target's rotation in subsequent frames. VPIT can be seen as a 3D extension of 2D siamese-based trackers [6], [19], paving the way for other 2D approaches to be successfully extended to the 3D case, while maintaining high tracking speed, even on embedded devices.

## III. Proposed method

Our proposed method is based on a modified PointPillars architecture for 3D single object tracking, which is originally trained for 3D object detection. In the following subsections, we first introduce our proposed modifications to the baseline architecture to shift from detection to tracking, and then describe the training and inference processes. Implementation of our method is publicly available in the OpenDR toolkit[1].

### A. Model architecture

Point cloud data is irregular and cannot be processed with 2D convolutional neural networks directly. This forces methods to either structure the data using techniques such as voxelization [20]–[23], or to use neural networks that work well on unordered data, such as MLPs with maximum pooling operations or Transformers [5], [9], [10], [24], [25].

Existing datasets for 3D object detection and tracking, such as KITTI [26] and NuScenes [27], consider only a single rotation angle: around the vertical axis. This limitation raises from the nature of scenes in these datasets, as they present data for outdoor object detection and tracking with objects moving on roads. These objects are mostly rotated in Bird's-eye View (BEV) space, and therefore only this rotation is considered for simplicity.

Since the rotation is limited to the BEV space, we can perform 2D-like Siamese tracking on a structured point cloud representation, such as PointPillars pseudo image [21]. This image is the result of voxelization, where the vertical size of each voxel takes up the whole vertical space, and therefore creating a 2D map in BEV space where each pixel represents a small subspace of a scene and points inside it. Objects in such an image have constant size, as they are not affected by projection size distortions, but their rotation makes the Axis-Aligned Bounding Boxes (AABB) tracking impractical.

Siamese models use an identical transformation $\theta(\cdot)$ to both inputs $x$ and $z$, which are then combined by some function $g(\cdot)$, i.e. $f(x,z) = g(\theta(x), \theta(z))$. Siamese tracking methods select $\theta(\cdot)$ to be an embedding function and $g(\cdot)$ to be a similarity measure function. If the input $x$ is considered to be a target image that we want to find inside the search image $z$, the output of such model is a similarity score map, that has high values on the most probable target object locations.

Tracking is performed by first initializing the target region $t_0$ with the given object location and a corresponding search region $s_0 = \sigma(t_0)$ that should be big enough to accommodate for possible object position offset during the time difference between the input frames. For the frames $F_{\tau-1}$ and $F_\tau$, the target region from previous frame $t_{\tau-1}$ and the corresponding search region $s_{\tau-1}$ are processed by applying the Siamese model to create next target and search regions:

$$\begin{aligned} t_\tau &= t_{\tau-1} + \delta(f(\xi_t(t_{\tau-1}), \xi_s(s_{\tau-1}))), \\ s_\tau &= \sigma(t_\tau), \end{aligned} \tag{1}$$

where the $\delta(\cdot)$ function transforms the similarity map into the target position offset and $\xi_t(\cdot)$ and $\xi_s(\cdot)$ functions transform the target and search regions, respectively, into an image which is processed by the embedding function $\theta(\cdot)$. More details about these functions in Section III-B. This process is repeated for each new frame to update the object location.

SiamFC [6] uses a fully convolutional network as $\theta(\cdot)$ and creates a set of search regions for each target to address the possible change in size due to the projection distortion. We follow the approach of SiamFC and use PointPillars [21] to structure point clouds by generating voxel pseudo image, which serves as an input to the Siamese model, that uses a reduced PointPillars' Region Proposal Network (RPN) as $\theta(\cdot)$ and a similarity function $g(a,b) = a \otimes b$, where $\otimes$ indicates the correlation operator. In practice, because most DL frameworks perform correlation in convolutional layers, the similarity function can be implemented as $g(a,b) = \mathrm{conv2D}_{\omega=b}(a)$, where $\omega$ are the weights of the layer.

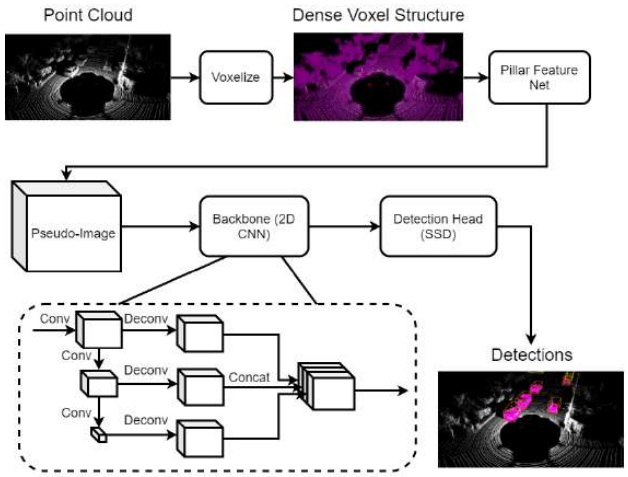



Fig. 1: Structure of PointPillars 3D object detection model. The RPN is a 2D CNN that takes a pseudo image as input.

The Region Proposal Network (RPN) in PointPillars is responsible for processing the input pseudo image using a fully convolutional network with 3 blocks of convolutional layers and 3 transposed convolutions that create same-sized features for final box regression and classification, as can be seen in Fig. 1. For Siamese tracking purposes, we are only interested in feature generation and there is no need for box regression and classification parts of the RPN. Therefore, we select convolutional blocks of the RPN to be used as a Feature Generation Network (FGN) $\theta(\cdot)$. The architecture of the model is shown in Fig. 2. The initial pipeline is similar to PointPillars for Detection, which includes voxelization of the input point cloud and processing it with the Pillar Feature Network to create a voxel pseudo image. The target and search regions of the pseudo image are processed by a Feature Generation Network, creating features that are compared by a cross-correlation module to find a position of the highest similarity, which serves as a new target position.

Each of the target ($t$) and search ($s$) regions are represented by a set of 5 values $(x, y, w, h, \alpha)$, where $(x, y)$ is the position of the region center in pseudo image space in pixels, $(w, h)$

[1]https://github.com/opendr-eu/opendr

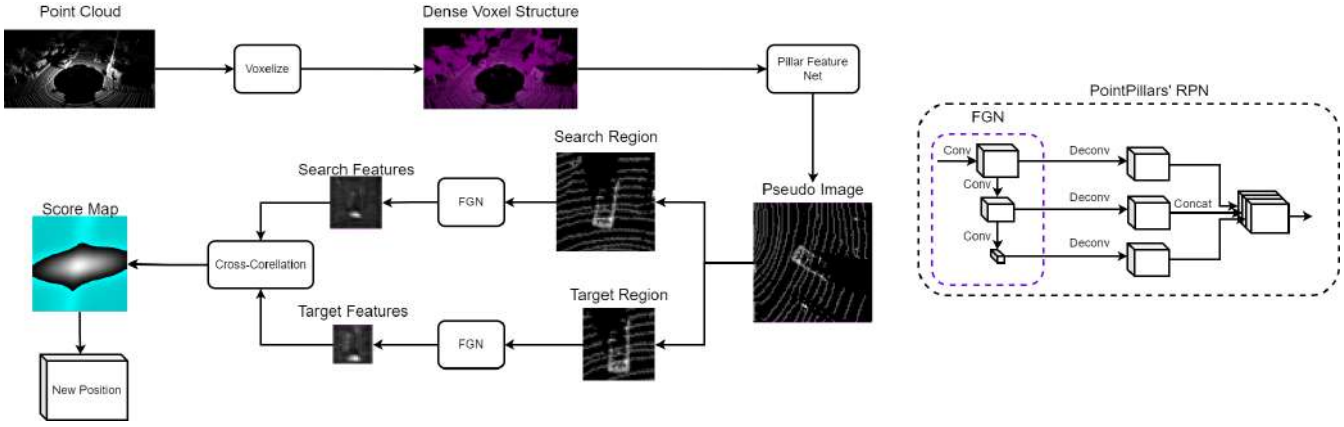



Fig. 2: Structure of the proposed Voxel Pseudo Image Tracking model. The input point cloud is voxelized and processed with the PointPillars' Pillar Feature Network to create a voxel pseudo image, which serves as an input to the Siamese model. The Feature Generation Network (FGN), which is a convolutional subnetwork of the Pointpillars' RPN, processes the target and search regions to create corresponding features that are then compared to find a position of the best similarity.

is the size of the region and $\alpha$ is the rotation angle. The ground truth bounding box $B_{gt}$ is described by the 3D position $(x, y, z)$, size $(w, h, d)$ and rotation $\alpha$. The corresponding target and search regions are created as follows:

$$\begin{aligned} t_0 &= \kappa_c((B^x_{gt}, B^y_{gt}, B^w_{gt}, B^h_{gt}, B^\alpha_{gt})), \\ s_0 &= \sigma(t_0), \end{aligned} \tag{2}$$

where $t_0$ is the initial target region, $s_0$ is the initial search region and the $\kappa_c(\cdot)$ is a function that adds context to the target region based on the amount of context parameter $c$:

$$\begin{aligned} \kappa_c(t) &= \begin{cases} \kappa_{c_+}(t), & \text{if } c > 0, \\ \kappa_{c_-}(t), & \text{otherwise}, \end{cases} \\ \kappa_{c_+}(x, y, w, h, \alpha) &= (x, y, m_n, m_n, \alpha), \\ m_n &= \sqrt{(w+m)(h+m)}, \\ m &= c(w+h), \\ \kappa_{c_-}((x, y, w, h, \alpha)) &= (x, y, w(1-c), h(1-c), \alpha), \end{aligned} \tag{3}$$

where $\kappa_{c_+}(\cdot)$ adds context to the object by making a square region that includes the original region inside, and $\kappa_{c_-}(\cdot)$ adds context by increasing each side of the region independently. The $\sigma(\cdot)$ function creates a search region, the size of which is defined by a hyperparameter $\sigma_s$ indicating the search region to target region size ratio, i.e.,:

$$\sigma(x, y, w, h, \alpha) = (x, y, \sigma_s w, \sigma_s h, \alpha). \tag{4}$$

The predicted output for the frame $\tau$ is computed as:

$$B_\tau = (t^x_\tau, t^y_\tau, B^z_{gt}, t^w_\tau, t^h_\tau, B^d_{gt}, t^\alpha_\tau). \tag{5}$$

### B. Training

We start from a pretrained PointPillars model on KITTI Detection dataset. The training can be performed on both KITTI Detection and KITTI Tracking datasets. Training on KITTI Detection dataset is done by considering objects separately and creating target and search region from their bounding boxes. For a set of ground truth boxes $\{B_{\mu_i} \,|\, i \in [1, N]\}$ in an input frame $\mu$, where $N$ is a number of ground truth objects in this frame, we create $N$ training samples by considering target-search pairs $\{(t_{\mu_i}, s_{\mu_i}) \,|\, i \in [1, N]\}$ created from these ground truth boxes as follows:

$$\begin{aligned} t_{\mu_i} &= \kappa_c((B^x_{\mu_i}, B^y_{\mu_i}, B^w_{\mu_i}, B^h_{\mu_i}, B^\alpha_{\mu_i})), \\ s_{\mu_i} &= \sigma_a(\sigma(t_{\mu_i}), t_{\mu_i}), \end{aligned} \tag{6}$$

where $\kappa_c(\cdot)$ and $\sigma(\cdot)$ functions are identical to the ones, described before, and $\sigma_a(\cdot)$ represents an augmentation technique where the center of the search region is shifted from the target center to imitate object movement between frames:

$$\sigma_a(s, t) = (t^x + \epsilon_x, t^y + \epsilon_y, s^w, s^h, s^\alpha), \tag{7}$$

with $\epsilon_x \sim \text{uniform}(-\frac{s^w - t^w}{2}, \frac{s^w - t^w}{2})$ and $\epsilon_y \sim \text{uniform}(-\frac{s^h - t^h}{2}, \frac{s^h - t^h}{2})$. In addition to the proposed augmentation, we use point cloud and ground truth boxes augmentations used in PointPillars, which include point cloud translation, rotation, point and ground truth database sampling.

For training on KITTI tracking dataset, we select the target and search regions from the same track $k$ and object $o_k$, but the corresponding point clouds and bounding boxes are taken from the different frames, modeling the variance of target object representation in time:

$$\begin{aligned} t_{o_k} &= \kappa_c((B^x_{f_t}, B^y_{f_t}, B^w_{f_t}, B^h_{f_t}, B^\alpha_{f_t})), \\ \hat{t}_{o_k} &= \kappa_c((B^x_{f_s}, B^y_{f_s}, B^w_{f_s}, B^h_{f_s}, B^\alpha_{f_s})), \\ s_{o_k} &= \sigma_a(\sigma(\hat{t}_{o_k}), \hat{t}_{o_k}), \end{aligned} \tag{8}$$

where $f_t$ and $f_s$ are a randomly selected frames from the track $k$ that contain the object of interest $o_k$. We balance the number of occurrences of the same object by selecting a constant number of $(f_t, f_s)$ samples for each object in the dataset.

After creating target and search regions, we apply $\xi_t(\cdot)$ and $\xi_s(\cdot)$ functions by first taking sub-images of those regions and

then, depending on the interpolation size parameter, applying bicubic interpolation to have a fixed size of the image:

$$\begin{aligned}\nu(x) &= \text{subimage}(\Pi(x), x),\\ \xi_\rho(x) &= \begin{cases}\text{bicubic}(\nu(x), \iota_\rho), & \text{if } \iota_\rho > 0,\\ \nu(x), & \text{otherwise,}\end{cases}\end{aligned} \tag{9}$$

where $x$ is a region to be processed, $\rho$ indicates that either target region $t$, or search region $s$ is selected, $\iota_r$ is an interpolation size parameter, which indicates the output size of the bicubic interpolation, and $\Pi(x)$ creates an AABB pseudo image from the input point cloud that contains the region $x$. These images are processed with the Feature Generation Network, which is the convolutional subnetwork of the PointPillars' RPN, as shown in Fig. 2.

The resulting features are compared using the cross-correlation module to create a score map. A Binary Cross-Entropy (BCE) loss is used to compare a predicted score map $M_x$ to the true label $M_y$ as follows:

$$\begin{aligned}BCE(M_x, M_y) &= \frac{1}{N}\sum_1^N l_n,\\ l_n &= -w_n\Big(M_{y_n}\log S(M_{x_n}) + (1 - M_{y_n})\log(1 - S(M_{x_n}))\Big),\end{aligned} \tag{10}$$

where $S(x) = 1/(1 + e^{-x})$ and $w_n$ is a scaling weight for each element. We follow SiamFC [6] and select the values of $w_n$ to equalize the total weight of positive and negative pixels on the ground truth score map.

The true label is created by placing positive values on a score map within a small distance of the projected target center, and zeros everywhere else. The value $v(p_x, p_y)$ of a pixel $(p_x, p_y)$ in a label map depends on its distance to the target center $d(p_x, p_y)$ and a hyperparameter $r$, which describes the radius of positive labels around the center:

$$\begin{aligned}&v(x, y) =\\ &\begin{cases}v_{\min}\frac{d(p_x,p_y)}{r} + v_{\max}(1 - \frac{d(p_x,p_y)}{r}), & \text{if } d(p_x, p_y) \le r + 1,\\ 0, & \text{otherwise,}\end{cases}\\ &d(p_x, p_y) = \sqrt{(p_x - c_x)^2 + (p_y - c_y)^2},\end{aligned} \tag{11}$$

where $(c_x, c_y)$ is the target center, $[v_{\min}, v_{\max}]$ is the range of values assigned to the pixels inside the circle of radius $r$, and the pixels inside the $(r, r + 1]$ range are assigned to values, lower than $v_{\min}$.

### C. Inference

Inference is split into two parts, i.e., initialization and tracking. The 3D bounding box of the object of interest is given by either a detection method, a user, or from a dataset, to create the initial target and its features, that will be used for the future tracking. During tracking, the last target position is used to place a search region, but instead of using only a single search region, we create multiple search regions with identical size and different rotations. This is done to allow to correct for rotation changes in the object. The set of $2K + 1$ search regions $\{s_{\tau_i} \mid i \in [-K, K]\}$ is created by altering the initial search region $s_{\tau_0}$, i.e., $s^{x,y,w,h}_{\tau_i} = s^{x,y,w,h}_{\tau_0}$ and $s^{\alpha}_{\tau_i} = s^{\alpha}_{\tau_0} + i\psi_\alpha$, where $\psi_\alpha$ is a rotation step between consequent search regions.

Target features are compared with all search features, and the one with the highest score is selected for a new rotation:

$$\begin{aligned}s_{\tau_{\max}} &= \underset{s_{\tau_i}}{\text{argmax}}\ \Lambda_i \max(f(\xi_t(t_\tau), \xi_s(s_{\tau_i}))),\\ t^{\alpha}_{\tau+1} &= \mu s^{\alpha}_{\tau_{\max}} + (1 - \mu)t^{\alpha}_{\tau},\end{aligned} \tag{12}$$

where $\Lambda_i$ is a rotation penalty multiplier, which is 1 for $i = 0$ and a hyperparameter value in $[0, 1]$ range for all other search regions, $\mu$ is a rotation interpolation coefficient and $f$ is a Siamese model.

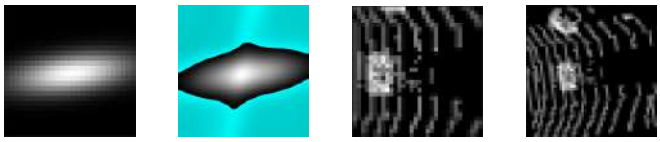
The score map $M$, obtained from the Siamese model, is upscaled with bicubic interpolation, increasing its size 16 times by default (score upscale parameter $u_M$). Based on the assumption that an object cannot move far from its previous position in consecutive frames, we use a penalty technique for the scores that are far from the center, by multiplying scores with a weighted penalty map. The penalty map is formed using either Hann window or a 2D Gaussian function. The maximum score position $(x, y)_{\max}$ of the upscaled score map is translated back from the score coordinates to image coordinates as follows:

$$\begin{aligned}(x, y)_{\max} &= \underset{(x,y)}{\text{argmax}}\ H(f(\xi_t(t_\tau), \xi_s(s_{\tau_{\max}})))_{(x,y)},\\ (x, y)^{\text{img}}_{\max} &= (x, y)_{\max}\frac{s^{(w,h)}_{\tau_{\max}}}{H_{\text{size}}},\\ H_{\text{size}} &= u_M M_{\text{size}},\\ H(M) &= \eta P(H_{\text{size}}) + (1 - \eta)\,\text{bicubic}(M, u_M, M_{\text{size}}),\end{aligned} \tag{13}$$

where $(x, y)^{\text{img}}_{\max}$ is a target position offset, which represents a prediction of the target object's movement, $H$ applies score interpolation and a penalty map $P(\text{size})$ to the upscaled output of the Siamese model, $M_{\text{size}}$ and $H_{\text{size}}$ are the 2D sizes of the original score map and the upscaled score map, respectively, and $\eta$ is a window influence parameter that controls how much the Hann window or a Gaussian penalty influence the score map.

After the current frame's prediction is ready, the search region is centered on a prediction, assuming that the object's position on the next frame should be close to its position on the previous frame. In order to make it easier to find the position of an object inside a search region, we apply a linear position extrapolation for the search region by centering it not on a previous prediction, but on a possible new prediction position. Given a target region from the last frame $t_{\tau-1}$ and the predicted target region $t_\tau$, the corresponding search region has its position defined $s^{x,y}_\tau = 2t^{x,y}_\tau - t^{x,y}_{\tau-1}$. This approach increases the chances of the target to be in the center of the search region or having a small deviation from it.

When the linear extrapolation is used, the penalty map $P(\text{size})$ (Eq. 13) is created using a 2D Gaussian with a covariance matrix $\Sigma$ that represents the angle of the extrapolation vector to penalize more positions that are not on the way of the predicted object's movement. This is done by creating a



(a) Gaussian penalty (b) Score map (c) Target pseudo image (d) Search pseudo image

Fig. 3: An example of a directional Gaussian penalty used with linear extrapolation, a corresponding score map and target and search pseudo images. Cyan color on the score map represents negative values.

Gaussian with independent variables and variances $\sigma_+$ for the desired direction and $\sigma_-$ for the opposite direction:

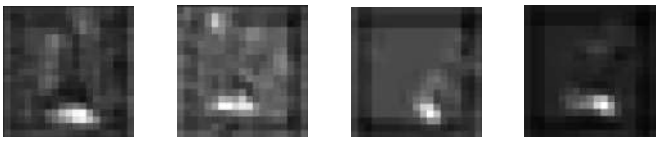
$$\mu = 0 \quad \text{and} \quad \Sigma_0 = \begin{bmatrix} \sigma_- & 0 \\ 0 & \sigma_+ \end{bmatrix}. \tag{14}$$

Bigger $\sigma_+$ allows a further offset alongside the extrapolation vector, while bigger $\sigma_-$ allows a higher deviation from it. The resulting Gaussian $\mathcal{N}(\mu, \Sigma_0)$ is rotated by $\phi = \arctan(e)$, where $e$ is the extrapolation vector, as follows:

$$R = \begin{bmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{bmatrix} \quad \text{and} \quad \Sigma = R\Sigma_0^{-1}R^T. \tag{15}$$

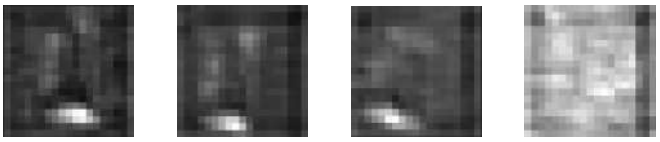
The creation of such a map for a high-dimensional upscaled score map is a computationally expensive task, and therefore, to optimize it, the penalty maps are hashed by their size and rotation to reuse in future frames. There is no need in keeping high precision of rotation value, as the penalty map will be almost identical for close $\phi$ values and will result in the same basis for predictions. Keeping this in mind, we define a rotation hash function $H_r(\phi) = \left\lfloor \frac{n\phi}{2\pi} \right\rfloor$. This hash function divides the full circle on $n$ equal regions, and the penalty map is taken for the closest $\phi$-sector. Fig. 3 shows an example of a Gaussian penalty created using this approach, a corresponding score map and target and search pseudo images.

Target features are created at the first frame and then used to find the object during the whole tracking sequence. However, due to a high variance of object representation on different distances to Lidar, initial features may be too far from the current representation of the object, as can be seen from Fig. 4. This problem may be resolved by using target features from the latest frame, but such approach leads to an error accumulation and drifts the target region to a background object. In order to balance between the aforementioned approaches, we introduce a target feature merge scale $m_{\text{tf}}$. Using the $m_{\text{tf}} = 0$, only the initial target features will be used through the tracking task, while using the $m_{\text{tf}} = 1$ results in the latest frame's target features overwriting the previous ones. Fig. 5 shows the effect of a high merge scale value $m_{\text{tf}} = 0.5$. The target region is slowly drifting to the right starting from frame 10 and the object is completely lost at frame 30. We use small values of $m_{\text{tf}}$ in range $[0, 0.01]$ in ablation study. This allows to keep the target features stable and not drift out from the object, while still having the possibility to update features if the distance to the object changes.



(a) $\tau = 0$ (b) $\tau = 40$ (c) $\tau = 60$ (d) $\tau = 80$

Fig. 4: Target features of the same object at different frames.



(a) $\tau = 0$ (b) $\tau = 10$ (c) $\tau = 20$ (d) $\tau = 30$

Fig. 5: Merged target features of the same object at different frames with $0.5$ merge scale. Starting from frame 10, the object representation drifts to the right and completely loses the target at frame 30.

By default, the model's position offset predictions are applied directly, but we can improve prediction stability by introducing the offset interpolation parameter $\omega$. Given the previous frame's target position $\zeta_{\tau-1}$ and a prediction for the current frame $\hat{\zeta}_\tau$, the interpolated position $\zeta_\tau$ is defined as follows:

$$\zeta_\tau = \omega\zeta_{\tau-1} + (1-\omega)\hat{\zeta}_\tau. \tag{16}$$

Current 3D detection and tracking datasets provide annotations for a rotation around the vertical axis only, which is sufficient for current robotic and autonomous driving tasks, but it may be too coarse for future tasks. For datasets with rotations around all axes and not only the vertical one, an additional regression branch can be added that, applied to a target region, will predict other rotation angles. A similar process can be applied to non-rigid objects in order to continuously update their dimensions.

TABLE I: Results of 3D Car tracking on KITTI dataset. Modality represents the type of data the tracking is performed on (PC for point cloud, BEV for Birds-Eye-View and VPI for voxel pseudo image). FPS values are reported on a 1080Ti GPU by a corresponding paper. FPS values with a star notation are obtained by running the methods' official implementations on a 1080ti GPU considering the full runtime of the network.

| Method | Modality | Success | Precision | FPS |
|---|---|---|---|---|
| 3DSRPN PCW [5] | PC | 56.32 | 73.40 | 16.7 |
| 3DSRPN PW [5] | PC | 57.25 | 75.03 | 20.8 |
| SCD3D-KF [13] | PC | 40.09 | 56.17 | 2.2 |
| SCD3D-EX [13] | PC | **76.94** | 81.38 | 1.8 |
| 3D Siam-2D [17] | PC+BEV | 36.3 | 51.0 | - |
| BAT [14] | PC | 65.38 | 78.88 | 23.96* |
| PTT-Net [10] | PC | 67.8 | **81.8** | 39.51* |
| P2B [9] | PC | 56.2 | 72.8 | 30.18* |
| VPIT (Ours) | VPI | 50.49 | 64.53 | **50.45** |

TABLE II: Evaluation of the fastest methods for 3D Car tracking on KITTI dataset. The evaluation is performed with official implementations on high-end and embedded GPU platforms with different combinations of GPU/CPU. 32C CPU and 104C CPU correspond to 32-core and 104-core CPUs, respectively.

| Method | Modality | Success | Precision | FPS | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | 1080Ti 32C CPU | 2080 32C CPU | 2080Ti 104C CPU | TX2 | Xavier |
| P2B [9] | PC | 56.2 | 72.8 | 30.18 | 26.34 | 38.93 | 6.20 | 10.37 |
| PTT-Net [10] | PC | **67.8** | **81.8** | 39.51 | 33.34 | 50.25 | 8.04 | 13.49 |
| VPIT (Ours) | VPI | 50.49 | 64.53 | **50.45** | **52.52** | **72.53** | **14.61** | **20.55** |

## IV. Experiments

We use KITTI [26] Tracking training dataset split to train and test our model, with tracks 0-18 for training and validation and tracks 19-20 for testing (as is common practice [5], [9], [10], [13], [17]). We use Precision and Success metrics as defined in One Pass Evaluation [28]. Precision is computed based on the difference between ground-truth and predicted object centers in 3D. Success is computed based on the 3D Intersection over Union (IoU) between predicted and ground truth 3D bounding boxes.

We use $1$ feature block from the original PointPillars model with $4$ layers in a block. The model is trained for $64,000$ steps with BCE loss, $1 * 10^{-5}$ learning rate and $2$ positive label radius. During inference, rotations count of $3$, rotation step of $0.15$ and rotation penalty of $0.98$ are used, together with $0.85$ penalty map multiplier, score upscale of $8$, target/search size of $(0, 0)$ (original sizes are used), context amount of $0.27$, rotation interpolation of $1$, offset interpolation of $0.3$, target feature merge scale of $0.005$ and linear search position extrapolation.

### A. Comparison with state-of-the-art

Evaluation results on 1080Ti GPU are given in Table I. Even though we are interested in embedded devices, we first measure FPS on a 1080Ti, since this is the most commonly used GPU amongst the compared methods, as a relative speed measure between them. VPIT is the fastest method and achieves competitive Precision and Success. We evaluate official implementations of the fastest methods (P2B, PTT, VPIT) on different high-end and embedded GPU platforms to showcase how the architecture of the device influences the models' FPS.

When computing FPS, we include all the time the model spends to process the input and create a final result. This excludes time spend to obtain data and to compute Success and Precision values, but includes pre- and post-processing steps.

As can be seen from Table II, in terms of speed, VPIT outperforms P2B by 67% on 1080Ti GPU, 99% on 2080 GPU and 86% on 2080Ti GPU with 104-core CPU. For embedded devices, the difference is bigger: VPIT outperforms P2B by 135% on TX2 and by 98% on Xavier. This indicated that the approach of VPIT is more suited to the robotic systems, where embedded devices are the most common, and it is harder to meet real-time requirements.

### B. Real-time evaluation

Application of 3D tracking methods is usually done for robotic systems that do not use high-end GPUs due to their high power consumption, but instead apply the computations on embedded devices, such as TX2 or Xavier. Following [11] we implement a predictive real-time benchmark. Given a time step $\tau$, the set of inputs $S_{\text{in}}$ that is visible to the model is limited by those inputs that appeared before $\tau$, i.e., $S_{\text{in}} = (x_i, \tau_i | i \leq \tau)$, where $(x_i, \tau_i)$ is a pair of an input frame and a corresponding time step. The processing time of the model is higher than zero, which means that the resulting prediction $p_i$ at time $\hat{\tau}_i$ cannot be compared to the label $y_i$ from the same frame, and therefore for each label $y_i$ from the dataset, it will be compared to the latest prediction $p_{l_{\text{pr}}(i)}$. The predictive real-time error $E_{\text{pr}}$, based on a regular error $E$, is computed for each ground truth label $y_i$ as follows:

$$\begin{aligned} E_{\text{pr}}(y_i) &= E(y_i, p_{l_{\text{pr}}(i)}), \\ l_{\text{pr}}(i) &= \underset{j}{\text{argmax}}\ \hat{\tau}_j \leq \tau_i. \end{aligned} \tag{17}$$

As shown in [11], the existing model can be improved for a predictive real-time evaluation by predicting the position of the object at a frame $i$ and then predicting its movement during the time period between the frames $i$ and $i+1$ using a Kalman Filter [29] or any other similar method. Such optimization can be applied to any of the methods that we compare, and therefore, to eliminate the factor of "predictive errors" where the model's error in prediction for the current frame is more leaned towards the prediction for the next frame, we introduce a non-prediction real-time benchmark. This benchmark effectively shifts all labels one frame forward, allowing the methods that are faster than the data FPS to be evaluated in the same way, as in a regular evaluation protocol. The only difference between the non-predictive and predictive benchmarks is in the way the latest prediction is defined:

$$l_{\text{npr}}(i) = \underset{j}{\text{argmax}}\ \hat{\tau}_j \leq \tau_{i+1}. \tag{18}$$

We evaluate the fastest 3D SOT methods (P2B, PTT, VPIT) on embedded devices with both non-predictive (Table III) and predictive (Table IV) benchmarks. We select Data FPS to be either 10 or 20, representing the most popular 10 and 20 Hz Lidars. The 10Hz Lidar is the easier case, as for the method to be real-time, it needs to sustain FPS, higher than 10, compared to 20 for the 20 Hz Lidar. During the evaluation, we compute

TABLE III: Evaluation of the fastest methods for 3D Car tracking on KITTI dataset in real-time settings without the predictive requirement. The evaluation is performed with official implementations on embedded GPU platforms for 10 and 20 Hz Lidars (Data FPS). Frame drop represents the percentage of frames that could not be processed due to the model's latency. Regular represents evaluation without real-time requirements.

| **Method** | **Data FPS** | **Success (non-predictive)** | | | **Precision (non-predictive)** | | | **FPS** | | **Frame drop** | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Regular | TX2 | Xavier | Regular | TX2 | Xavier | TX2 | Xavier | TX2 | Xavier |
| P2B [9] | 10 | 56.20 | 21.90 | 36.50 | 72.80 | 21.70 | 42.40 | 6.17 | 10.07 | 37.41% | 7.00% |
| PTT-Net [10] | 10 | **67.80** | 29.50 | **63.60** | **81.80** | 30.00 | **75.10** | 6.90 | 12.38 | 28.21% | 0.81% |
| VPIT (Ours) | 10 | 50.49 | **50.31** | 50.49 | 64.53 | **64.08** | 64.53 | **14.31** | **20.55** | **0.68%** | **0.00%** |
| P2B [9] | 20 | 56.20 | 10.90 | 16.70 | 72.80 | 7.90 | 15.0 | 5.54 | 9.61 | 70.57% | 51.56% |
| PTT-Net [10] | 20 | **67.80** | 17.90 | 26.50 | **81.80** | 15.70 | 26.60 | 6.61 | 11.88 | 64.14% | 38.95% |
| VPIT (Ours) | 20 | 50.49 | **38.96** | **47.70** | 64.53 | **45.50** | **59.87** | **14.37** | **20.06** | **30.17%** | **2.38%** |

TABLE IV: Evaluation of the fastest methods for 3D Car tracking on KITTI dataset in real-time settings with the predictive requirement. The evaluation is performed with official implementations on embedded GPU platforms for 10 and 20 Hz Lidars (Data FPS). Frame drop represents the percentage of frames that could not be processed due to the model's latency. Regular represents evaluation without real-time requirements

| **Method** | **Data FPS** | **Success (predictive)** | | | **Precision (predictive)** | | | **FPS** | | **Frame drop** | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Regular | TX2 | Xavier | Regular | TX2 | Xavier | TX2 | Xavier | TX2 | Xavier |
| P2B [9] | 10 | 56.20 | 19.10 | 30.30 | 72.80 | 17.80 | 33.70 | 6.17 | 10.07 | 36.31% | 6.79% |
| PTT-Net [10] | 10 | **67.80** | 24.90 | **50.70** | **81.80** | 23.40 | 59.00 | 7.07 | 12.26 | 26.15% | 0.90% |
| VPIT (Ours) | 10 | 50.49 | **45.82** | 46.68 | 64.53 | **57.76** | **59.28** | **14.61** | **20.55** | **0.57%** | **0.00%** |
| P2B [9] | 20 | 56.20 | 9.10 | 13.50 | 72.80 | 6.00 | 10.90 | 5.54 | 9.47 | 69.57% | 50.31% |
| PTT-Net [10] | 20 | **67.80** | 15.40 | 23.20 | **81.80** | 13.10 | 21.50 | 6.67 | 12.06 | 62.49% | 37.35% |
| VPIT (Ours) | 20 | 50.49 | **34.00** | **41.39** | 64.53 | **36.61** | **49.36** | **14.40** | **20.77** | **29.41%** | **1.56%** |

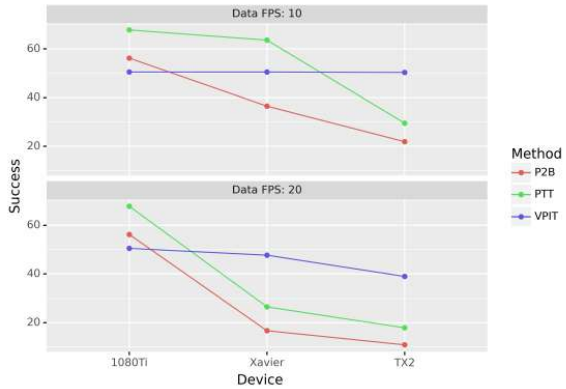



Fig. 6: Evaluation of the fastest models with real-time requirements on embedded devices and a desktop GPU. Data FPS represents 10 and 20 Hz Lidars. Devices are sorted in descending order by their computational power.

how many frames could not be processed due to the model's latency and represent it as a Frame drop percentage.

Only VPIT on Xavier (more powerful embedded device) for a 10Hz Lidar could not suffer from any frame drop, resulting in the regular Success and Precision values for a non-predictive benchmark and suffering $3.81$ Success and $5.25$ Precision for a predictive benchmark. The highest Frame drop of $60-70\%$ is seen for P2B and PTT on TX2 for a 20Hz Lidar. In this case, Success values of P2B and PTT are $6$ and $4$ times lower than during the regular evaluation, respectively, which indicates that these methods cannot work under the aforementioned conditions. For every evaluation case, except the Xavier with a 10Hz Lidar, VPIT outperforms P2B and PTT in Success, Precision, FPS and Frame drop, but PTT still maintains the best Success and Precision on Xavier with a 10Hz Lidar. As can be seen from Fig. 6, Success drop of VPIT is the smallest, while P2B and PTT lose most of their tracking accuracy on TX2 for any Lidar and on Xavier for a 20Hz Lidar.

Evaluation of the other methods, presented in Table I, is not needed, as their FPS is 2 or more times slower than the FPS of P2B and PTT, which will result in a complete loss of Success and Precision on any of the real-time benchmarks that were discussed.

### C. Ablation study

We performed experiments to determine the influence of different hyperparameters on the Success and Precision metrics. We used KITTI Tracking tracks $10-11$ for validation and $0-9$, $12-18$ for training. The influence of selected hyperparameters on Success and Precision metrics is similar, and therefore we present only the effect on Success metric on validation subset in Fig. 7.

Decreasing the number of feature blocks from the Feature Generation Network, leads to better Success values. This can be due to an increasing receptive field with each new block that leads to over smoothing of the resulting feature maps and makes it harder to define a precise position of an object of interest.

Next, we follow SiamFC [6] and use target and search upscaling with $\iota_t = (127, 127)$ and $\iota_s = (255, 255)$. This results in constant-sized target and search pseudo-images, corresponding feature and score maps, which allows implementing mini-batch training and reduce the training time.

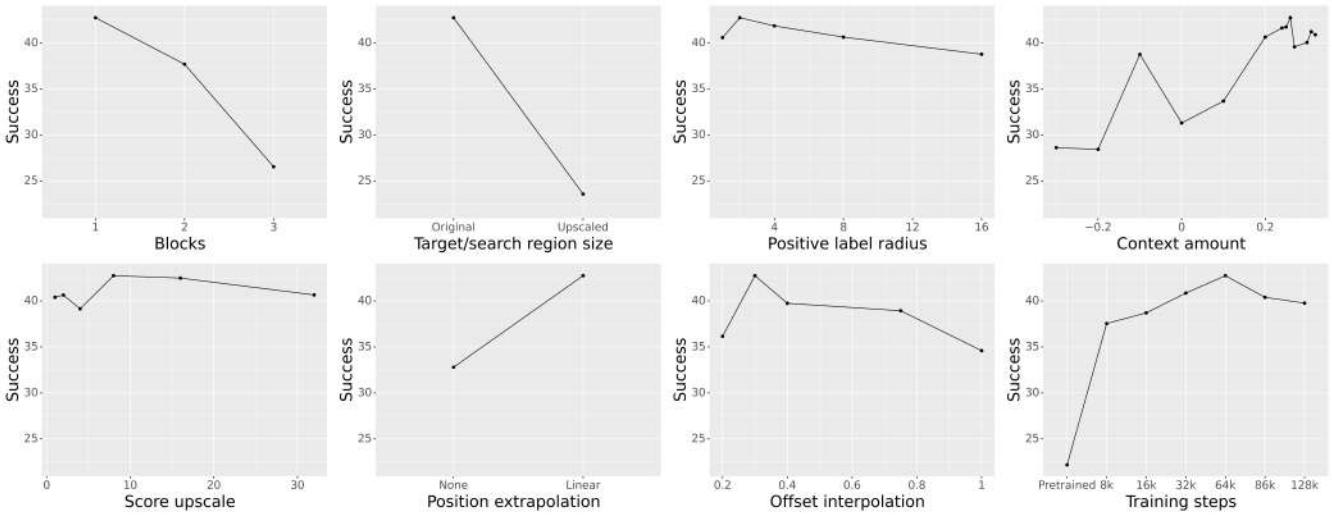



Fig. 7: Influence of different hyperparameter values on Success metric on validation KITTI Tracking subset.

However, such approach does not work well for voxel pseudo images, achieving $81\%$ less Success than when using original sizes ($\iota_t = (0, 0)$ and $\iota_s = (0, 0)$).

Furthermore, following SiamFC [6], positive values of the context amount parameter $c$ result in square target and search regions. In contrast to it, we also use negative $c$ values to add independent context to each dimension of the region, keeping its aspect ratio. Positive context leads to better Success with the maximum at $c = 0.26$, which is $10\%$ higher than the result for $c = -0.1$.

Regarding the positive label radius, $r = 2$ results in the best training procedure, but this hyperparameter has a low influence on the final Success value, as well as the target feature merge scale $m_{\text{tf}}$, window influence $\eta$, score upscale $u_M$, search region scale $\sigma_s$, rotation step $\psi_\alpha$ and rotation penalty $\Lambda$.

Next, linear position extrapolation leads to a $30\%$ increase in Success compared to a conventional search region placement and no directional penalty. With offset interpolation $\omega = 0.3$, the Success value increases by $24\%$, compared to the direct application of the model's prediction ($\omega = 1$).

Finally, a pretrained PointPillars backbone creates features that are capable of differentiating between target class and other object types, but those features cannot be used effectively to differentiate between different objects from the same class. This serves as a good starting point for the SOT task, and then training for $64000$ steps results in the best model for selected hyperparameters.

## V. Conclusions

In this work, we proposed a novel method for 3D SOT called VPIT. We focus on stepping out of the point-based approaches for this problem and studying methods to use structured data for 3D SOT. We formulate a lightweight method that uses PointPillars' pseudo images as a search space and apply a Siamese 2D-like approach on these pseudo images to find both position and rotation of the object of interest. Experiments show that VPIT is the fastest method, and it achieves competitive Precision and Success values. Moreover, we implement a real-time evaluation protocol for 3D single object tracking and use it for evaluation of the fastest methods on embedded devices, that are the most popular choice for robotic systems. Results showcase that other methods are less suited for such devices and lose most of their ability to track objects due to a high latency of predictions. The proposed method allows for an easy adaptation of 2D tracking ideas to a 3D SOT task while keeping the model lightweight.

## References

[1] "Kitti 3d object detection leaderboard," 2022, [Accessed 26-January-2022]. [Online]. Available: http://www.cvlibs.net/datasets/kitti/eval_object.php?obj_benchmark=3d

[2] D. S. Bolme, J. R. Beveridge, B. A. Draper, and Y. M. Lui, "Visual object tracking using adaptive correlation filters," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2010, pp. 2544–2550.

[3] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, "High-speed tracking with kernelized correlation filters," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 3, p. 583–596, 2015.

[4] D. Held, S. Thrun, and S. Savarese, "Learning to track at 100 fps with deep regression networks," *1604.01802*, 2016.

[5] Z. Fang, S. Zhou, Y. Cui, and S. Scherer, "3d-siamrpn: An end-to-end learning method for real-time 3d single object tracking using raw point cloud," *IEEE Sensors Journal*, vol. 21, no. 4, p. 4995–5011, 2021.

[6] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, and P. H. Torr, "Fully-convolutional siamese networks for object tracking," *arXiv:1606.09549*, 2016.

[7] B. Li, J. Yan, W. Wu, Z. Zhu, and X. Hu, "High performance visual tracking with siamese region proposal network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8971–8980.

[8] B. Li, W. Wu, Q. Wang, F. Zhang, J. Xing, and J. Yan, "Siamrpn++: Evolution of siamese visual tracking with very deep networks," *arXiv:1812.11703*, 2018.

[9] H. Qi, C. Feng, Z. Cao, F. Zhao, and Y. Xiao, "P2b: Point-to-box network for 3d object tracking in point clouds," *arXiv:2005.13888*, 2020.

[10] J. Shan, S. Zhou, Z. Fang, and Y. Cui, "Ptt: Point-track-transformer module for 3d single object tracking in point clouds," *arXiv:2108.06455*, 2021.

[11] M. Li, Y.-X. Wang, and D. Ramanan, "Towards streaming perception," in *European Conference on Computer Vision*. Springer, 2020, pp. 473–488.

[12] Y. Li, L. Ma, Z. Zhong, F. Liu, M. A. Chapman, D. Cao, and J. Li, "Deep learning for lidar point clouds in autonomous driving: A review," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 8, pp. 3412–3432, 2021.

[13] S. Giancola, J. Zarzar, and B. Ghanem, "Leveraging shape completion for 3d siamese tracking," 2019.

[14] C. Zheng, X. Yan, J. Gao, W. Zhao, W. Zhang, Z. Li, and S. Cui, "Box-aware feature enhancement for single object tracking on point clouds," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 13 199–13 208.

[15] H. Zou, J. Cui, X. Kong, C. Zhang, Y. Liu, F. Wen, and W. Li, "F-siamese tracker: A frustum-based double siamese network for 3d single object tracking," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 8133–8139.

[16] S. Jiayao, S. Zhou, Y. Cui, and Z. Fang, "Real-time 3d single object tracking with transformer," *IEEE Transactions on Multimedia*, 2022.

[17] J. Zarzar, S. Giancola, and B. Ghanem, "Efficient bird eye view proposals for 3d siamese tracking," *arXiv:1903.10168*, 2020.

[18] M. Kristan, J. Matas, A. Leonardis, M. Felsberg, R. Pflugfelder, J.-K. Kämäräinen, H. J. Chang, M. Danelljan, L. Cehovin, A. Lukežič *et al.*, "The ninth visual object tracking vot2021 challenge results," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 2711–2738.

[19] P. Nousi, A. Tefas, and I. Pitas, "Dense convolutional feature histograms for robust visual object tracking," *Image and Vision Computing*, vol. 99, p. 103933, 2020.

[20] Y. Zhou and O. Tuzel, "VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[21] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "PointPillars: Fast Encoders for Object Detection from Point Clouds," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[22] Z. Liu, X. Zhao, T. Huang, R. Hu, Y. Zhou, and X. Bai, "TANet: Robust 3D Object Detection from Point Clouds with Triple Attention," in *AAAI Conference on Artificial Intelligence*, 2020.

[23] Q. Chen, L. Sun, Z. Wang, K. Jia, and A. L. Yuille, "Object as Hotspots: An Anchor-Free 3D Object Detection Approach via Firing of Hotspots," in *European Conference on Computer Vision*, 2020.

[24] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," *arXiv:1612.00593*, 2017.

[25] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," *arXiv:1706.02413*, 2017.

[26] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? The KITTI vision benchmark suite," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[27] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuScenes: A Multimodal Dataset for Autonomous Driving," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

[28] M. Kristan, J. Matas, A. Leonardis, T. Vojir, R. Pflugfelder, G. Fernandez, G. Nebehay, F. Porikli, and L. Cehovin, "A novel performance evaluation methodology for single-target trackers," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 11, p. 2137–2155, 2016.

[29] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *Journal of Basic Engineering*, vol. 82, no. 1, pp. 35–45, 1960.

## 7.5 3D Multi-Object Tracking Using Graph Neural Networks With Cross-Edge Modality Attention

The appended paper [12] follows.

# 3D Multi-Object Tracking Using Graph Neural Networks with Cross-Edge Modality Attention

Martin Büchner and Abhinav Valada

***Abstract*—Online 3D multi-object tracking (MOT) has witnessed significant research interest in recent years, largely driven by demand from the autonomous systems community. However, 3D offline MOT is relatively less explored. Labeling 3D trajectory scene data at a large scale while not relying on high-cost human experts is still an open research question. In this work, we propose Batch3DMOT which follows the tracking-by-detection paradigm and represents real-world scenes as directed, acyclic, and category-disjoint tracking graphs that are attributed using various modalities such as camera, LiDAR, and radar. We present a multi-modal graph neural network that uses a cross-edge attention mechanism mitigating modality intermittence, which translates into sparsity in the graph domain. Additionally, we present attention-weighted convolutions over frame-wise k-NN neighborhoods as suitable means to allow information exchange across disconnected graph components. We evaluate our approach using various sensor modalities and model configurations on the challenging nuScenes and KITTI datasets. Extensive experiments demonstrate that our proposed approach yields an overall improvement of 3.3% in the AMOTA score on nuScenes thereby setting the new state-of-the-art for 3D tracking and further enhancing false positive filtering.**

## I. Introduction

3D multi-object tracking (MOT) is an essential component of the scene understanding pipeline of autonomous robots. It aims at inferring associations between occurrences of object instances at different time steps in order to predict plausible 3D trajectories. These trajectories are then used in various downstream tasks such as trajectory prediction [1] and navigation [2]. Tracking multiple objects under real-time constraints in an online setting is challenging due to both intermediate track prediction when facing false negatives and robust false positive filtering. Owing to recent advances in LiDAR-based object detection [3], the 3D tracking task has also seen significant performance improvements.

Real-world deployment of these online methods in areas such as autonomous driving poses several challenges. When requiring regulatory approval, its robust behavior must be demonstrated on large sets of reference data which is arduous to obtain due to the lack of extensive ground truth. Therefore, performing high-quality offline labeling of real-world traffic scenes provides the means to test online methods on a larger scale and further sets a benchmark for what is within the realms of possibility for online methods. With respect to generating pseudo ground truth, our proposed method aims at minimizing the number of false positive trajectories at high recalls.

In this paper, we present Batch3DMOT, an offline 3D tracking framework that follows the tracking-by-detection paradigm and utilizes multiple sensor modalities (camera, LiDAR, radar) to solve a multi-frame, multi-object tracking objective. Sets of 3D object detections per frame are first turned into attributed nodes. In order to learn offline 3D tracking, we employ a graph neural network (GNN) that performs time-aware neural message passing with intermediate frame-wise attention-weighted neighborhood convolutions. Different from popular Kalman-based approaches such as AB3DMOT [4], which essentially tracks objects of different semantic categories independently, our method uses a single model that operates on category-disjoint graph components. As a consequence, it leverages inter-category similarities to improve tracking performance. While Brasó *et al.* were able to solve a single-category 2D offline tracking objective using graph neural networks [5], this work focuses on the 3D MOT task while ensuring balanced performance across different semantic categories.

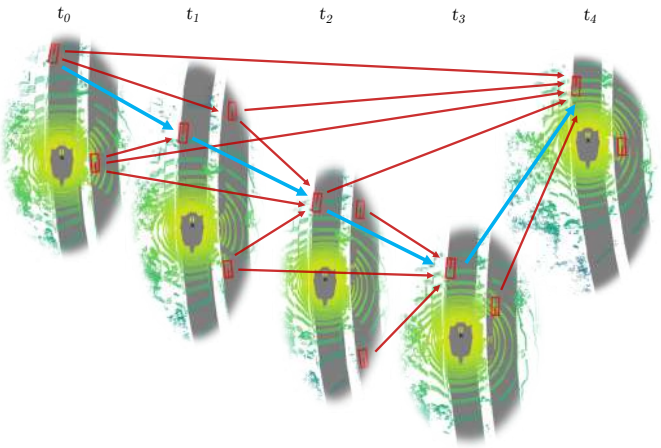



Fig. 1: Birds-eye view visualization of a 3D offline tracking scenario showing the road surface and LiDAR point clouds recorded at different time steps. The goal in tracking is to find plausible chains of edges connecting objects across time that best explain the evolution of an object instance. This representation only shows the edges accompanying a single object instance.

When evaluating typically used modalities such as LiDAR, we can make a striking observation: On the one hand, detection features such as bounding box size or orientation are consistently available across time. A similar observation can be made for camera features, even if the object is (partially) occluded. On the other hand, sensor modalities such as LiDAR or radar do not necessarily share this availability. Due to their inherent sparsity, constructing a feature, e.g., for faraway objects, is typically impractical as it does not serve as a discriminative feature that can be used in tracking. This potential modality intermittence translates to sparsity in the graph domain, which is tackled in this work using our proposed cross-edge modality attention. This enables an edge-wise agreement on whether to include the particular modality in node similarity finding.

Our main contributions can be summarized as follows:

Department of Computer Science, University of Freiburg, Germany.
This work was funded by the European Union's Horizon 2020 research and innovation program under grant agreement No 871449-OpenDR.
This paper provides supplementary material at *https://arxiv.org/abs/2203.10926*.

- A novel multimodal GNN framework for offline 3D multi-object tracking on multi-category tracking graphs including k-NN neighborhood attention across semantic graph components.
- A cross-edge attention mechanism that uses intermittent sensor data to substantiate the differentiation between active and inactive edges.
- Methodology and pre-processing pipeline for constructing category-disjoint tracking graphs over multiple timesteps as well as a novel agglomerative trajectory clustering scheme for effective trajectory generation.
- Extensive evaluations and ablation study on the nuScenes [6] and KITTI [7] datasets using different detection approaches.
- The code and pre-trained models are publicly available at http://batch3dmot.cs.uni-freiburg.de.

## II. Related Work

Multi-object tracking (MOT) can be categorized into online and offline settings. Whereas online methods are limited to using past and current data, offline methods can efficiently leverage future data to find solutions to the global data association problem. Besides a temporal categorization, MOT can be applied in the 2D [8]–[10] or the 3D domain [3], [4], [11], [12], exploiting either 2D or 3D object detections. Finally, two commonly followed approaches involve the tracking-by-detection paradigm [3], [5], [8] and joint object detection and tracking [9]. In this section, we briefly review offline 2D MOT and discuss selected 3D MOT methods, relevant to our work.

*2D Multi-Object Tracking*: 2D MOT has been extensively studied by the scientific community. Often, both online and offline methods are jointly evaluated on a single benchmark [13]. Typically, the underlying datasets comprise largely static scenes with various angles of view and at high frame rates showing a single object category. Most offline methods formulate MOT as a graph association problem solved using optimization techniques from graph and network theory, e.g., min-cost flow optimization [14], min-clique graphs [15], lifted multicuts [16], and lifted disjoint paths [17]. Exploiting deep learning, several methods investigate either pair-wise appearance similarities [15] or specifically focus on learning the data association task via end-to-end backpropagation [14]. The advent of graph neural networks (GNNs) further allows learning higher-order similarities on graph structures. Along the same line of research, the offline tracker NMPTrack [5] proposes neural message passing to effectively represent both past and future of each detection by leveraging a time-aware prior. Inspired by this idea, we also exploit future information via a time-aware prior.

*3D Multi-Object Tracking*: Compared to popular 2D datasets, available 3D MOT datasets are more challenging since they involve intricate sensor motion and significantly smaller frame rates [6], [7], [13]. On the other hand, 3D instance detection at varying depth levels allows to effectively resolve occlusion.

Conventional 3D tracking-by-detection approaches mostly rely on bounding box information, following Bewley *et al.* [18] in using a Kalman filter as a motion model and the Hungarian algorithm for bipartite data association. While Weng *et al.* [4] use 3D-IoU as the matching criterion, Chiu *et al.* [19] employ the Mahalanobis distance, estimate the initial noise and state covariance of the Kalman filter from the training set, and choose a greedy algorithm for data association. Different from these approaches, CenterPoint [3] is a 3D object detection model that utilizes a keypoint detector to first predict object centers and then perform regression of object attributes, e.g., dimension, orientation, and velocity. Additionally, CenterPoint proposes 3D tracking based on the closest-point matching of object velocity vectors. Combining both online and offline paradigms, FG-3DMOT [20] casts the tracking problem as a factor graph over 3D object detections represented as a Gaussian mixture model in order to find probabilistic trajectory assignments. Other models incorporate 2D object detections as they are less prone to occlusions than their 3D counterparts [11], [21].

In addition to bounding box information, multiple works include 2D/3D appearance features to substantiate pair-wise affinity representation [22]. Deep learning allows to learn semantic features via encoding: Popular methods [12], [21] use image classification networks as encoders for representing image data and PointNet [23] architectures for learning point cloud features. Regarding the inclusion of appearance features, GNN3DMOT [21] is the work most similar to ours. It concatenates encoded modality features before regressing an affinity matrix used for bipartite matching. Recent state-of-the-art approaches [21], [24] facilitate graph neural networks to capture higher-order artefacts on graph structures. OGR3MOT [24] follows NMPTrack [5] in using neural message passing but solves the online 3DMOT problem while leveraging Kalman state predictions for improved track representations.

Although the aforementioned 3D tracking methods show remarkable results in the online setting, they are insufficient in the scenario of offline 3D MOT due to moderate false positive handling. The only two methods solving 3D offline tracking [12], [20] do not provide publicly available implementations, nonetheless we show a comparison on the KITTI benchmark dataset [7]. Different to these aforementioned offline approaches we utilize a graph neural network to learn the tracking task using higher-order node similarities. Our approach differs from NMPTrack [5] by introducing a novel modality and node representation scheme relevant for 3D tracking and a novel agglomerative trajectory clustering scheme that yields high recall and fewer false positives. Different from OGR3MOT [24], we include multiple sensor modalities and model trajectories based on object similarity instead of exploiting Kalman filters for predictive track representation in online tracking.

## III. Technical Approach

Following the tracking-by-detection paradigm, we turn a set of detections per frame $O_t = \{o_1, ..., o_n\}$ into nodes on a directed acyclic graph $G = (V, E)$ that holds an ordered set of frames. The graph consists of a set of nodes $j \in V$ that are connected via directed edges $E \subseteq \{(j, i) \mid (j, i) \in V^2 \text{ and } j \neq i\}$, where edges are directed in a forward-time manner. Instead of using detection edges, we follow the approach of Brasó *et al.* [5] in collapsing them. As a consequence, nodes always reside in a specific frame and edges

only connect nodes at different timestamps while satisfying $t_j < t_i$. Tracking multiple objects in the offline setting entails finding a set of edge-disjoint trajectories $\mathcal{T} = \{T_1, \ldots, T_m\}$ that represents the most plausible association of detections over time. Since our approach involves learning on graph-structured data, both nodes and edges are attributed. We refer to the node feature matrix as $\mathbf{X} = [\mathbf{h}_1, ..., \mathbf{h}_N]^T \in \mathbb{R}^{N \times D}$ where $h_i \in \mathbb{R}^D$ represents a single node feature. Similarly, we denote the edge features $\mathbf{X}_e = [..., \mathbf{h}_{ji}, ...]^T \in \mathbb{R}^{|E| \times D_e}$, where $\mathbf{h}_{ji}$ is the edge feature associated with edge $(j, i)$.

### A. Feature Representation

In typical tracking scenarios such as autonomous driving, we are confronted with a multitude of sensor modalities such as camera, LiDAR, radar, or even thermal images. While the detections are often derived only from a single sensor modality such as LiDAR or camera, the entirety of modalities can still be utilized for improved similarity finding of detections in the tracking task. Our approach fuses 3D pose & motion features (3D-PM) from bounding boxes with 2D as well as 3D appearance features from (surround) cameras (2D-A), LiDAR (3D-AL) as well as radar sensors (3D-R). Different from tracking in the image plane, 3D bounding box information essentially represents a more discriminative feature in 3D tracking due to available depth information [21]. Most importantly, this simplifies re-association after false negatives (FN) generated by occlusions or missed detections but also eases the identification of false positives (FP). Instead of solely exploiting bounding box information in terms of relative node differences for an initial edge feature [5], the 3D-PM feature constitutes the primary node feature in the proposed approach.

*1) 3D Pose and Motion Feature:* We turn each 3D bounding box in the set of detections into an explicit 3D-PM feature without further encoding:

$$\mathbf{h}_{\text{PM},i} = [x, y, z, w, l, h, \gamma, v_x, v_y, \boldsymbol{c}, \mathcal{S}, t]^T \in \mathbb{R}^{11+C}, \quad (1)$$

where $x, y, z$ denotes the 3D object center position in ego-vehicle coordinates. The 3D bounding box dimensions are given by $w, l, h$, while the box orientation is expressed by the yaw angle $\gamma$ about the positive z-axis w.r.t. the ego-vehicle frame. Similarly, $v_x, v_y$ describe the relative object center velocity in the x-y-plane. In addition, a one-hot class vector $\mathbf{c}$ over $C$ classes is appended to encode semantic categories. The detection confidence score $\mathcal{S} \in [0, 1]$ provides an additional means to differentiate between plausible and implausible detections. Finally, a relative timestamp is included. We choose ego-vehicle coordinates over global map coordinates to increase generalization performance.

*2) 2D Appearance Features:* Each detection generates an appearance that is potentially observed on camera. For each detection, the 3D bounding box corners are projected into the image plane and a convex hull of that set is computed. A hull-enclosing rectangle defines the image patch and the respective camera showing most of the object is selected. Thus, the approach includes object backgrounds under the assumption that in-between frames the background stays approximately constant. A fully-convolutional auto-encoder architecture utilizing residual skip connections is employed to learn image features $\mathbf{h}_{\text{2D-A},j}$ as latent space representations. In the case of occlusions, we still use that respective appearance feature and overcome this issue using higher-order similarity finding through the chosen GNN architecture.

*3) 3D Appearance Features:* In order to include 3D shape information, the sparse LiDAR point cloud within and in close proximity to the objects' 3D bounding box is extracted while neglecting the points' reflectance value. In order to account for pose estimation errors a slightly enlarged cuboid is used to associate LiDAR points to the respective object. The masked point cloud is encoded using a PointNet-architecture [23] that is trained towards predicting object categories. This is motivated by prior works that showed that PointNet works well on segmented point clouds [21], [23]. A higher-dimensional feature $\mathbf{h}_{\text{3D-A},j}$ (128-dim.) is taken as the 3D-AL feature used for tracking.

*4) Radar Features:* In addition to LiDAR measurements, radar detections can be used for two reasons: Firstly, they provide a highly accurate radial velocity measurement between the particular sensor and the object (not the actual velocity) and secondly, they provide measurements of objects in large distances, which are captured imperfectly with cameras or sparse LiDAR readings. The measured radial velocity $v_r$ is split into two orthogonal components $(v_x, v_y)$ represented in the ego-vehicle frame that are each compensated by the ego-vehicle motion [6]. The raw set of radar reflections is clustered and the height coordinate is neglected because the radar's longitudinal wave characteristic renders the height coordinate not decisive and erroneous more often than not. We arrive at a radar parametrization $r_P = (x, y, v_x, v_y)$, where $x, y$ is the 2D object position after transformation from the radar coordinate frame into ego-vehicle coordinates. Since each radar detection does not hold a height coordinate we perform 2D pillar expansion [25] and associate radar detections to objects as soon as the enlarged objects' cuboid and the pillar intersect. Since the chosen pillar representation does not represent an element of a Euclidean group as in the LiDAR case, we follow a naive approach and remove all coordinate-sensitive transforms in the PointNet architecture and merely transform each object's feature that consists of multiple radar detections in a permutation-invariant manner to arrive at the radar appearance feature $\mathbf{h}_{\text{R},j}$.

### B. Graph Construction

The chosen approach arranges nodes on a tracking graph over 5 frames in a sliding window manner with a stride of 1. The tracking performance improves drastically when learning the tracking task with actual detections instead of ground truth annotations since FP filtering and FN handling pose major challenges in real-world tracking. Consequently, ground truth annotation identifiers need to be paired with actual detections in order to construct edge labels for the learning stage. Under the assumption that ground truth annotations generally do not show significant intra-category overlap, we match detection results to geometrically close annotations in the birds-eye view (BEV).

*1) Initial Node and Edge Embeddings:* The initial node features only consists of the 3D-PM feature:

$$\mathbf{h}_i = [\mathbf{h}_{\text{PM},i}]^T \in \mathbb{R}^{11+C+96}. \quad (2)$$

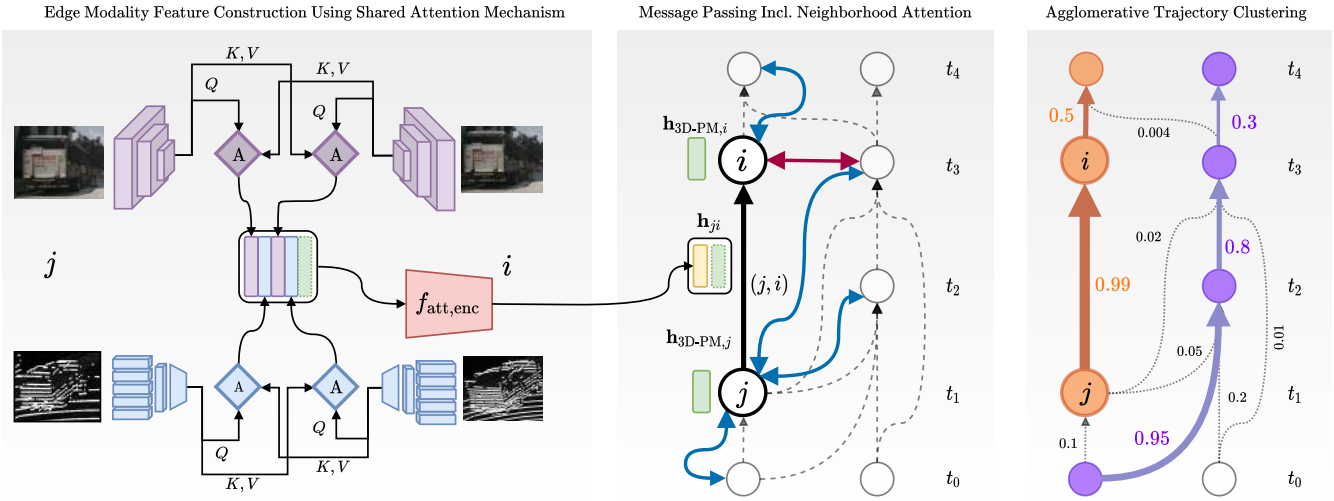



Fig. 2: Overview of our Batch3DMOT architecture. A cross-edge modality attention mechanism fuses the features of the involved objects to construct an edge feature (left). Message passing including inter-category neighborhood attention propagates information. Blue arrows denote time-aware message passing, and red arrows denote frame-wise information propagation (middle). Predicted edge scores are turned into trajectory hypotheses using agglomerative trajectory clustering (right).

Throughout this work, we found that it is more beneficial to add modality such as $\mathbf{h}_{\text{2D-A},i}$ during the message passing stage instead of as an initial feature as shown in Sec. IV. The edge features are defined as

$$\mathbf{h}_{ji} = [\Delta x_{ji}, \Delta v_{ji}, \Delta\gamma_{ji}, \Delta s_{ji}, \Delta t_{ji}]^T \in \mathbb{R}^5, \quad (3)$$

where $\Delta x_{ji}$ denotes Euclidean distance between the object centers and $\Delta v_{ji}$ the L2 norm of both velocity vectors. The smallest signed yaw difference of the two detections is given by $\Delta\gamma_{ji}$ while $\Delta s_{ji}$ is the log-volume-ratio and $\Delta t_{ji}$ the time difference.

*2) Graph Connectivity:* When investigating the effect of graph connectivity on the tracking result we found that it is beneficial to only connect nodes of the same object category rather than utilizing inter-category edges that could potentially overcome class prediction errors from the object detection task. This essentially renders the problem a disconnected graph with multiple components, which generally limits information propagation when learning. While the 2D MOT task is mostly focused on one object category, the 3D MOT task faces both the curse of dimensionality leading to a high number of detections per frame and multiple object categories. As a consequence, we found that limiting the number of possible edges represents an essential prior to the learning problem. Based on the normalized kinematic similarity metric

$$v^*_{ji} = \frac{\frac{1}{2}\Delta x^*_{ji} + \frac{1}{4}\Delta\gamma^*_{ji} + \frac{1}{4}\Delta v^*_{ji}}{\max_q\{\frac{1}{2}\Delta x^*_{qi} + \frac{1}{4}\Delta\gamma^*_{qi} + \frac{1}{4}\Delta v^*_{qi} \mid \forall k : t_q < t_i\}}, \quad (4)$$

the k-nearest neighbors in the past of every node $i$ are selected for edge construction, which essentially extracts a promising corridor based on similar position, velocity vectors, and yaw angles, while $\Delta x^*_{qi}, \Delta\gamma^*_{qi}, \Delta v^*_{qi}$ itself represent neighborhood-normalized distances. In the following, directed edges are constructed pointing from each of the $k$ neighbors to node $i$. Regarding the following graph learning step, edge labels denoting the active/inactive edges are necessary. We examine whether two nodes hold the same instance identifier and only connect them as an active edge if they represent the closest time-wise occurrence of the same object instance. Otherwise, edges hold labels of value 0.

### C. Message Passing Graph Neural Network

This work employs the principle of time-aware neural message passing [5], which is extended to allow information exchange between inter-category nodes that reside in particular disconnected graph components. In addition, we present a novel way to include intermittent sensor modalities across edges.

*1) Cross-Edge Modality Attention:* Initial node and edge features are encoded to produce approximately evenly-sized node and edge embeddings

$$f^v_{enc}(\mathbf{X}) = \mathbf{H}^{(0)}_v, \quad f^e_{enc}(\mathbf{X}_e) = \mathbf{H}^{(0)}_e, \quad (5)$$

where the two networks take the form of MLPs. In the case of additional sensor modalities, the edge feature is augmented using modality cross-attention between the respective nodes' features to which edge (j,i) is incident to. Thus, each nodes' feature is attending and is being attended in order to find an agreement on whether to utilize the respective modality during the edge feature update. Based on that, we define the following queries $\mathbf{Q}$, keys $\mathbf{K}$ and values $\mathbf{V}$ for both attention directions:

$$\mathbf{Q}_{ij} = \mathbf{X}_{\text{sens},i} \quad \mathbf{K}_{ij} = \mathbf{X}_{\text{sens},j} \quad \mathbf{V}_{ij} = \mathbf{X}_{\text{sens},j} \quad (6)$$

$$\mathbf{Q}_{ji} = \mathbf{X}_{\text{sens},j} \quad \mathbf{K}_{ji} = \mathbf{X}_{\text{sens},i} \quad \mathbf{V}_{ji} = \mathbf{X}_{\text{sens},i}, \quad (7)$$

where $\mathbf{X}_{\text{sens},j}$ could represent either a LiDAR $\mathbf{h}_{\text{3D-A},j}$ or radar feature $\mathbf{h}_{\text{R},j}$ of the respective node. We use a standard multi-head attention mechanism per modality in order to compute attention-weighted features using head-specific linear transforms $(\mathbf{W}^Q_u, \mathbf{W}^K_u, \mathbf{W}^V_u)$ to attend to multiple regions with the respective modality feature:

$$\text{MultiHead}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{Concat}(head_1, ...head_h)\mathbf{W}^O, \quad (8)$$

$$\text{where } \text{head}_u = \text{Softmax}(\frac{\mathbf{Q}\mathbf{W}^Q_u)(\mathbf{K}\mathbf{W}^K_u)^T}{\sqrt{d_k}})\mathbf{V}\mathbf{W}^V_u. \quad (9)$$

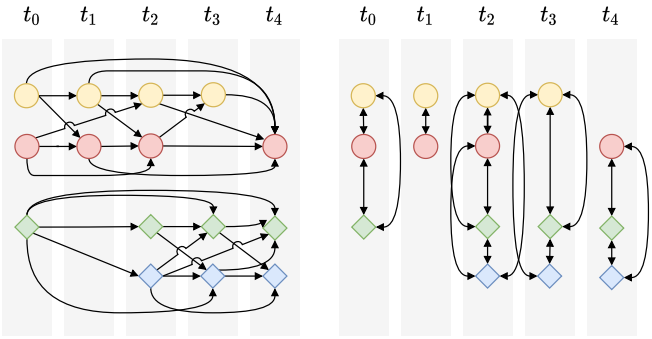



Fig. 3: Both graph connectivity cases: Time-aware message passing operates on a time-directed acyclic tracking graph that holds disconnected semantic components (left) while attention-weighted neighborhood convolution is performed on temporary frame-wise k-NN graphs (right).

The attended modality features are then concatenated and encoded as depicted in Fig. 2. In the case of both LiDAR and camera we arrive at:

$$\mathbf{H}^{*}_{e,\text{att}} = f_{\text{att,enc}}\left(\left[\mathbf{X}^{*}_{\text{3D-A},i}, \mathbf{X}^{*}_{\text{2D-A},i}, \mathbf{X}^{*}_{\text{3D-A},j}, \mathbf{X}^{*}_{\text{2D-A},j}, \mathbf{X}_{e}\right]\right), \tag{10}$$

which constitutes the attention-weighted modality edge feature used during the edge update step in message passing that is briefly covered in the following.

*2) Time-Aware Message Passing Using Inter-Category Graph Attention:* A single message passing layer consists of an edge feature update based on the neighboring node features $\mathbf{h}_i^{(l-1)}$, $\mathbf{h}_j^{(l-1)}$ and the current edge feature $\mathbf{h}_{ji}^{(l-1)}$. In addition, our approach involves the multi-modal attention-weighted similarity feature $\mathbf{h}_{ji,\text{att}}^{(0)}$ , which is fed in each iteration to substantiate the update based on appearance similar modality features:

$$\mathbf{h}_{ji}^{(l)} = f_e\left(\left[\mathbf{h}_i^{(l-1)}, \mathbf{h}_j^{(l-1)}, \mathbf{h}_{ji}^{(l-1)}, \mathbf{h}_{ji,\text{att}}\right]\right), \tag{11}$$

where $[\cdot,\cdot,\cdot]$ represents the concatenation of the four representations as an input to a ReLU-activated MLP $f_e$. With respect to updating node features, messages $\mathbf{m}_{ij}^{(l)}$ are crafted based on either neighbors in the past $\mathcal{N}_{past}(j)$ or neighbors in the future $\mathcal{N}_{fut}(j)$ of a node $j$. In the next step, all messages from the future and from the past neighborhood of a node, respectively, are aggregated using a permutation-invariant sum, which results in node-specific past and future features:

$$\mathbf{h}_{j,past}^{(l)} = \sum_{i\in\mathcal{N}_{past}(j)} \underbrace{f_v^{past}\left(\left[\mathbf{h}_i^{(l-1)}, \mathbf{h}_{ji}^{(l)}, \mathbf{h}_i^{(0)}\right]\right)}_{\mathbf{m}_{ij}^{(l)}}, \tag{12}$$

$$\mathbf{h}_{j,fut}^{(l)} = \sum_{i\in\mathcal{N}_{fut}(j)} \underbrace{f_v^{fut}\left(\left[\mathbf{h}_i^{(l-1)}, \mathbf{h}_{ji}^{(l)}, \mathbf{h}_i^{(0)}\right]\right)}_{\mathbf{m}_{ji}^{(l)}}. \tag{13}$$

The functions $f_v^{fut}$ and $f_v^{fut}$ again take the form of MLPs and transform the recently updated edge feature $\mathbf{h}_{ji}^{(l)}$, the initial node feature $\mathbf{h}_i^{(0)}$ as well as the current node representation $\mathbf{h}_i^{(l-1)}$. The final node update is reached by combining the past and future feature and feeding it to a function $f_v$:

$$\mathbf{h}_j^{(l)} = f_v\left(\left[\mathbf{h}_{j,past}^{(l)}, \mathbf{h}_{j,fut}^{(l)}\right]\right), \tag{14}$$

which again takes the form of a ReLU-activated MLP.

In order to enable inter-category information exchange in between message passing steps, we construct temporary frame-wise graphs for which each node $j$ is connected to its top-k neighbors (without self-loops) having identical timestamps regardless of object category. Then, a graph attention layer (GAT) [26] propagates node features in an attention-weighted manner to produce linear combinations of neighbor nodes:

$$\mathbf{h}'_j = \alpha_{ii}\mathbf{\Theta}\mathbf{h}_j^{(l)} + \sum_{i\in\mathcal{N}_t(j)} \alpha_{ji}\mathbf{\Theta}\mathbf{h}_i^{(l)}, \tag{15}$$

while $i$ and $j$ in this case do not represent nodes in different frames but in an identical one ($t_i = t_j$). Attention weights among the nodes per frame are used to elect nodes that are of relevance to one another such as overlapping detections of different semantic categories, which would normally reside in two different disconnected graph components. The attention weights

$$\alpha_{ji} = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{a}^T\left[\mathbf{W}\mathbf{h}_i, \mathbf{W}\mathbf{h}_j\right]\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\text{LeakyReLU}\left(\mathbf{a}^T\left[\mathbf{W}\mathbf{h}_i, \mathbf{W}\mathbf{h}_k\right]\right)\right)} \tag{16}$$

are softmax-normalized across neighborhoods while the attention-mechanism consists of a single-layer feedforward neural network represented by a weight vector $\mathbf{a}$. A final edge classifier MLP $f_e^{reg}$ downprojects each edge feature to a single Sigmoid-activated scalar that denotes the edge activation score.

*3) Loss Formulation:* As the approach considers multiple semantic categories that exhibit different frequencies of occurrence, it faces significant category imbalance with respect to the number of nodes per category. This directly translates to an even more unbalanced category-specific number of edges contained in the graph. Having chosen a disconnected graph, the nodes incident to a particular edge are always of the same category. Based on that, we employ a class-balanced loss formulation that takes a binary cross-entropy and weights edges based on category frequencies:

$$\mathcal{L}_{\text{CB}} = \frac{1}{|E|}\sum_{(j,i)\in E}\frac{1-\beta}{1-\beta^{n_{ji}}} y_{ji}\log(p_\phi^{ji}) + (1-y_{ji})\log(1-p_\phi^{ji}),$$

where $\beta$ represents a hyperparameter and $n_{ji}$ is the absolute number of objects with respect to the node categories involved per edge. The respective weights are estimated based on object category frequencies in the training set [27].

### D. Inference and Graph Traversal

The outputs of the GNN architecture are Sigmoid-valued scores that represent whether an edge is likely to be active/inactive. Instead of thresholding at an edge score of 0.5 to find active/inactive edges to turn into trajectories, we follow the spirit of *ByteTrack* [8] and try to associate (nearly) *every* detection with a preliminary trajectory. Based on the assumption that the predicted edge scores show some inherent order, i.e., FP edges exhibit lower scores than TP edges within local neighborhoods of the graph, we propose a score-based agglomerative trajectory clustering paradigm (Algorithm 1). The edge score predictions of multiple overlapping batches are averaged per edge. All edges are arranged in descending order and empty (ordered) clusters are initialized that will later hold output trajectories. In the following, we loop through all edges from the highest to lowest score and check whether the edge

**Algorithm 1:** Agglomerative Trajectory Clustering.

$E_{pred}, N_{meta} \leftarrow$ CombineBatches($GNN(\mathbf{X}, \mathbf{X}_e)$)
$E^*_{pred} \leftarrow$ DescSortEdgesByScore($e_{pred}$)
$vis \leftarrow$ CreateVisitedNodesDict()
$\mathcal{C} \leftarrow$ CreateEmptyClustersDict()
**for** $e_{ji}, score\ in\ E^*_{pred}$ **do**
  **if** $j \notin vis$ *and* $i \notin vis$ **then**
    $\mathcal{C} \leftarrow$ CreateNewCluster($e_{ji}$)
    UpdateVisitedNodes($e_{ji}, \mathcal{C}$)
  **else**
    **if** $j \notin vis$ *and* $i \in vis$ **then**
      **if** *$i$ is leading node in $\mathcal{C}(i)$* **then**
        $\mathcal{C} \leftarrow$ AddToCluster($e_{ji}$)
        $vis \leftarrow$ UpdateVisitedNodes($e_{ji}, \mathcal{C}$)
    **else if** $j \in vis$ *and* $i \notin vis$ **then**
      **if** *$j$ is trailing node in $\mathcal{C}(j)$* **then**
        $\mathcal{C} \leftarrow$ AddToCluster($e_{ji}$)
        $vis \leftarrow$ UpdateVisitedNodes($e_{ji}, \mathcal{C}$)
    **else if** $j \in vis$ *and* $i \in vis$ **then**
      **if** *$j$ is trailing $\mathcal{C}(j)$ and $i$ is leading $\mathcal{C}(i)$* **then**
        $\mathcal{C} \leftarrow$ JoinClusters($e_{ji}$)
        $vis \leftarrow$ UpdateVisitedNodes($e_{ji}, \mathcal{C}$)
**return** TurnClustersIntoTrajectories($\mathcal{C}$)

is constrained or unconstrained. If constrained, it is checked whether the edge would essentially add time-wise leading or trailing nodes to one of the temporary clusters or if it joins two clusters. In the case of joining two clusters, an additional score-wise threshold needs to be met. Otherwise, the edge does not violate any tracking constraints and a new cluster is initialized.

## IV. Experimental Evaluation

In this section, we present quantitative and qualitative evaluations of our proposed Batch3DMOT on the nuScenes [6] and KITTI [7] datasets using the average multiple-object tracking accuracy (AMOTA) and multiple-object tracking accuracy (MOTA) metrics, respectively. Similar to existing methods, we evaluate our model on the nuScenes test set as well as the KITTI 2D MOT benchmark. We provide additional experimental data in the supplementary material.

*Detections and GT Matching*: In this approach, we use the detections provided by MEGVII [28] and CenterPoint [3] for nuScenes. On the KITTI dataset, we use Point-RCNN detections [29] as also used by FG3DMOT [20] and AB3DMOT [4]. We match detections to ground truth trajectory labels to obtain identifiers. As proposed earlier [6], [28], the L2 center distance is often used for matching, which is beneficial for faraway objects. Our empirical findings show that especially large objects suffer from this heuristic since, e.g., their respective length is not predicted correctly, which leads to considerable object center translations and effectively renders the L2 distance uninformative. Therefore, we follow a bi-level approach by first selecting a close radius (L2) and then checking whether detection and annotation exhibit a significant BEV-IoU overlap.

*Implementation Details*: Each batch consists of five frames where each node is connected to its 40-nearest neighbors in the past. Object 2D-A features are scaled to a $32 \times 32$-dimensional RGB image. The fully-convolutional image encoder is built upon the ResNet architecture and is trained for 80 epochs using a learning rate (LR) of $0.002$ and a batch size of 32. LiDAR point clouds are aggregated over multiple frames, normalized, centered, and rotationally permuted to mimic orientation errors in 3D object detection. In order to generate a 3D-A feature, a minimum of five LiDAR points must exist. Otherwise, the object does not hold a 3D-A feature. The LiDAR PointNet is trained for 500 epochs using a batch size of 64 and a LR of $0.001$. We do not employ pre-training on other datasets contrary to previous findings [5] since this decreased model performance. In the case of radar, each object must hold at least two radar detections to generate a feature. The radar network is trained for 1000 epochs using a batch size of 256 and a LR of 0.0002. The GNN models are trained for 100 epochs using a batch size of two (10 frames in total) and LRs between $4e^{-5}$ and $1e^{-4}$, which largely depends on the detections used and the number of edges contained in the graph. We perform 6 message passing steps with intermediate frame-wise neighborhood convolutions (20 k-NN), which we ablate in the experiments presented in the supplementary material in Sec. S.2.C. We estimate the class-balancing factors based on the absolute frequencies of ground truth annotations in the training set and find the hyperparameter $\beta = 0.8$ empirically. While the modality attention modules use two attention heads, it proved to be sufficient to use a single head in the frame-wise neighborhood attention mechanism. Our evaluations show that training feature encoders and the GNN model in an end-to-end manner or using transfer learning decreases the performance.

TABLE I: Comparison of AMOTA scores on the nuScenes validation set. Bold/underlined numbers denote **best**/second best model scores, respectively.

| Method | Overall | Bicyc. | Bus | Car | Moto. | Ped. | Trailer | Truck |
|---|---|---|---|---|---|---|---|---|
| *AB3DMOT [4] [28]* | 0.179 | 0.09 | 0.489 | 0.36 | 0.051 | 0.091 | 0.111 | 0.142 |
| *Prob3DMOT [19] [28]* | 0.561 | 0.272 | 0.741 | 0.735 | 0.506 | 0.755 | 0.337 | 0.580 |
| CenterPoint [3] | 0.665 | 0.458 | 0.801 | 0.842 | 0.615 | 0.777 | 0.504 | 0.656 |
| ProbMM-3DMOT [22] | 0.687 | 0.490 | 0.820 | 0.843 | 0.702 | 0.766 | **0.534** | 0.654 |
| 3D-PM-MEGVII [28] | 0.623 | 0.368 | 0.759 | 0.789 | 0.655 | 0.796 | 0.378 | 0.617 |
| 3D-PM-CP [3] | 0.708 | 0.540 | 0.837 | 0.849 | 0.728 | 0.813 | 0.497 | 0.689 |
| 3D-PM-C-CP [3] | 0.709 | 0.542 | 0.837 | **0.851** | 0.733 | 0.813 | 0.502 | 0.688 |
| 3D-PM-CL-CP [3] | **0.715** | 0.540 | **0.855** | **0.851** | **0.748** | **0.821** | 0.493 | 0.695 |
| 3D-PM-CLR-CP [3] | 0.713 | **0.545** | 0.851 | 0.850 | 0.736 | 0.820 | 0.494 | **0.696** |

### A. Quantitative Results and Ablation Study

We report our results on the nuScenes validation set in Tab. II. Additionally, we also present the category-specific AMOTA scores in Tab. I. While there is no existing offline method on the nuScenes benchmark, we compare against a strong set of state-of-the-art online trackers. The main baselines include AB3DMOT [4], Prob3DMOT [19], two online Kalman filter-based methods, and CenterPoint [3] which performs closest-point matching of predicted velocity vectors. These methods currently represent the naive choice when generating pseudo ground truth due to their inherent simplicity and robustness as well as high tracking accuracy in terms of the AMOTA score. Nonetheless, we argue that there is room for improvement by examining multiple frames in a batch-manner. In our case, we chose a batch length of 5 frames for three reasons: 1) Typical birth and death memory matching time thresholds used in bipartite association [4] are in a similar range. 2) We expect an object to reappear after 2-3 frames of false negative detections while neglecting long-term occlusions. 3) With an increasing number of objects per batch, the number of edges increases exponentially, which makes the graph learning problem significantly more complex. Based on these factors,

TABLE II: Ablation study on the nuScenes validation set. All results shown are derived using CenterPoint detections [3].

| Method | PM | C | L | R | AMOTA↑ | AMOTP↓ | MOTA↑ | Recall↑ | FP↓ | FN ↓ | IDS ↓ | FRAG ↓ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CenterPoint [3] | ✓ | | | | 0.664 | **0.567** | 0.562 | 0.698 | 13187 | 20446 | 562 | 424 |
| OGR3MOT [24] | ✓ | | | | 0.693 | 0.627 | 0.602 | – | – | – | **262** | **332** |
| w/o MP layers | ✓ | | | | 0.519 | 0.960 | 0.471 | 0.592 | **7206** | 33801 | 7065 | 2648 |
| 60 k-NN | ✓ | | | | 0.578 | 0.728 | 0.493 | 0.633 | 12159 | 27187 | 4497 | 1646 |
| Connected graph comp. | ✓ | ✓ | ✓ | | 0.646 | 0.842 | 0.599 | 0.702 | 7621 | 23011 | 1233 | 761 |
| TA-NMP [5] | ✓ | ✓ | | | 0.668 | 0.698 | 0.589 | 0.714 | 11106 | 21806 | 1810 | 769 |
| w/o Aggl. Traj. Clust. | ✓ | | | | 0.683 | 0.682 | 0.592 | 0.699 | 11030 | 20260 | 1271 | 434 |
| Stacked modalities | ✓ | ✓ | ✓ | | 0.689 | 0.678 | 0.602 | 0.688 | 9525 | 20954 | 938 | 536 |
| w/o 2D-A attention | ✓ | ✓ | | | 0.698 | 0.657 | 0.602 | 0.700 | 9951 | 20641 | 886 | 403 |
| w/o CB Loss | ✓ | | | | 0.702 | 0.617 | 0.607 | 0.723 | 11467 | 18516 | 758 | 418 |
| w/o Neigborhood GAT | ✓ | | | | 0.703 | 0.644 | 0.604 | 0.716 | 11465 | 18691 | 767 | 416 |
| Batch3DMOT-3D-PM | ✓ | | | | 0.708 | 0.630 | **0.612** | 0.719 | 11102 | 18640 | 688 | 383 |
| Batch3DMOT-3D-PM-C | ✓ | ✓ | | | 0.709 | 0.622 | 0.608 | 0.716 | 11307 | 18722 | 664 | 375 |
| **Batch3DMOT-3D-PM-CL** | ✓ | ✓ | ✓ | | **0.715** | 0.598 | **0.612** | **0.726** | 11175 | **18494** | 598 | 357 |
| Batch3DMOT-3D-PM-CLR | ✓ | ✓ | ✓ | ✓ | 0.713 | 0.592 | 0.611 | **0.726** | 11196 | 18520 | 622 | 385 |

(a) Ground Truth Trajectories (b) Kalman Filter-Based Tracking [22] (c) Batch3DMOT (Ours)

Fig. 4: Comparison of FP filtering on the nuScenes validation set. The Kalman filter-based tracking approach [22] shows mediocre FP filtering, while our proposed Batch3DMOT using only 3D-PM features for pseudo ground truth generation shows superior FP filtering.

we see the grounds for comparison with the chosen baselines. We ablate on the number of frames to consider in Sec. S.2.A. of the supplementary material.

Tab. II shows that an increase from 40 to 60 k-nearest neighbors per node results in a stark decrease in tracking accuracy (-12.8%). The best choice of $k$ can only be found empirically, however $10e^3$ serves as a suitable maximum number of edges per batch. Fig. S.2 (b) in the supplementary material presents the tracking performance for 10 and 20 nearest neighbors. Independently, we observe a considerable decrease in tracking performance when connecting nodes of different semantic categories (-6%). Phrasing the offline 2D MOT GNN by Braso *et al.* [5] as an offline 3D MOT method provides an additional baseline. It uses identical edge features but the 2D-A feature as the sole node feature, which performs worse than our architecture, but achieves similar recalls. Moreover, reducing our model to a node similarity network (no message passing) results in an AMOTA score of 0.519. Furthermore, we observe a slight decrease in AMOTA when the frame-wise neighborhood GAT aggregations are removed (-0.5% wrt. best performing model). Modality intermittence is especially severe when stacking all the modalities as a node feature (Tab. II), which ultimately motivated our modality attention mechanism. Furthermore, we also observe that the class-balancing scheme slightly enhances the tracking result. Finally, we test-wise replace our agglomerative trajectory clustering with a bidirectional depth-first-search algorithm that iterates from high score to low score edges, which performs worse than our proposed agglomerative clustering paradigm. We provide a more extensive parameter study in Sec. S.2 of the supplementary material.

We observe the highest AMOTA tracking score for the 3D-PM-CL model on the validation set (see Tab. II), which demonstrates the efficacy of the proposed modality attention module (see Tab. II). Including 2D-A features leads to a small performance increase compared to the pose-only variant (3D-PM). Here, occlusions are the limiting factor that restricts a larger improvement in performance, which is supported by a large accuracy increase when including LiDAR. The PointNet architecture succeeds effectively at *extracting* local object information. The 3D-PM-CLR architecture exhibits a slight accuracy decrease compared to the CL-counterpart, which can be attributed to the severe sparsity and quality of the radar detections in nuScenes. Additionally, we present the category-specific AMOTA scores in Tab. I. On the nuScenes test set, the 3D-PM-CL model outperforms the pose-and-motion variant (Tab. III). Batch3DMOT achieves an AMOTA score of 0.689, which outperforms the state-of-the-art online method OGR3MOT [24] by 3.3% using the same detections. Note that EagerMOT uses both 2D and 3D detections which the other methods do not. We also report the performance on the KITTI test set for the car category in Tab. IV and observe that our model achieves competitive results as FG-3DMOT [20], while using only 5 frames.

### B. Qualitative Results

Fig. 4 illustrates the accumulation over 40 frames of a scene on the nuScenes dataset. We observe that our proposed Batch3DMOT method removes a large number of detections that essentially represent FPs when compared with the trajectory

TABLE III: Comparison of the 3D MOT performance on the nuScenes test set evaluated on the official benchmarking server.

| Method | Detections | PM | C | L | AMOTA↑ | AMOTP↓ | MOTA↑ | Recall↑ | FP↓ | FN↓ | IDS↓ | FRAG↓ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *AB3DMOT [4]* | *MEGVII [28]* | ✓ | | | 0.151 | 1.501 | 0.154 | 0.276 | 15088 | 75730 | 9027 | 2557 |
| *Prob3DMOT [19]* | *MEGVII [28]* | ✓ | | | 0.550 | 0.798 | 0.459 | 0.600 | 17533 | 33216 | 950 | 776 |
| ProbMM3DMOT [22] | CP [3] | ✓ | ✓ | | 0.655 | 0.617 | 0.555 | **0.707** | 18061 | 23323 | 1043 | 717 |
| CenterPoint [3] | CP [3] | ✓ | | | 0.650 | **0.535** | 0.536 | 0.680 | 17355 | 24557 | 684 | 553 |
| OGR3MOT [24] | CP [3] | ✓ | | | 0.656 | 0.620 | 0.554 | 0.692 | 17877 | 24013 | **288** | **371** |
| *EagerMOT* [11] | *CP [3] + Cascade RCNN* | ✓ | ✓ | | 0.677 | 0.550 | 0.568 | 0.727 | 17705 | 24925 | 1156 | 601 |
| Batch3DMOT (Ours) | CP [3] | ✓ | | | 0.683 | 0.633 | 0.568 | 0.679 | **15290** | 22692 | 994 | 562 |
| Batch3DMOT (Ours) | CP [3] | ✓ | ✓ | ✓ | **0.689** | 0.604 | **0.570** | 0.704 | 15580 | **22353** | 718 | 427 |

We do not highlight *methods that use different sets of detections [4], [11], [19]* but still report them in this table for completeness.

TABLE IV: Comparison of the 2D MOT performance on the KITTI test set.

| Method | MOTA↑ | MOTP↑ | MT↑ | ML↓ | IDS↓ | FRAG↓ |
|---|---|---|---|---|---|---|
| *DSM [12]* | 0.762 | 0.834 | 0.600 | 0.831 | 296 | 868 |
| AB3DMOT [4] | 0.838 | 0.853 | 0.669 | 0.114 | **9** | 224 |
| FG-3DMOT (online) [20] | 0.837 | 0.846 | 0.680 | 0.099 | **9** | 375 |
| FG-3DMOT (offline) [20] | 0.880 | 0.850 | 0.755 | 0.119 | 20 | 117 |
| Batch3DMOT (5 frames) [29] | **0.886** | **0.868** | **0.767** | **0.088** | 19 | **74** |

ground truth. We also observe that our approach works especially well on still-standing objects, while Prob3DMOT [19] yields a higher number of FPs. Additional insight into low- and high-confidence model predictions is presented in Sec. S.1, Sec. S.3, and Fig. S.3 of the supplementary material. Fig. 4 meets the requirements with respect to generating pseudo-groundtruth. This is further exemplified in Sec. S.1.B of the supplementary material by training on pseudo-labeled test-data and in Sec. S.1.C for training an online 3D Kalman filter from data statistics that include weak pseudo-labeled annotations.

## V. Conclusion

In this work, we proposed a framework for addressing the offline 3D MOT task using a multi-modal graph neural network including a novel agglomerative trajectory construction scheme. We presented extensive results on two challenging datasets demonstrating that our approach achieves state-of-the-art performance. We also showed the benefits of our proposed cross-edge modality attention in mitigating the effect of modality intermittence. Our method was able to improve tracking accuracy compared to current online methods using the same detections and shows enhanced false positive filtering. In future work, we plan to extend our approach to also cope with long-term occlusions.

## References


[1] N. Radwan, W. Burgard, and A. Valada, "Multimodal interaction-aware motion prediction for autonomous street crossing," *The International Journal of Robotics Research*, vol. 39, no. 13, pp. 1567–1598, 2020.

[2] M. Mittal, R. Mohan, W. Burgard, and A. Valada, "Vision-based autonomous uav navigation and landing for urban search and rescue," *arXiv preprint arXiv:1906.01304*, 2019.

[3] T. Yin, X. Zhou, and P. Krahenbuhl, "Center-based 3d object detection and tracking," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, 2021, pp. 11 784–11 793.

[4] X. Weng, J. Wang, D. Held, and K. Kitani, "3d multi-object tracking: A baseline and new evaluation metrics," in *Int. Conf. on Intelligent Robots and Systems*, 2020, pp. 10 359–10 366.

[5] G. Brasó and L. Leal-Taixé, "Learning a neural solver for multiple object tracking," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, 2020, pp. 6246–6256.

[6] W. K. Fong, R. Mohan, H. J. Valeria, L. Zhou, H. Caesar, O. Beijbom, and A. Valada, "Panoptic nuscenes: A large-scale benchmark for lidar panoptic segmentation and tracking," *IEEE Robotics and Automation Letters*, 2022.

[7] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, 2012.

[8] Y. Zhang, P. Sun, Y. Jiang, D. Yu, Z. Yuan, P. Luo, W. Liu, and X. Wang, "Bytetrack: Multi-object tracking by associating every detection box," *arXiv preprint arXiv:2110.06864*, 2021.

[9] J. V. Hurtado, R. Mohan, W. Burgard, and A. Valada, "Mopt: Multi-object panoptic tracking," *arXiv preprint arXiv:2004.08189*, 2020.

[10] F. R. Valverde, J. V. Hurtado, and A. Valada, "There is more than meets the eye: Self-supervised multi-object detection and tracking with sound by distilling multimodal knowledge," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, 2021, pp. 11 612–11 621.

[11] A. Kim, A. Ošep, and L. Leal-Taixé, "Eagermot: 3d multi-object tracking via sensor fusion," *Int. Conf. on Robotics and Automation*, 2021.

[12] D. Frossard and R. Urtasun, "End-to-end learning of multi-sensor 3d tracking by detection," in *Int. Conf. on Robotics and Automation*, 2018.

[13] P. Dendorfer, H. Rezatofighi, A. Milan, J. Shi, D. Cremers, I. Reid, S. Roth, K. Schindler, and L. Leal-Taixé, "Mot20: A benchmark for multi object tracking in crowded scenes," *arXiv preprint:2003.09003*, 2020.

[14] S. Wang and C. C. Fowlkes, "Learning optimal parameters for multi-target tracking with contextual interactions," *Int. Journal of Computer Vision*, vol. 122, no. 3, p. 484–501, 2016.

[15] A. Zamir, A. Dehghan, and M. Shah, "Gmcp-tracker: Global multi-object tracking using generalized minimum clique graphs," *Europ. Conf. on Computer Vision*, 2012.

[16] S. Tang, M. Andriluka, B. Andres, and B. Schiele, "Multiple people tracking by lifted multicut and person re-identification," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, 2017.

[17] A. Hornakova, R. Henschel, B. Rosenhahn, and P. Swoboda, "Lifted disjoint paths with application in multiple object tracking," in *Int. Conf. on Machine Learning*, 2020, pp. 4364–4375.

[18] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," in *Proc.of the IEEE Int. Conf. on Image Processing*, 2016, pp. 3464–3468.

[19] H.-k. Chiu, A. Prioletti, J. Li, and J. Bohg, "Probabilistic 3d multi-object tracking for autonomous driving," *arXiv preprint arXiv:2001.05673*, 2020.

[20] J. Pöschmann, T. Pfeifer, and P. Protzel, "Factor graph based 3d multi-object tracking in point clouds," in *Int. Conf. on Intelligent Robots and Systems*, 2020, pp. 10 343–10 350.

[21] X. Weng, Y. Wang, Y. Man, and K. M. Kitani, "GNN3DMOT: Graph neural network for 3d multi-object tracking with 2d-3d multi-feature learning," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, 2020, pp. 6498–6507.

[22] H. kuang Chiu, J. Li, R. Ambrus, and J. Bohg, "Probabilistic 3d multi-modal, multi-object tracking for autonomous driving," *Int. Conf. on Robotics and Automation*, 2021.

[23] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, 2017, pp. 652–660.

[24] J.-N. Zaech, A. Liniger, D. Dai, M. Danelljan, and L. Van Gool, "Learnable online graph representations for 3d multi-object tracking," *IEEE Robotics and Automation Letters*, pp. 1–1, 2022.

[25] R. Nabati and H. Qi, "Centerfusion: Center-based radar and camera fusion for 3d object detection," in *IEEE Winter Conference on Applications of Computer Vision*, 2021, pp. 1527–1536.

[26] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *Int. Conf. on Learning Representations*, 2018.

[27] Y. Cui, M. Jia, T.-Y. Lin, Y. Song, and S. Belongie, "Class-balanced loss based on effective number of samples," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, 2019, pp. 9268–9277.

[28] B. Zhu, Z. Jiang, X. Zhou, Z. Li, and G. Yu, "Class-balanced grouping and sampling for point cloud 3d object detection," *arXiv preprint arXiv:1908.09492*, 2019.

[29] S. Shi, X. Wang, and H. Li, "PointRCNN: 3d object proposal generation and detection from point cloud," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, 2019, pp. 770–779.

# 3D Multi-Object Tracking Using Graph Neural Networks with Cross-Edge Modality Attention

## - Supplementary Material -

Martin Büchner and Abhinav Valada

In this supplementary material, we (i) portray applications that delineate the usability of generated pseudo-labels, (ii) we present additional ablation experiments on main model parameters and (iii) give qualitative insights into low and high confidence model predictions.

### S.1. Generating Pseudo Trajectory Labels

In this section, we describe different methodologies to demonstrate whether the proposed model can approximately meet demands deriving from reference-data generation.

#### A. Trajectory Postprocessing

Since the proposed model does not include an update step fusing predictions and measurements, the predicted trajectories are estimates based on original detections. In order to yield smoother trajectories serving as labels, we further process them in a series of steps described below:

- Trajectory interpolation regarding missing timesteps.
- Yaw angle projection into $\gamma \in [-\pi, \pi]$ range to prevent further interpolation errors.
- Compute an intra-track BEV-IoU as a measure for still-standing objects (e.g. parking cars), which is computed as the product of all BEV-IoUs of pairs of boxes.
- Yaw correction on still-standing objects that suffer from orientation error of approx. $\pm\pi$: For intra-track BEV-IoUs greater than 0.7, we cluster yaw angles into two regimes (track-wise). All angles contained in the track are overwritten using the mean yaw angle of the majority class.
- Due to the chosen batch-size of 5 trajectories the trajectories suffer from ID switches occasionally, which is convenient to detect for still-standing objects. For pairs of trajectories where each track shows an intra-track BEV-IoU $> 0.7$ we check for a BEV-IoU $> 0.6$ of the involved mean object poses of the two still-standing instances. If that threshold is met, the two trajectories are joined under one ID.
- Lastly, we interpolate trajectories using a weighted running average scheme in order to yield smoother object motion, which should ultimately guarantee a more suitable pseudo-ground truth.

Based on these trajectories, we conducted two additional experiments outlined in the following.

Department of Computer Science, University of Freiburg, Germany.
Project page: http://batch3dmot.cs.uni-freiburg.de

TABLE S.1: Comparison of different training sets used to estimate Kalman filter covariance matrices of Prob3DMOT [22] using CenterPoint [3] object proposals. Results are shown in terms of the AMOTA scores on the nuScenes validation set.

| Training Set | Overall | Bicyc. | Bus | Car | Moto | Ped. | Trailer | Truck |
|---|---|---|---|---|---|---|---|---|
| *nusc-train* | 0.614 | 0.387 | 0.791 | **0.780** | 0.528 | **0.698** | **0.494** | 0.622 |
| *nusc-train + pseudo-test* | **0.624** | **0.436** | 0.808 | 0.779 | **0.549** | 0.693 | 0.457 | **0.645** |
| *pseudo-train + pseudo-test* | 0.611 | 0.377 | **0.822** | 0.768 | 0.540 | 0.695 | 0.447 | 0.625 |

#### B. Pseudo-Label Training

For testing the efficacy of our model predictions, we employ a pseudo-label training scheme that is exemplified for the nuScenes dataset. We use the additional data samples in the test split, which does not contain openly accessible annotations. We employ the 3D-PM-CL instance (see Table II) optimized on the training set to yield pseudo-labels for the test-split. The postprocessing steps outlined in the previous section are applied to yield refined trajectories. Using a combination of the training split and pseudo-labels of the test-split (or a subset), a new 3D-PM model is trained. In general, we assume that the unlabeled test split is created by the same data generation process as the labeled share of the training dataset.

We propose a method to filter high confidence predictions from the test-split. Leveraging high-confidence model predictions allow to capitalize on strengths of the model instead of its weaknesses. While we do not know whether a scene itself is more or less complex, we can analyze whether the model was confident about its predictions. A batch of 5 frames yields a set of edge scores, each normalized in $[0, 1]$. As each edge prediction does not represent a statement in comparison to another randomly chosen edge, each edge stands for itself. Thus, the problem of edge score prediction essentially boils down to a case of binary classification. We record the predicted edge scores per batch, construct a histogram (see Fig. S.1(a)) and normalize scores in order to construct a synthetic probability distribution. Computing the normalized entropy of the distribution as

$$H = -\sum_{(j,i)\in E} \frac{z_{(j,i)} \log z_{(j,i)}}{\log |E|}, \tag{1}$$

provides a measure of tracking uncertainty. A uniform edge score distribution leads to an entropy of $H = 1$ and non-uniform distributions lead to $H < 1$. In terms of tracking, we observe that confident model predictions generally show smaller entropies (their distributions are less uniform) and vice versa (see Fig. S.3). Note that the histograms depicted in Fig. S.1 are log-scaled. By computing an average scene entropy using

TABLE S.2: Comparison of different pseudo-label training schemes utilizing the 3D-PM model architecture on the nuScenes validation set.

| Training Set | AMOTA↑ | AMOTP↓ | MOTA↑ | Recall↑ | FP↓ | FN↓ | IDS↓ | FRAG↓ |
|---|---|---|---|---|---|---|---|---|
| *nusc-train* | 0.708 | 0.630 | 0.612 | 0.719 | 11102 | 18640 | 688 | 383 |
| *nusc-train + pseudo-test* | 0.709 | 0.605 | 0.611 | 0.717 | 11470 | 18566 | 626 | 369 |
| *nusc-train + pseudo-test-entropy* | **0.711** | 0.611 | 0.607 | **0.726** | 11323 | 18460 | 663 | 386 |
| *pseudo-train + pseudo-test* | 0.708 | 0.592 | 0.606 | 0.712 | 11958 | 18543 | 683 | 365 |
| *pseudo-test* | 0.705 | 0.592 | 0.605 | 0.722 | 12330 | 18317 | 724 | 376 |

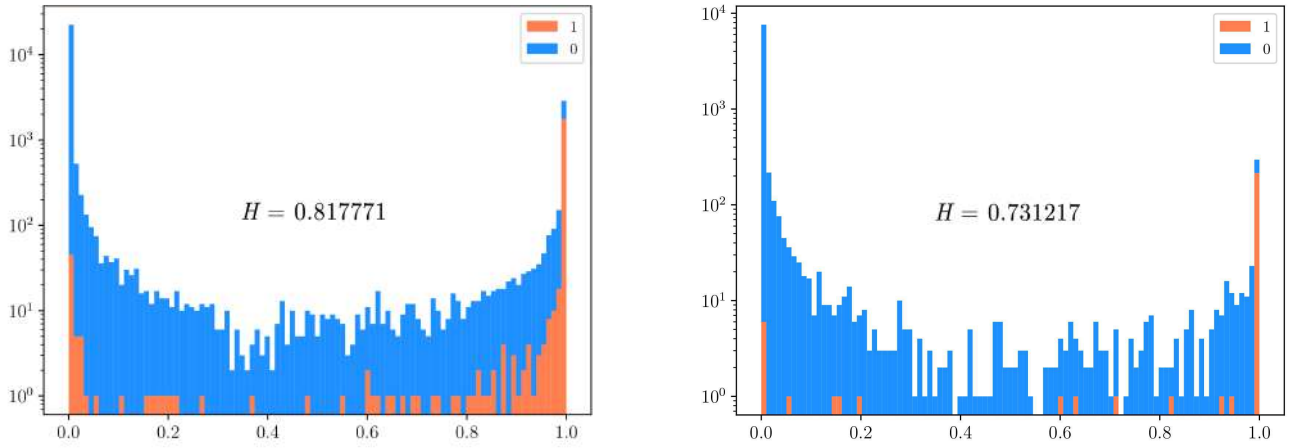



Fig. S.1: Log-scaled edge score histograms displaying absolute frequencies over respective edge scores including the normalized entropy of the corresponding normalized probability distribution: Low confidence (left), high confidence (right). Orange bars represent edges with GT label 1, blue bars represent inactive edges with GT label 0.

the respective batch entropies, the set of unlabeled scenes can be categorized into relatively certain and uncertain predictions. The low-confidence (high entropy) model prediction given in Fig. S.1 shows a large number of false positive predictions (blue-colored edge scores in $[0.6, 1.0]$), which is reflected in the cluttered tracking result depicted in Fig. S.3 (left). On the contrary, the high-confidence tracking prediction produces a less-cluttered set of trajectores as given in Fig. S.3 (right). This is further detailed in Sec. S.3.

As means to demonstrate our findings, we report the performance of the 3D-PM model when adding either unfiltered (*nusc-train + pseudo-test*) or entropy-filtered data (*nusc-train + pseudo-test-entropy*) to the human-annotated training set. The entropy-filtered data contains only scenes that show a scene-entropy higher than the mean scene entropy. As presented in Table S.2 the entropy-filtering induces a 0.2% improvement compared to the unfiltered case. Additionally, we also trained the same 3D-PM architecture using only trajectory labels that originate from the 3D-PM-CL instance (*pseudo-train + pseudo-test*), which produced a similar outcome as the human-annotations case (*nusc-train*). Most notably, we do not observe any performance decline as an effect of adding weaker annotations.

We show that we can even train the 3D-PM model architecture using only pseudo-labels from the test split which contains 150 scenes. We observe an AMOTA of 0.705 (*pseudo-test*) and a recall of 0.722 (see Table S.2). Therefore, the model shows a slight decrease in performance of about half a percent, however, using only 20% of the data samples that are weaker than human-annotations.

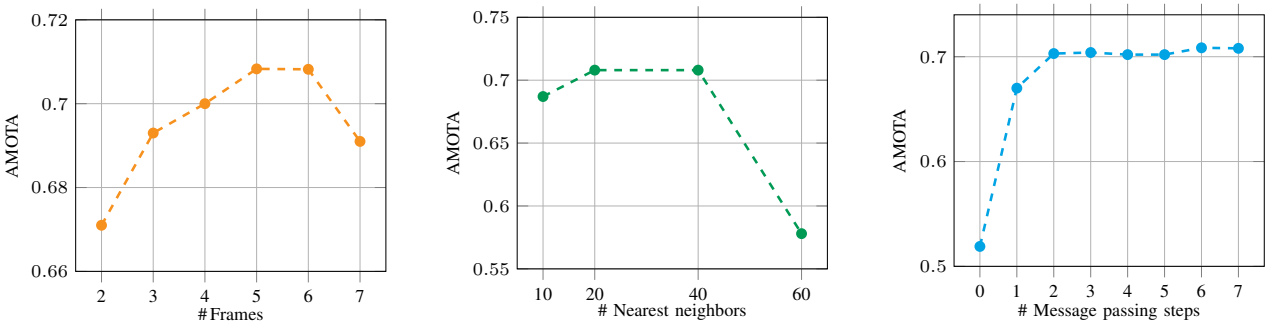
### C. Training an Online Kalman Filter Tracking Model

In addition to the pseudo-label training scheme, the experiments presented in this section use Kalman filter covariance matrix estimation introduced for Prob3DMOT [22] using the model generated pseudo-labels. The postprocessing steps outlined in Sec. S.1 are adopted in the same manner as in the experiment described above. We use a conjunction of pseudo-labeled nuScenes test data and human-annotations on the nuScenes train set to estimate the state uncertainty covariance $\Sigma$, the observation noise covariance $R$, and process uncertainty covariance $Q$ used by Prob3DMOT [22].

The results are presented in Table S.1. We observe a notable 1% gain in overall tracking accuracy compared to the standard case when using both the original training set and the pseudo-labels (*nusc-train + pseudo-test*). Especially the *Bicycle, Motorcycle* and *Truck* classes show performance improvements. We observe only a small overall decrease (-1.3%) when using pseudo-labels generated for both training (*pseudo-train*) and test set (*pseudo-test*). This demonstrates the efficacy of using pseudo-labels for *training* Kalman filters, based on data statistics. Analogous to the previous experiment, we do not observe significant performance decreases due to weaker annotations.

## S.2. Additional Ablation Study

In this section, we present an additional ablation study on the main hyperparameters of the proposed model. The most influential parameters in the Batch3DMOT framework are the number of frames, the number of nearest neighbors, and the GNN depth (the number of message passing steps). In order




(a) Effect of different frame rates (b) Variation of the number of nearest neighbors (c) Tracking accuracy over different GNN depths

Fig. S.2: Additional ablation study on the main Batch3DMOT model parameters. All chosen parameters stay constant apart from the one varied while its effect is measured using the AMOTA tracking score on the nuScenes validation set. The investigated model is the Batch3DMOT-3D-PM variant.

to identify suitable parameters (see Sec. IV), we perform a parameter study on these variables. All experiments originate from model trainings on the nuScenes train set and evaluated on the validation split. In each study, we only vary the parameter being ablated and keep all the other hyperparameters fixed.

### A. Number of Frames

The number of frames per batch determine whether the FN detections based on occlusions or FP detections due to, e.g., noisy readings or misjudgement, can be recovered from. Empirically, we find that a number of 5 frames is sufficient for stable tracking (especially in case of the 2Hz framerate used in nuScenes) and still provide the grounds for comparison against 3D Kalman filter tracking models. Note that the Batch3DMOT framework only performs linear one-step interpolation of output trajectories to arrive at the results. Thus, the model itself is not capable of overcoming occlusions based on intermediate prediction-update steps as used in Kalman filtering settings. Analogously, the chosen frame rate should allow stable offline tracking with the exception of long-term occlusion handling. The study presented in Fig. S.2 (c) shows a gradual increases in tracking accuracy (measured in terms of AMOTA) until a number of 5 and 6 frames is reached under 40 nearest neighbors. For 7 frames, we observe a stark decline, presumably due to the overall number of edges rising above the critical threshold as discussed in Sec. IV.

### B. Number of Nearest Neighbors

We investigate a variation of the number of nearest neighbors (NN) leading to a edges connected in the graph construction stage. This analysis only concerns the case of semantic category-disjoint edges. Based on the findings presented in Fig. S.2 (b), an increase in the number of neighbors higher than 40 generally leads to a performance decrease. As outlined in Sec. IV, we observe a maximum number of edges that guarantees learning success, which is exceeded in this case. On the contrary, we do not observe a performance decrease when only connecting the 20 NN over 5 frames using the proposed kinematic similarity metric (Eq. (4)), which effectively shows the efficacy of the metric. In our case, we choose 40 NN so as to overcome potential $\pm\pi$ orientation flips and velocity misjudgements stemming of noisy 3D object proposals. For 10 NN, we observe a 2.1% performance decline compared to 20 and 40 NN (AMOTA 0.708). In general, using 20 NN over 6 frames allows further performance improvement.

### C. Number of Message Passing Steps

The GNN depth is a crucial parameter determining the degree of information exchange across the proposed tracking graph. As depicted in Fig. S.2 (c), executing at least one message passing step increases tracking accuracy from AMOTA 0.519 to 0.670. Further incremental increases lead to slight improvements of the tracking performance with a maximum at 6 message passing steps (AMOTA 0.708). Compared to the other two parameters, the GNN depth bears less potential for further optimization.

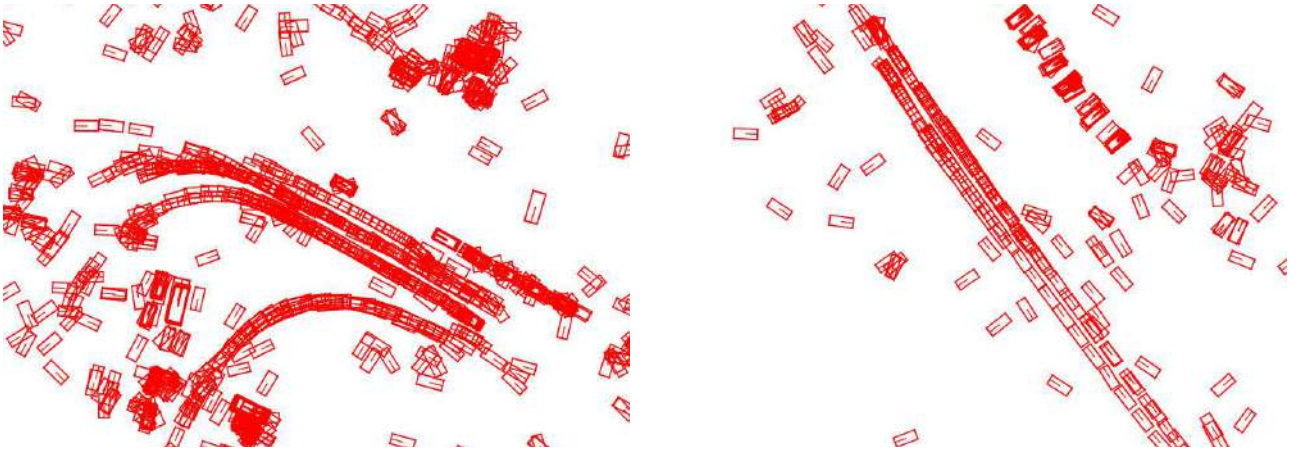
## S.3. Qualitative Insights

The low-confidence and high-confidence scenes shown in Fig. S.1 are also depicted in an accumulated BEV manner over 40 frames in Fig. S.3. The left column illustrates the detections, predicted trajectories and ground truth trajectories of the low-confidence scene shown in Fig. S.1 (left). The right column of Fig. S.3 depicts the results for a high-confidence scene.

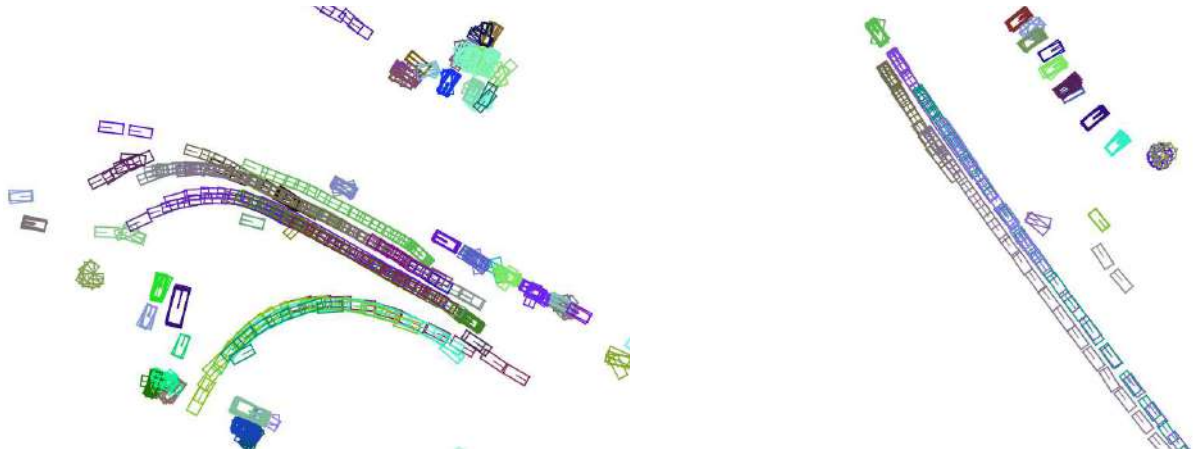
In accordance with the edge score histogram (Fig. S.1 right), we observe a much less cluttered tracking result in (Fig. S.3 (b) right). On the contrary, the low confidence scene (Fig. S.3 left) exhibits a higher number of (presumably) false positive detections contained in the predicted trajectories. Therefore, we attribute the significant portion of incorrectly predicted edge scores with ground truth label of 0 in the range $[0.6, 1.0]$ (see Fig. S.1 left).

These findings provide a qualitative understanding of the efficacy of the introduced entropy-filtering system. Due to the unavailability of ground truth edge labels in the test split, it is infeasible to separately assess the prediction quality of either active (orange regime) or inactive (blue regime) edges in a segregated manner as shown in Fig. S.1.

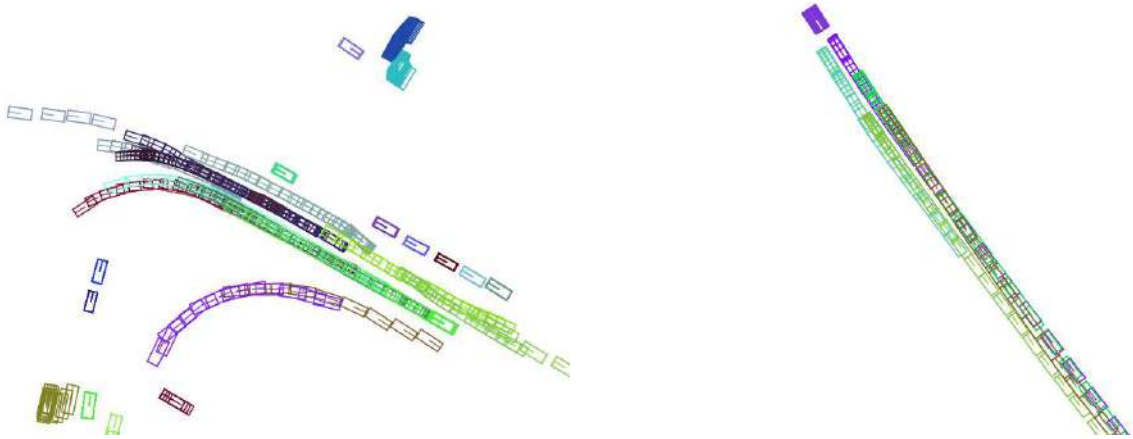
Low-confidence scene ($H = 0.817771$) High-confidence scene ($H = 0.731216$)



a) Accumulated 3D object proposals of CenterPoint [3] across 40 frames before matching.



b) Accumulated, non-interpolated trajectory predictions of Batch3DMOT-3D-PM-CL across 40 frames (*Car* category) for two levels of confidence. Each color denotes a particular tracking ID. Low confidence (left) and high confidence (right).



c) Accumulated ground truth trajectory predictions across 40 frames (*Car* category). Each color denotes a particular tracking ID.

Fig. S.3: Qualitative tracking results for two exemplary low confidence (left) and high confidence (right) scenes from the nuScenes validation split: Input 3D object proposals (a), unrefined predicted trajectories (b) and ground truth trajectories (c).

## 7.6 PADLoC: LiDAR-Based Deep Loop Closure Detection and Registration using Panoptic Attention

The appended paper [2] follows.

# PADLoC: LiDAR-Based Deep Loop Closure Detection and Registration using Panoptic Attention

José Arce[1], Niclas Vödisch[1], Daniele Cattaneo[1], Wolfram Burgard[2], and Abhinav Valada[1]

***Abstract*—A key component of graph-based SLAM systems is the ability to detect loop closures in a trajectory to reduce the drift accumulated over time from the odometry. Most LiDAR-based methods achieve this goal by using only the geometric information, disregarding the semantics of the scene. In this work, we introduce PADLoC, a LiDAR-based loop closure detection and registration architecture comprising a shared 3D convolutional feature extraction backbone, a global descriptor head for loop closure detection, and a novel transformer-based head for point cloud matching and registration. We present multiple methods for estimating the point-wise matching confidence based on diversity indices. Additionally, to improve forward-backward consistency, we propose the use of two shared matching and registration heads with their source and target inputs swapped by exploiting that the estimated relative transformations must be inverse of each other. Furthermore, we leverage panoptic information during training in the form of a novel loss function that reframes the matching problem as a classification task in the case of the semantic labels and as a graph connectivity assignment for the instance labels. We perform extensive evaluations of PADLoC on multiple real-world datasets demonstrating that it achieves state-of-the-art performance. The code of our work is publicly available at http://padloc.cs.uni-freiburg.de.**

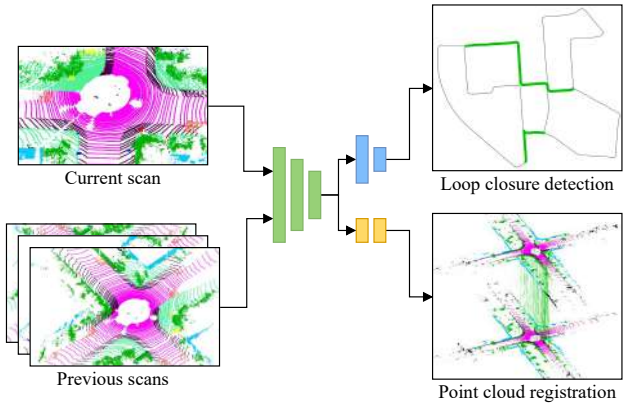



Fig. 1. We propose PADLoC that jointly detects loop closures for LiDAR-based SLAM and simultaneously performs point cloud registration. In addition to geometric information, we leverage panoptic segmentation annotations during training to facilitate more robust point matching.

## I. Introduction

Simultaneous Localization and Mapping (SLAM) is a core task of autonomous mobile robots. Typically, SLAM approaches consist of two steps: alignment of consecutive measurements, e.g., from wheel odometry, followed by loop closure detection and registration. Reliable loop closure detection enables a robot to recognize places it has seen before to optimize its world representation and belief of its current position, reducing the drift over time. Thus, it is considered a fundamental component of SLAM systems. Many SLAM systems have been proposed for different sensor modalities including cameras [1] and LiDARs [2]. While vision-based methods fail in challenging lighting conditions such as illumination changes, LiDAR-based approaches are more robust to such alterations and provide a more accurate representation of the environment. In this work, we address the joint problem of loop closure detection and map registration for LiDAR-based SLAM. A high-level overview of our approach is depicted in Fig. 1.

Similar to other fields, learning-based approaches have started to replace handcrafted methods due to their better generalization ability and faster runtime [3], [4]. Typically, deep neural networks predict point correspondences which are then used in differential singular value decomposition (SVD) to compute the transformation between two point clouds [5], [6]. Motivated by the success of transformers in natural language processing and computer vision tasks, attention-based architectures were recently introduced for point cloud registration [6], [7], [8] to encode context across points. While existing works do not consider the semantic meaning of the different inputs to a transformer cell, i.e., queries, keys, and values, we explicitly take advantage of the internal structure by feeding in abstract features and raw points separately.

Although geometric information suffices for classical point cloud registration such as Iterative Closest Point (ICP) [9], they can be further stabilized by integrating semantic information [2], [10], [11]. Inspired by recent semantic mapping approaches [10], [12] and methods that exploit panoptic information for vision-based loop closure detection [13], we leverage panoptic segmentation of point clouds in this work. Unlike related methods, our approach requires panoptic labels only while training but not during deployment, making it more versatile. We evaluate the loop closure detection and point cloud registration performance on three real-world autonomous driving datasets, namely, KITTI [14], Ford campus [15], and an in-house dataset recorded in Freiburg, Germany. We compare against both state-of-the-art handcrafted and deep learning-based methods and demonstrate that PADLoC achieves state-of-the-art performance. We also present several ablation studies on the different components of our approach validating our architectural design choices.

The main contributions of this work are as follows:

1) We propose PADLoC, a transformer encoder architecture for point cloud matching and registration. Unlike existing methods, we use separate inputs as keys, values, and

[1] Department of Computer Science, University of Freiburg, Germany.
[2] Department of Engineering, University of Technology Nuremberg, Germany.
This work was funded by the European Union's Horizon 2020 research and innovation program under grant agreement No 871449-OpenDR and the DFG Emmy Noether Program.

queries effectively, exploiting the transformer structure.
2) We define a novel loss function that leverages panoptic information for registration. We further propose formulating both geometric and panoptic registration losses as bidirectional functions that greatly improve performance.
3) We study the effect of multiple weighting methods in SVD to enhance point matching.
4) We extensively evaluate our proposed approach and compare it to other point cloud matching and registration methods, using two openly available datasets and in-house data recorded in Freiburg, Germany.
5) We release our code and the trained models at http://padloc.cs.uni-freiburg.de.

## II. Related Work

In this section, we first provide an overview of LiDAR-based loop closure detection techniques for SLAM, followed by various methods for point cloud registration, and finally describe approaches that leverage semantic segmentation for either task.

*Loop Closure Detection:* Traditionally, handcrafted methods for LiDAR loop closure detection can be categorized into local feature-based and global feature-based methods. Inspired by the success of local feature-based methods in images, approaches from the first category design similar descriptors and adapt them to 3D point cloud data. 3D keypoint descriptors such as Fast Point Feature Histograms (FPFH) [16] and Normal-Aligned Radial Features (NARF) [17] are used to extract local features, which are then aggregated in a bag-of-word model to detect loop closures. More recently, HOPN [18] exploits a bird's-eye view (BEV) representation and normal information to increase robustness to noise and viewpoint changes. Global feature-based approaches, on the other hand, summarize the whole point cloud into a single fingerprint, which is then compared against the fingerprints from past frames to detect loops. The M2DP [19] descriptor projects the point cloud into multiple 2D planes and combines density information computed on each plane into a global descriptor. Scan Context [20] combines a polar coordinate representation with partitioning to generate an image as a global descriptor. Subsequent works extended this method by adding additional information such as intensity [21] and semantic data [22]. Recently, many deep learning-based approaches have been proposed to overcome some of the limitations of handcrafted methods. PointNetVLAD [23] is built on top of the PointNet [24] architecture and generates a compact descriptor. OverlapNet [25] projects the point cloud into a range image and predicts the overlap and the yaw misalignment between a pair of frames. To increase viewpoint robustness and to reduce inference time, OverlapTransformer [26] adapts OverlapNet by including a transformer module. In this work, we build upon LCDNet [5] that uses learning-based feature extraction to generate global descriptors. LCDNet significantly improves loop closure in challenging conditions, such as reverse loops and, unlike other methods, does not require an ad-hoc function to compare two global descriptors.

*Point Cloud Registration:* Standard techniques for point cloud registration can be broadly classified into two main categories. The first category comprises the Iterative Closest Point (ICP) algorithm [9] and its variants [10], [27]. These methods require an initial guess on the transformation and then iteratively alternate between finding matches between points by exploiting some heuristics and estimating the transformation based on these matches. Methods of the second category use a two-stage approach. They first extract local point features, e.g., FPFH [16], and then regress the transformation using robust estimators such as RANSAC [28]. While methods of the first category are prone to get stuck in local minima if the provided initial guess is not accurate enough, approaches of the second category are sensitive to noise and incorrect matches. Many deep learning-based approaches have also been proposed to solve the point cloud registration task. PointNetLK [29] is a pioneering work that combines an architecture inspired by PointNet [24] and a modified Lucas-Kanade algorithm to iteratively improve the registration. Inspired by the success of transformers in other fields, Deep Closest Point [6] uses an attention-based module to predict soft matches between two point clouds, which are fed to a differentiable SVD layer to infer a rigid transformation. Following the same idea, both GeoTransformer [7] and REGTR [8] directly learn to predict point correspondences using both self and cross-attention. Our previous work LCDNet [5] combines a state-of-the-art feature extraction architecture with a place recognition head and a relative pose head for simultaneous loop closure detection and point cloud registration. In this work, we adapt LCDNet [5] by integrating a transformer-based registration and matching module.

*Semantic-Aided Mapping and Localization:* Only a handful works have proposed to leverage semantic information for large-scale mapping and localization [10], [30], and particularly for loop closure detection. Based on semantic segmentation, SuMa++ [10] filters dynamic objects from a LiDAR-based map and extends the ICP algorithm with additional semantic constraints. While SuMa++ does not utilize semantic information for loop closure detection, RINet [31] explicitly addresses LiDAR-based place recognition via a rotation invariant global descriptor combining semantic and geometric information. For the same task, Kong *et al.* [11] propose to build a graph representation of point clouds, which are enriched by both semantic and instance segmentation and perform graph similarity matching. SA-LOAM [2] integrates a semantic-aided variant of ICP into the popular LOAM pipeline for point cloud registration. To address loop closure, it uses a similar graph representation as Kong *et al.* [11]. SV-Loop [13] is a loop closure detection method for vision-based SLAM. It separately proposes loop closure candidates based on raw images and panoptic segmentation maps, which are then fused to extract the most feasible candidates. In our approach, we exploit panoptic annotations of point clouds while predicting both loop closure detection and point cloud registration. Additionally, we only utilize them during the training process but not for deployment, making the method more versatile.

## III. Technical Approach

In this section, we introduce our novel PADLoC architecture for joint loop closure detection and point cloud registration. First, we detail the overall approach comprising the modules shown in Fig. 2. We then describe the loss functions that we employ, including our proposed loss that leverages panoptic annotations of point clouds.

### A. Model Architecture

In this section, we describe the individual components of the PADLoC architecture. We build upon our previously proposed LCDNet [5], where instead of using a differentiable approximation of the optimal transport to obtain point matches, we propose to leverage the cross-attention matrices of transformers. The learnable keys, queries, and values weights yield a better latent representation of the features, and thus more reliable matches. As depicted in Fig. 2, the overall PADLoC architecture consists of three modules: feature extraction, loop closure detection, and point cloud registration. During training, we employ a triplet-based training scheme by feeding in an anchor point cloud along with a positive sample of a loop closure and a negative sample.

*Feature Extraction:* The feature extraction backbone converts raw input scans into a high-dimensional embedding that is used as a common input for both loop closure detection and point cloud registration. It effectively exploits global and local contexts and is built upon the PV-RCNN architecture [32]. In detail, a point cloud $\mathbf{P}$, comprising 3D coordinates and reflectance values, is discretized into a voxel grid which is then passed through four sparse 3D convolutional layers to generate the feature maps at different resolutions. The final feature map is then stacked to form a BEV feature map. Additionally, the original point cloud is downsampled using the Farthest Point Sampling (FPS) algorithm to uniformly select $n$ keypoints. The feature vector of each sampled keypoint is assembled by combining the feature maps from each convolutional layer in a neighborhood of the sampled keypoint using the Voxel Set Abstraction module [32]. The raw input of each sampled keypoint is also appended to each feature vector, along with the corresponding entry in the BEV feature map. Finally, these intermediate features are fed through a multilayer perceptron to obtain the final feature vector for each sampled point. This module thus outputs the sampled keypoints $\mathbf{Q}$ and the corresponding features $\mathbf{F}$.

*Loop Closure Detection:* The global descriptor module of PADLoC further encodes the previously extracted features to perform loop closure detection. For this task, we employ the NetVLAD layer [33] to convert the feature vectors $\mathbf{F}$ of the anchor, the positive, and the negative points to their respective final descriptor $\mathbf{D}$. In detail, NetVLAD learns $k$ clusters along with corresponding descriptors, which are aggregated in a single descriptor $v$ for the entire point cloud. The final descriptors $\mathbf{D}$ of length $g$ are then obtained via a context gating layer. This learnable pooling operation with weights $\mathbf{W}_G$ and bias $\mathbf{b}_G$ is defined as

$$\mathbf{D} = \sigma\big(\mathbf{W}_G \cdot v + \mathbf{b}_G\big) \odot v, \tag{1}$$

where $\sigma(\cdot)$ refers to the logistic sigmoid function and $\odot$ denotes the element-wise multiplication.

During inference, the descriptors are stored in such a manner that allows for efficient querying of the nearest neighbor in descriptor space. If the distance between the descriptor of the current scan and its nearest neighbor is below a predefined threshold, they are considered to form a loop closure. To avoid matching consecutive scans, we introduce a small temporal distance between the current scan and potential neighbors.

*Point Matching:* The matching module shown in Fig. 3 predicts soft correspondences $\widehat{M}_s^t$ between keypoints $\mathbf{Q}^s$ and $\mathbf{Q}^t$ of a source point cloud $s$ and a target point cloud $t$, respectively. Additionally, it outputs projected target coordinates $\widehat{\mathbf{Q}}^t$ which are linear combinations of the original target coordinates with a one-to-one pairing with the points of the source set and a confidence weight $\widehat{w}_M$ for each of these matches. Inspired by the success of transformers in related tasks, we propose a novel architecture that performs cross-attention directly on the encoder part, obviating the need for a decoder by feeding independent inputs for the queries, keys, and values.

$$\widehat{\mathbf{Q}}^t = \mathbf{W}_Q \cdot \mathrm{TEL}\left(\mathbf{F}^s, \mathbf{F}^t, \mathbf{Q}^t\right) + \mathbf{b}_Q, \tag{2}$$

where $\mathrm{TEL}(q, k, v)$ is a transformer encoder layer, as defined in [34], but applied to independent query $q$, key $k$, and value $v$ inputs. $\mathbf{W}_Q \in \mathbb{R}^{3\times f}$ and $\mathbf{b}_Q \in \mathbb{R}^3$ are learnable weights and biases used to reduce the dimensionality of the output from the size $f$ of the features $\mathbf{F}$ to 3D space. We directly use the encoder's attention matrix as our matching $\widehat{M}_s^t$, since it already encodes the similarity between the features of the two sets of points. Moreover, each row in the attention matrix represents the probability distribution of matching the corresponding point from the source set to all of the points from the target set, given that it is non-negative and adds up to one due to the use of the softmax function.

From the matching matrix $\widehat{M}_s^t$, we compute a confidence weight for every pair of point correspondences by penalizing the dispersion of the distributions represented by each row. We propose using a diversity metric for that purpose, such as the Shannon Entropy (E), the order-$r$ Hill number ($\mathrm{D}^r$), or the Berger-Parker index (BP), defined as

$$\mathrm{E}(\mathbf{p}) = -\sum_i p_i \cdot \log(p_i), \tag{3}$$

$$\mathrm{D}^r(\mathbf{p}) = \Big(\sum_i p_i^r\Big)^{\frac{1}{1-r}}, \tag{4}$$

$$\mathrm{BP}(\mathbf{p}) = \max(\mathbf{p}), \tag{5}$$

where $\mathbf{p}$ is a vector of probabilities.

The weights $\widehat{w}_M$ are obtained using either of the aforementioned metrics by normalizing their output to a $[0, 1]$ range, where the two extreme weights of 0 and 1 respectively correspond to a uniform and an infinitely sharp distribution.

*Point Cloud Registration:* To obtain the final relative transformation $\widehat{H}_s^t$ from a source point cloud to a target point cloud, we perform a weighted version of the Kabsch-Umeyama algorithm that finds the optimal translation and rotation between

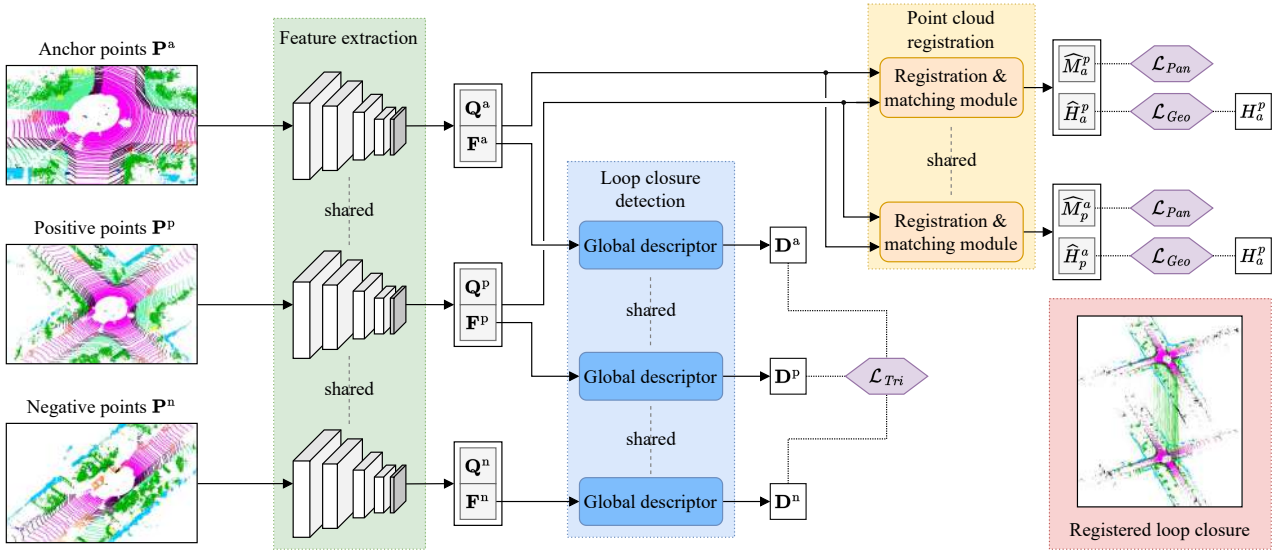



Fig. 2. Overview of our proposed PADLoC architecture for joint loop closure detection and point cloud registration. It consists of a shared feature extractor (green) followed by a global descriptor head (blue) for loop closure detection and a registration and matching module (orange) to estimate the 6-DoF transform between two point clouds (red). To train the global descriptor, we use a triplet loss (purple) that compares the anchor point cloud with a positive and negative sample. For the registration module, we leverage losses (purple) based on both geometric and panoptic information.

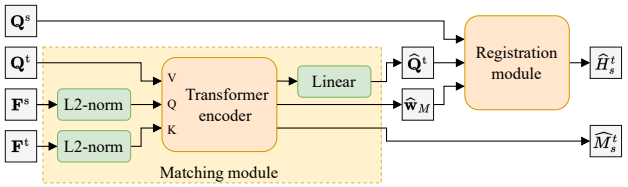



Fig. 3. The matching module consists of a transformer encoder that takes the extracted features of the source keypoints $\mathbf{F}^s$ as query, the features of the target keypoints $\mathbf{F}^t$ as key, and the corresponding target keypoints $\mathbf{Q}^t$ as value. It outputs both soft correspondences $\widehat{M}_s^t$ and projected target points $\widehat{\mathbf{Q}}^t$ along with confidence weights $\widehat{w}_M$. The latter is fed together with the source keypoints $\mathbf{Q}^s$ to a registration module that performs weighted SVD to estimate the final transform $\widehat{H}_s^t$.

two sets of points by minimizing the root mean square error of the point pairs. First, the correspondences between the sampled source keypoints $Q^s$ and the projected target keypoints $\widehat{Q}^t$ are weighted by the matching confidences $\widehat{w}_M$. Subsequently, the optimal translation is computed as the difference between the weighted centroids of the two point clouds. Finally, the optimal rotation is obtained via SVD of the weighted covariance matrix of the two sets of keypoints. This approach is fully differentiable and thus allows end-to-end training by measuring the error of the predicted transformation with respect to the ground truth relative pose.

### B. Loss Functions

Our total loss function consists of a weighted sum of the triplet loss $\mathcal{L}_{Tri}$ for loop closure detection as well as a geometric loss $\mathcal{L}_{Geo}$ and the newly proposed panoptic loss $\mathcal{L}_{Pan}$ for point cloud registration. The following paragraphs describe these losses in greater detail.

*Triplet Loss:* For the loop closure detection task, we use the triplet loss. It enforces a small distance between the descriptors of an anchor point cloud and a positive point cloud, i.e., a loop closure LiDAR scan while increasing the distance between the descriptors of the anchor and a negative point cloud, i.e., a LiDAR scan taken at a different place.

$$\mathcal{L}_{Tri} = \max\left\{d(\mathbf{D}^{\mathrm{a}}, \mathbf{D}^{\mathrm{p}}) - d(\mathbf{D}^{\mathrm{a}}, \mathbf{D}^{\mathrm{n}}) + m, 0\right\}, \tag{6}$$

where the descriptors of the anchor, the positive, and the negative sample are denoted by $\mathbf{D}^{\mathrm{a}}$, $\mathbf{D}^{\mathrm{p}}$, and $\mathbf{D}^{\mathrm{n}}$, respectively. $d(\cdot)$ is a given distance function and $m$ refers to the desired separation margin.

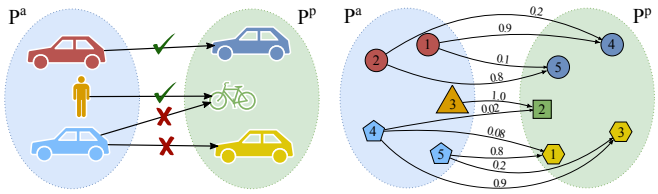
*Geometric Loss:* We formulate our geometric loss $\mathcal{L}_{Geo}$ as a sum of a pose loss $\mathcal{L}_{Pos}$ and an auxiliary matching loss $\mathcal{L}_{Mat}$. For the pose loss, we compare the predicted relative transformation $\widehat{H}_a^p$ from the anchor to the positive sample with the ground truth transformation $H_a^p$ by applying both to the coordinates of the same sampled point cloud $\mathbf{Q}^{\mathrm{a}}$. Then we compute the mean absolute error in the Euclidean space.

$$\mathcal{L}_{Pos} = \mathrm{mean}\left(\mathrm{abs}\left(\widehat{H}_a^p \cdot \mathbf{Q}^{\mathrm{a}} - H_a^p \cdot \mathbf{Q}^{\mathrm{a}}\right)\right) \tag{7}$$

We further evaluate the geometric correspondence between the sampled anchor $\mathbf{Q}^{\mathrm{a}}$ and positive points $\mathbf{Q}^{\mathrm{p}}$ leveraging the predicted matching matrix $\widehat{M}_p^a$. In detail, we transform the anchor points with the ground truth transformation $H_p^a$ and project the positive sample with $\widehat{M}_p^a$.

$$\mathcal{L}_{Mat} = \mathrm{mean}\left(\mathrm{abs}\left(H_a^p \cdot \mathbf{Q}^{\mathrm{a}} - \widehat{M}_p^a \cdot \mathbf{Q}^{\mathrm{p}}\right)\right) \tag{8}$$

*Panoptic Loss:* In addition to the geometric point correspondences, we propose to leverage panoptic information to register two point clouds. In detail, we formulate a novel panoptic loss $\mathcal{L}_{Pan}$ as the sum of semantic misclassification losses $\mathcal{L}_{Sem}$ and $\mathcal{L}_{Mes}$ as well as a multi-matched object loss $\mathcal{L}_{Mmo}$.




(a) Object matching (b) Graph representation

Fig. 4. The multi-matched object loss penalizes matching an object in the anchor point cloud to multiple objects in the positive sample. Unlike the semantic misclassification losses, the multi-matched object loss does not consider the semantic class, as depicted in (a). By exploiting a graph representation shown in (b) of the point cloud, it enforces that all points of the same object are matched to points of another object.

We treat the matching process as a classification task, where the projected positive points are assigned a semantic class. While a cross-entropy loss is commonly used in classification problems, due to the fact that the proposed class logits are not the output of either a logistic or softmax activation, we empirically found that a mean absolute error resulted in a more stable training process. First, we use the semantic labels to construct one-hot encoded matrices $\mathbf{K}^{\mathrm{a}}$ and $\mathbf{K}^{\mathrm{p}}$ for the anchor and positive samples, respectively. Using the predicted matching matrix $\widehat{M}_p^a$, we define the semantic loss as

$$\mathcal{L}_{Sem} = \operatorname{mean}\left(\operatorname{abs}\left(\mathbf{K}^{\mathrm{a}} - \widehat{M}_p^a \cdot \mathbf{K}^{\mathrm{p}}\right)\right). \tag{9}$$

Additionally, to allow flexibility in the semantic misclassification, we define a mapping from the semantic class labels to a set of super-classes, e.g., both *car* and *truck* belong to the *vehicle* class. Further details can be found in Sec. IV-A. Analogously to the semantic loss, we construct one-hot encoded matrices $\mathbf{J}^a$ and $\mathbf{J}^p$ and define the meta-semantic loss as

$$\mathcal{L}_{Mes} = \operatorname{mean}\left(\operatorname{abs}\left(\mathbf{J}^{\mathrm{a}} - \widehat{M}_p^a \cdot \mathbf{J}^{\mathrm{p}}\right)\right). \tag{10}$$

In our novel multi-matched object loss, we further exploit the instance labels to encourage the network to match entire objects consistently from one point cloud to the other. This is done by penalizing matches of points from a single object in the anchor to multiple objects in the positive sample. Unlike the previously introduced semantic misclassification losses, the multi-matched object loss does not consider the semantic class of objects, as depicted in Fig. 4 (a).

Since instance labels may not be consistent throughout a driving sequence, it is not feasible to purely rely on the IDs. Therefore, we construct adjacency matrices $\mathbf{O}^{\mathrm{a}}$ and $\mathbf{O}^{\mathrm{p}}$ of a graph representation of the point clouds, where nodes represent points and edges connect points of the same instances of a semantic class. The predicted matching matrices $\widehat{M}_p^a$ and $\widehat{M}_a^p$ can then be considered as weighted, directed, bipartite graphs between the two sets of points (see Fig. 4 (b)). Finally, we formulate the multi-matched object loss as

$$\mathcal{L}_{Mmo} = \operatorname{mean}\left((1 - \mathbf{O}^{\mathrm{a}}) \odot \left(\widehat{M}_p^a \cdot \mathbf{O}^{\mathrm{p}} \cdot \widehat{M}_a^p\right)\right), \tag{11}$$

where $\odot$ denotes the element-wise multiplication.

*Reverse Losses:* Finally, we add a second instance of the registration module that processes the swapped source $s$ and target $t$ inputs and predicts the inverse relative transformation. Both the geometric and the panoptic losses can be reformulated accordingly. The total loss is then formulated by averaging the results of both the original and the reverse versions.

## IV. Experimental Evaluation

In this section, we evaluate our proposed PADLoC architecture with respect to multiple handcrafted and learning-based methods. We perform several experiments and present both the loop closure detection and the point cloud registration results. Finally, we evaluate the design choices in PADLoC by performing multiple ablation studies.

### A. Implementation Details

We perform experiments on two publicly available autonomous driving datasets, namely the KITTI odometry benchmark [14] and the Ford campus vision and LiDAR dataset [15]. Additionally, we also present results on a more challenging in-house dataset recorded in Freiburg, Germany. For training, we leverage the ground truth panoptic annotations from the SemanticKITTI dataset [37]. In particular, we train all models on sequences {00, 05, 06, 07, 09} and, if not specified otherwise, evaluate on sequence 08. We consider a loop closure between two point clouds if their poses are within a $4\,\mathrm{m}$ distance and took place at a minimum of 50 frames apart to avoid consecutive scans. Unless otherwise specified, we use $n = 4096$ keypoints, set the feature size to $f = 640$, the descriptor length to $g = 256$, and the number of clusters $k = 64$. To improve the invariance of the model with respect to the inputs' position and orientation, we augment the data during training by applying a random rigid transformation to the input point clouds with a uniform translation of $\pm 1.5\,\mathrm{m}$ in the $x$ and $y$ axes and $\pm 0.25\,\mathrm{m}$ along $z$, and a uniform rotation of $\pm 3°$ for the roll and pitch angles and $\pm 180°$ for the yaw. We train all our models on a server with 4 NVIDIA RTX A6000 GPUs for 150 epochs with a batch size of $b = 8$. We use the Adam optimizer with an initial learning rate of $\lambda = 0.004$, halved after epochs 40 and 80, and with a weight decay of $5 \times 10^{-6}$.

The total loss function is computed as a weighted sum of the components described in Sec. III-B, with weights $w_{Tri} = 1.0$, $w_{Pos} = 1.0$, $w_{Mat} = 0.05$, $w_{Sem} = 0.125$, $w_{Mes} = 0.5$, and $w_{Mmo} = 10.0$. We use a triplet margin of $m = 0.5$ and the the L2 distance as the distance function in Eq. 6. For the semantic super-classes, we follow the definitions of Cityscapes [38] and group the semantic labels into *flat*, *human*, *vehicle*, *construction*, *object*, *nature*, and *void*. Based on the ablation study presented in Sec. IV-D, we use the Berger-Parker index to compute the confidence weights.

### B. Loop Closure Detection

To evaluate the loop closure detection performance, we compare PADLoC with the handcrafted methods M2DP [19], Intensity Scan Context (ISC) [21], Scan Context [35], and

TABLE I
COMPARISON OF LOOP CLOSURE DETECTION AND POINT CLOUD REGISTRATION PERFORMANCE

| | Method | KITTI Seq. 08 [14] | | | Ford Seq. 01 [15] | | | Freiburg *(in-house)* | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | AP ↑ | $r_{err}$ [°] ↓ | $t_{err}$ [m] ↓ | AP ↑ | $r_{err}$ [°] ↓ | $t_{err}$ [m] ↓ | AP ↑ | $r_{err}$ [°] ↓ | $t_{err}$ [m] ↓ |
| Handcrafted | M2DP [19] | 0.05 | — | — | 0.89 | — | — | 0.60 | — | — |
| | Scan Context* [35] | 0.65 | 3.11 | — | 0.97 | 16.68 | — | **0.74** | 52.70 | — |
| | LiDAR-Iris* [36] | 0.64 | 1.84 | — | 0.90 | 1.66 | — | 0.73 | 46.24 | — |
| | ISC* [21] | 0.31 | 6.27 | — | 0.62 | 6.15 | — | 0.38 | 51.02 | — |
| | ICP (pt2pt) [9] | — | 160.63 | 2.41 | — | 9.56 | 2.79 | — | 89.43 | 2.37 |
| | ICP (pt2pl) [9] | — | 160.73 | 2.49 | — | 9.16 | 2.62 | — | 89.25 | 2.25 |
| Learning | DCP [6] | — | 46.06 | 2.59 | — | 12.14 | 3.42 | — | 45.70 | 2.30 |
| | OverlapNet* [25] | 0.32 | 65.45 | — | 0.79 | 9.44 | — | 0.59 | 70.91 | — |
| | LCDNet [5] | 0.76 | **0.37** | 0.19 | 0.97 | 1.82 | 1.44 | 0.65 | 10.08 | **0.91** |
| | PADLoC (ours) | **0.81** | **0.37** | **0.16** | **0.98** | **1.50** | **1.33** | 0.67 | **9.30** | 1.41 |

Comparison of the average precision (AP) for loop closure detection as well as rotation error $r_{err}$ and translation error $t_{err}$ for point cloud registration of PADLoC with previous methods. All learning-based models are trained on the KITTI odometry benchmark dataset. PADLoC uses panoptic annotations from the SemanticKITTI dataset. Methods denoted with * only estimate the yaw between two point clouds instead of a full 6-DoF transformation. Bold and underlined values denote the best and second best scores, respectively.

LiDAR-Iris [36], as well as with the learning-based approaches LCDNet [5], OverlapNet [25], and Deep Closest Point (DCP) [6]. For DCP, we combine the feature extraction module of PADLoC with a full transformer-based matching module based on the authors' code release. For the other methods, we directly use the official code published by the respective authors. To compute the results on OverlapNet, we download the model weights provided on the project website that are trained on KITTI. We re-train the other learning-based methods on sequences {00, 05, 06, 07, 09} of the KITTI odometry benchmark [14], where PADLoC leverages the ground truth panoptic annotations from the SemanticKITTI dataset [37]. We evaluate all methods on sequence 08 of the KITTI dataset, sequence 01 of the Ford dataset, and an in-house dataset recorded in Freiburg, Germany.

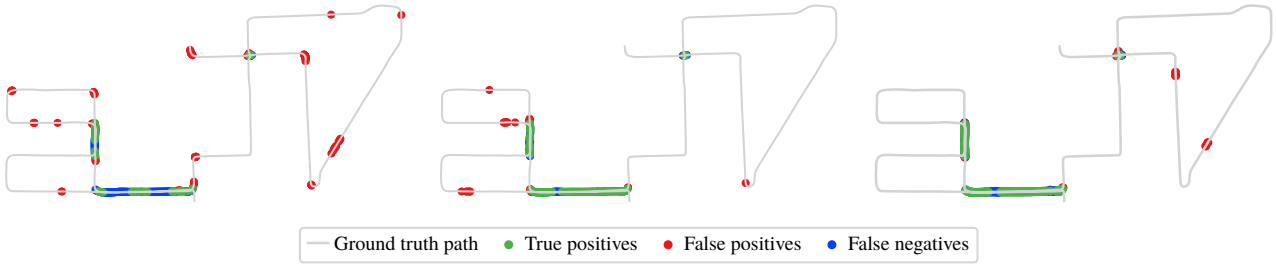
When evaluating PADLoC, we generate a descriptor $\mathbf{D}_i$ for every scan $i$ in a sequence and compute its similarity with that of all frames prior to the 50 previous scans. If a scan $j$ with the closest descriptor to that of scan $i$ has a similarity higher than a threshold $\tau$, then the pair $(i, j)$ is considered to form a loop closure. If the distance between the two ground truth poses is within $4\,\mathrm{m}$ for the KITTI dataset and $10\,\mathrm{m}$ for the Ford and Freiburg datasets, then it is considered as a true positive. Otherwise, it is considered a false positive. Conversely, if the pose distance is within $4\,\mathrm{m}/10\,\mathrm{m}$, but the similarity between the descriptors is below the threshold $\tau$, then we regard it as a false negative. By changing the value of $\tau$, we obtain precision-recall pairs that are then used to compute the average precision (AP).

In Table I, we report the average precision (AP) of PADLoC and the aforementioned baseline methods. Notably, PADLoC achieves the highest performance across the entire board for the evaluation sequences of both KITTI and Ford datasets. For our in-house Freiburg dataset, PADLoC yields the highest AP compared to the other learning-based approaches. Although the proposed transformer-based registration head and the panoptic losses do not directly influence the loop closure detection module, by sharing the same feature extractor between the two branches and training for the two tasks jointly, the better feature representation learned using our novel module and losses also improve the loop closure detection performance compared to LCDNet, which achieved the second best AP on both KITTI and Ford. Qualitative results of these methods on the KITTI dataset are visualized in Fig. 5. Compared to OverlapNet, both LCDNet and PADLoC correctly detect a higher number of loop closures, whereas PADLoC is able to further reduce the number of false positives. In Fig. 6, we plot the corresponding precision-recall curves that are used to compute the AP scores. We observe that PADLoC can maintain a higher precision for increased recall than LCDNet.

### C. Point Cloud Registration

To evaluate the point cloud registration performance, we compare PADLoC with the same handcrafted and learning-based methods described in Sec. IV-B, except for M2DP that does not perform point cloud registration. Since these handcrafted methods only estimate the yaw between two point clouds instead of the full 6-DoF transformation, we additionally compare with the Iterative Closest Point algorithm (ICP) [9], using both point-to-point and point-to-plane distances. Following the standard experimental setup [5], for LCDNet, DCP, and PADLoC, we perform point cloud registration with RANSAC using the extracted features before the respective matching layers.

As a measure of registration accuracy, we compute the rotation error $r_{err}$ in degrees and the translation error $t_{err}$ in meters of all positive pairs. We then average the errors over the entire sequence and present the results in Table I. We observe that PADLoC yields the smallest rotation error compared to all the handcrafted and learning-based methods on each of the evaluation sequences in the datasets. Additionally, it yields the smallest translation error on both the KITTI and Ford datasets, as well as the second lowest translation error on our in-house Freiburg dataset. LCDNet achieves the second best performance in most evaluations while achieving the lowest translation error on the Freiburg dataset. This result shows that while the feature extraction architecture and the training scheme play an important role, leveraging the cross-modal attention




(a) OverlapNet [25] (b) LCDNet [5] (c) PADLoC (ours)

Fig. 5. Qualitative loop closure detection results on KITTI sequence 08. The ground truth path corresponds to true negatives. While LCDNet reduces both false positives and false negatives compared to OverlapNet, the proposed PADLoC further decreases false positives.

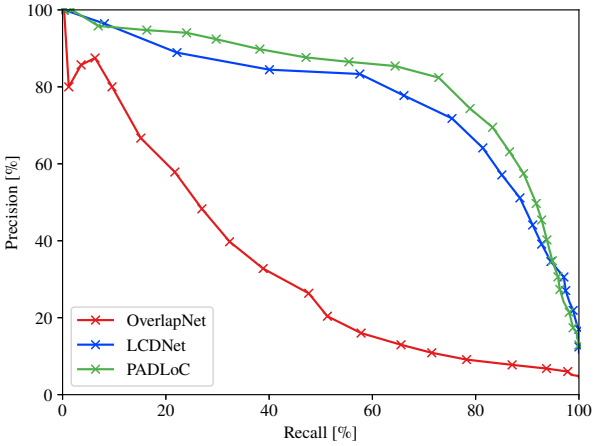



Fig. 6. Precision-recall curves for loop closure detection of learning-based methods evaluated on sequence 08 of the KITTI dataset.

matrices from the transformer architecture and the panoptic information during training further improves the point cloud registration performance. While LiDAR-Iris achieves the lowest rotation error across all the handcrafted methods, it only estimates the yaw angle instead of the full 6-DoF transformation.

### D. Ablation Studies

In this section, we present ablation studies to analyze the major design choices in the PADLoC architecture. As the RANSAC-based point cloud registration described in Sec. IV-C is applied only during inference and does not impact the training stage, all the experiments reported in this section do not exploit RANSAC.

*Confidence Weighting:* We investigate the effect of different weighting schemes on the performance of both loop closure detection and point cloud registration tasks. In Table II, we present the average precision (AP) as well as the registration errors $r_{err}$ and $t_{err}$ for the six weighting methods. In particular, uniform weights corresponding to unweighted SVD, column sum representing the method used in LCDNet [5], where weights are the sums along the columns of the matching matrix, and the diversity metrics from Sec. III-A, i.e., the Shannon Entropy, the order-$r$ Hill number with $r \in \{2, 4\}$, and the Berger-Parker index. We observe that both the Hill numbers and the Berger-Parker index outperform the other confidence

TABLE II
ABLATION STUDY ON CONFIDENCE WEIGHTS

| Method | AP ↑ | $r_{err}$ [°] ↓ | $t_{err}$ [m] ↓ |
|---|---|---|---|
| Uniform | 0.73 | 4.63 | 3.76 |
| Column sum | 0.76 | 6.34 | 3.62 |
| Shannon | 0.50 | 21.86 | 3.99 |
| Hill (r=2) | **0.89** | 2.45 | 2.00 |
| Hill (r=4) | 0.84 | 2.47 | 2.12 |
| Berger-Parker | 0.81 | **2.35** | **1.43** |

Average precision (AP) of loop closure detection as well as the mean error of point cloud registration, evaluated on KITTI sequence 08 for different weightings used in SVD.

TABLE III
INFLUENCE OF THE LOSS FUNCTIONS

| $\mathcal{L}_{Geo}$ | $\mathcal{L}_{Pan}$ | $\mathcal{L}_{Rev}$ | AP ↑ | $r_{err}$ [°] ↓ | $t_{err}$ [m] ↓ |
|---|---|---|---|---|---|
| ✓ | | | 0.70 | 3.09 | 1.62 |
| ✓ | ✓ | | 0.78 | 3.36 | 1.71 |
| ✓ | ✓ | ✓ | **0.81** | **2.35** | **1.43** |

Average precision (AP) of loop closure detection and the mean error of point cloud registration, evaluated on KITTI sequence 08 for the different loss functions.

weighting methods. Due to the substantially smaller translation error of the Berger-Parker index, improving the registration by more than $0.5\,\mathrm{m}$, we use this method in our final design.

*Effect of Losses:* To demonstrate the efficacy of our proposed panoptic loss $\mathcal{L}_{Pan}$ and the impact of formulating all losses in a bidirectional manner ($\mathcal{L}_{Rev}$), we consecutively add them to the original geometric loss $\mathcal{L}_{Geo}$. We present the results for both the loop closure detection and point cloud registration tasks in Table III. We observe that adding the proposed panoptic losses increases the average loop closure detection precision by further constraining which points can be matched together based on their semantic and instance labels. Furthermore, by including the second matching and registration head, along with its corresponding reverse losses as illustrated in the bottom row, the added bidirectional consistency constraint yields the highest AP and the smallest registration errors.

## V. Conclusion

In this paper, we proposed the novel PADLoC architecture for LiDAR-based joint loop closure detection and point

cloud registration. PADLoC is composed of a common feature extractor, a global descriptor as well as a transformer-based registration and matching module. Unlike previous approaches, we feed different inputs as value, query, and key to the transformer encoder exploiting its internal structure. We further introduced a new loss function that leverages ground truth panoptic annotations by penalizing matching points from different semantic classes as well as across multiple objects, and validated its positive impact. Through extensive experimental evaluations, we demonstrated the efficacy of PADLoC compared to both handcrafted and learning-based methods. In particular, we show that we can take advantage of the principles behind attention mechanisms to design transformer-based models with lower complexity than full encoder-decoder architectures, which yield more accurate results. Future work will focus on exploiting panoptic information in an online manner and applying the matching approach of PADLoC to point cloud registration tasks in other domains.

## REFERENCES


[1] N. Vödisch, D. Cattaneo, W. Burgard, and A. Valada, "Continual SLAM: Beyond lifelong simultaneous localization and mapping through continual learning," in *Int. Symposium of Robotics Research*, 2022.

[2] L. Li, X. Kong, X. Zhao, W. Li, F. Wen, H. Zhang, and Y. Liu, "SA-LOAM: Semantic-aided LiDAR SLAM with loop closure," in *Int. Conf. on Robotics and Automation*, 2021, pp. 7627–7634.

[3] B. Bešić and A. Valada, "Dynamic object removal and spatio-temporal RGB-D inpainting via geometry-aware adversarial learning," *IEEE Transactions on Intelligent Vehicles*, vol. 7, no. 2, pp. 170–185, 2022.

[4] N. Gosala and A. Valada, "Bird's-eye-view panoptic segmentation using monocular frontal view images," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 1968–1975, 2022.

[5] D. Cattaneo, M. Vaghi, and A. Valada, "LCDNet: Deep loop closure detection and point cloud registration for LiDAR SLAM," *IEEE Transactions on Robotics*, pp. 1–20, 2022.

[6] Y. Wang and J. Solomon, "Deep Closest Point: Learning representations for point cloud registration," in *Int. Conf. on Computer Vision*, 2019, pp. 3522–3531.

[7] Z. Qin, H. Yu, C. Wang, Y. Guo, Y. Peng, and K. Xu, "Geometric transformer for fast and robust point cloud registration," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, 2022, pp. 11 143–11 152.

[8] Z. J. Yew and G. H. Lee, "REGTR: End-to-end point cloud correspondences with transformers," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, June 2022, pp. 6677–6686.

[9] Z. Zhang, "Iterative point matching for registration of free-form curves and surfaces," *Int. Journal of Computer Vision*, vol. 13, no. 2, pp. 119–152, 1994.

[10] X. Chen, A. Milioto, E. Palazzolo, P. Giguère, J. Behley, and C. Stachniss, "SuMa++: Efficient LiDAR-based semantic SLAM," in *Int. Conf. on Intelligent Robots and Systems*, 2019, pp. 4530–4537.

[11] X. Kong, X. Yang, G. Zhai, X. Zhao, X. Zeng, M. Wang, Y. Liu, W. Li, and F. Wen, "Semantic graph based place recognition for 3D point clouds," in *Int. Conf. on Intelligent Robots and Systems*, 2020, pp. 8216–8223.

[12] N. Radwan, A. Valada, and W. Burgard, "VLocNet++: Deep multitask learning for semantic visual localization and odometry," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 4407–4414, 2018.

[13] Z. Yuan, K. Xu, B. Deng, X. Zhou, P. Chen, and Y. Ma, "SV-Loop: Semantic-visual loop closure detection with panoptic segmentation," in *2021 IEEE 6th International Conference on Signal and Image Processing (ICSIP)*, 2021, pp. 245–250.

[14] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, 2012, pp. 3354–3361.

[15] G. Pandey, J. R. McBride, and R. M. Eustice, "Ford campus vision and lidar data set," *The International Journal of Robotics Research*, vol. 30, no. 13, pp. 1543–1552, 2011.

[16] R. B. Rusu, N. Blodow, and M. Beetz, "Fast point feature histograms (FPFH) for 3D registration," in *Int. Conf. on Robotics and Automation*, 2009, pp. 3212–3217.

[17] B. Steder, R. B. Rusu, K. Konolige, and W. Burgard, "Point feature extraction on 3D range scans taking into account object boundaries," in *Int. Conf. on Robotics and Automation*, 2011, pp. 2601–2608.

[18] L. Luo, S.-Y. Cao, Z. Sheng, and H.-L. Shen, "LiDAR-based global localization using histogram of orientations of principal normals," *IEEE Transactions on Intelligent Vehicles*, pp. 1–1, 2022.

[19] L. He, X. Wang, and H. Zhang, "M2DP: A novel 3D point cloud descriptor and its application in loop closure detection," in *Int. Conf. on Intelligent Robots and Systems*, 2016, pp. 231–237.

[20] G. Kim and A. Kim, "Scan Context: Egocentric spatial descriptor for place recognition within 3D point cloud map," in *Int. Conf. on Intelligent Robots and Systems*, 2018, pp. 4802–4809.

[21] H. Wang, C. Wang, and L. Xie, "Intensity Scan Context: Coding intensity and geometry relations for loop closure detection," in *Int. Conf. on Robotics and Automation*, 2020, pp. 2095–2101.

[22] L. Li, X. Kong, X. Zhao, T. Huang, W. Li, F. Wen, H. Zhang, and Y. Liu, "SSC: Semantic scan context for large-scale place recognition," in *Int. Conf. on Intelligent Robots and Systems*, 2021, pp. 2092–2099.

[23] M. Angelina Uy and G. Hee Lee, "PointNetVLAD: Deep point cloud based retrieval for large-scale place recognition," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, 2018.

[24] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep learning on point sets for 3D classification and segmentation," *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, 2017.

[25] X. Chen, T. Läbe, A. Milioto, T. Röhling, J. Behley, and C. Stachniss, "OverlapNet: A siamese network for computing lidar scan similarity with applications to loop closing and localization," *Autonomous Robots*, 2021.

[26] J. Ma, J. Zhang, J. Xu, R. Ai, W. Gu, and X. Chen, "OverlapTransformer: An efficient and yaw-angle-invariant transformer network for LiDAR-based place recognition," *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 6958–6965, 2022.

[27] S. Bouaziz, A. Tagliasacchi, and M. Pauly, "Sparse iterative closest point," in *Computer graphics forum*, vol. 32, no. 5. Wiley Online Library, 2013, pp. 113–123.

[28] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[29] Y. Aoki, H. Goforth, R. A. Srivatsan, and S. Lucey, "PointNetLK: Robust & efficient point cloud registration using PointNet," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, 2019.

[30] A. L. Ballardini, D. Cattaneo, and D. G. Sorrenti, "Visual localization at intersections with digital maps," in *Int. Conf. on Robotics and Automation*, 2019, pp. 6651–6657.

[31] L. Li, X. Kong, X. Zhao, T. Huang, W. Li, F. Wen, H. Zhang, and Y. Liu, "RINet: Efficient 3D lidar-based place recognition using rotation invariant neural network," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4321–4328, 2022.

[32] S. Shi, C. Guo, L. Jiang, Z. Wang, J. Shi, X. Wang, and H. Li, "PV-RCNN: Point-voxel feature set abstraction for 3D object detection," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, 2020, pp. 10 526–10 535.

[33] R. Arandjelović, P. Gronat, A. Torii, T. Pajdla, and J. Sivic, "NetVLAD: CNN architecture for weakly supervised place recognition," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 40, no. 6, pp. 1437–1451, 2018.

[34] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, vol. 30, 2017.

[35] G. Kim, S. Choi, and A. Kim, "Scan Context++: Structural place recognition robust to rotation and lateral variations in urban environments," *IEEE Transactions on Robotics*, vol. 38, no. 3, pp. 1856–1874, 2022.

[36] Y. Wang, Z. Sun, C.-Z. Xu, S. E. Sarma, J. Yang, and H. Kong, "LiDAR Iris for loop-closure detection," in *Int. Conf. on Intelligent Robots and Systems*, 2020, pp. 5769–5775.

[37] 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 *Int. Conf. on Computer Vision*, 2019.

[38] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The Cityscapes dataset for semantic urban scene understanding," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, 2016, pp. 3213–3223.

## 7.7 Continual SLAM: Beyond Lifelong Simultaneous Localization and Mapping through Continual Learning

The appended paper [112] follows.

# Continual SLAM: Beyond Lifelong Simultaneous Localization and Mapping through Continual Learning

Niclas Vödisch[1], Daniele Cattaneo[1], Wolfram Burgard[2], and Abhinav Valada[1]

**Abstract** Robots operating in the open world encounter various different environments that can substantially differ from each other. This domain gap also poses a challenge for Simultaneous Localization and Mapping (SLAM) being one of the fundamental tasks for navigation. In particular, learning-based SLAM methods are known to generalize poorly to unseen environments hindering their general adoption. In this work, we introduce the novel task of continual SLAM extending the concept of lifelong SLAM from a single dynamically changing environment to sequential deployments in several drastically differing environments. To address this task, we propose CL-SLAM leveraging a dual-network architecture to both adapt to new environments and retain knowledge with respect to previously visited environments. We compare CL-SLAM to learning-based as well as classical SLAM methods and show the advantages of leveraging online data. We extensively evaluate CL-SLAM on three different datasets and demonstrate that it outperforms several baselines inspired by existing continual learning-based visual odometry methods. We make the code of our work publicly available at http://continual-slam.cs.uni-freiburg.de.

## 1 Introduction

An essential task for an autonomous robot deployed in the open world without prior knowledge about its environment is to perform Simultaneous Localization and Mapping (SLAM) to facilitate planning and navigation [11, 27]. To address this task, various SLAM algorithms based on different sensors have been proposed, including classical methods [28] and learning-based approaches [3, 18]. Classical methods typically rely on handcrafted low-level features that tend to fail under challenging conditions, e.g., textureless regions. Deep learning-based approaches mitigate such problems due to their ability to learn high-level features. However, they lack the ability to generalize to out-

[1]Department of Computer Science, University of Freiburg, Germany,
[2]Department of Engineering, University of Technology Nuremberg, Germany
This work was funded by the European Union's Horizon 2020 research and innovation program under grant agreement No 871449-OpenDR.

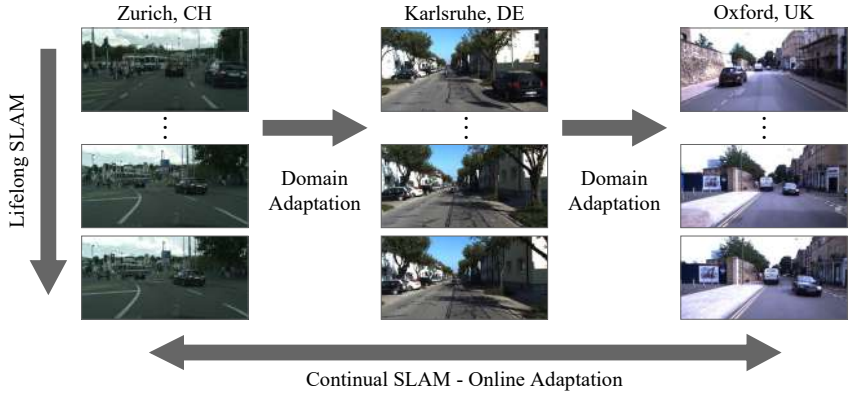



Fig. 1: While lifelong SLAM considers the long-term operation of a robot in a single dynamically changing environment, domain adaptation techniques aim toward transferring knowledge gained in one environment to another environment. The newly defined task of continual SLAM extends both settings by requiring omnidirectional adaptation involving multiple environments. Agents have to both quickly adapt to new environments and effectively recall knowledge from previously visited environments.

of-distribution data, with respect to the training set. For visual SLAM, such out-of-distribution data can correspond to images sourced from cities in different countries or under substantially different conditions. In the following, we use the term *environment* to refer to a bounded geographical area. While different environments can share the same fundamental structure, e.g., urban areas, their specific characteristics prevent the seamless transfer of learned features, resulting in a domain gap between cities [1].

In the context of this work, lifelong SLAM [31] considers the long-term operation of a robot in a single environment (see Fig. 1). Although this environment can be subject to temporal changes, the robot is constrained to stay within the area borders [14], e.g., to obtain continuous map updates [15] within a city. Recent works attempt to relax this assumption by leveraging domain adaptation techniques for deep neural networks, including both regularization [33] and online adaptation of the employed model [19, 20, 23]. While a naive solution for adapting to a new environment is to source additional data, this is not feasible when the goal is to ensure the uninterrupted operation of the robot. Moreover, changes within an environment can be sudden, e.g., rapid weather changes, and data collection and annotation often come at a high cost. Therefore, adaptation methods should be trainable in an unsupervised or self-supervised manner without the need for ground truth data. As illustrated in Fig. 1, the setting addressed in domain adaptation only considers unidirectional knowledge transfer from a single known to a single unknown environment [1] and thus does not represent the open world, where the number of new environments that a robot can encounter is infinite and previously seen environments can be revisited. To address this gap, we take the next step by considering more complex sequences of environments and formulate the novel task of continual SLAM that leverages insights from both continual learning (CL) and lifelong SLAM. We propose a dual-network architecture called CL-SLAM to balance adaptation to new environments and memory retention of preceding environments. To assess its efficacy,

we define two metrics, adaptation quality and retention quality, and compare CL-SLAM to several baselines inspired by existing CL-based VO methods involving three different environments. We make the code of this work publicly available at http://continual-slam.cs.uni-freiburg.de. The supplementary material can be found at https://arxiv.org/abs/2203.01578.

## 2 Related Work

*Visual Odometry / SLAM:* Visual odometry (VO) and vision-based SLAM estimate camera motion from a video. Allowing for self-supervised training, monocular VO can be tackled jointly with depth estimation based on photometric consistency. SfMLearner [34] uses an end-to-end approach consisting of two networks to predict depth from a single image and camera motion from two consecutive images. The networks are trained in parallel by synthesizing novel views of the target image. Monodepth2 [6] extends the loss function to account for occluded and static pixels. Other works such as DF-VO [32] eliminate the need for a pose network by leveraging feature matching based on optical flow. While these methods show superior performance [20], computing a gradient of the predicted pose with respect to the input image is not possible using classic point matching algorithms. To reduce drift, DeepSLAM [18] combines unsupervised learning-based VO with a pose graph backend taking global loop closures into account. In this work, we use a trainable pose network with velocity supervision [8] to resolve scale ambiguity. Similar to DeepSLAM, we detect loop closures and perform graph optimization.

*Continual Learning:* Traditionally, a learning-based model is trained for a specific task on a dedicated training set and then evaluated on a hold-out test set sampled from the same distribution. However, in many real-world applications, the data distributions can differ or even change over time. Additionally, the initial task objective might be altered. Continual learning (CL) and lifelong learning [31] address this problem by defining a paradigm where a model is required to continuously readjust to new tasks and/or data distributions without sacrificing the capability to solve previously learned tasks, thus avoiding catastrophic forgetting. Most CL approaches employ one of three strategies. First, experience replay includes rehearsal and generative replay. Rehearsal refers to reusing data samples of previous tasks during adaptation to new tasks, e.g., the replay buffer in CoMoDA [16]. Minimizing the required memory size, the most representative samples should be carefully chosen or replaced by more abstract representations [7]. Similarly, generative replay constructs artificial samples by training generative models [30]. Second, regularization [21] prevents a CL algorithm from overfitting to the new tasks to mitigate forgetting, e.g., knowledge distillation. Third, architectural methods [13] preserve knowledge by adding, duplicating, freezing, or storing parts of the internal model parameters. They further include dual architectures that are inspired by mammalian brains [25], where one model learns the novel task and a second model memorizes previous experience. In this work, we combine architectural and replay strategies by leveraging a dual-network architecture with online adaptation incorporating data rehearsal.

*Online Adaptation for Visual Odometry and Depth Estimation:* Recently, Luo *et al.* [23] employed a subtask of CL for self-supervised VO and depth estimation, opening a new avenue of research. Online adaptation enables these methods to enhance the trajectory and depth prediction on a test set sourced from a different data distribution than the originally used training set. Both Zhang *et al.* [33] and CoMoDA [16] primarily target the depth estimation task. While Zhang *et al.* propose to learn an adapter to map the distribution of the online data to the one of the training data, CoMoDA updates the internal parameters of the depth and pose networks based on online data and a replay buffer. The work in spirit most similar to ours is done by Li *et al.* [19]. They propose to substitute the standard convolutional layers in the depth and pose networks with convolutional LSTM variants. Then, the model parameters are continuously updated using only the online data. In subsequent work, Li *et al.* [20] replace the learnable pose network by point matching from optical flow. Note that all existing works purely focus on one-step adaptation, i.e., transferring knowledge gained in one environment to a single new environment. In this paper, we introduce continual SLAM to take the next step by considering more complex deployment scenarios comprising more than two environments and further alternating between them.

## 3 Continual SLAM

**Problem Setting:** Deploying a SLAM system in the open world substantially differs from an experimental setting, in which parameter initialization and system deployment are often performed in the same environment. To overcome this gap, we propose a new task called *Continual SLAM*, illustrated in Fig. 1, where the robot is deployed on a sequence of diverse scenes from different environments.

Ideally, a method addressing the continual SLAM problem should be able to achieve the following goals: 1) quickly adapt to unseen environments while deployment, 2) leverage knowledge from previously seen environments to speed up the adaptation, and 3) effectively memorize knowledge from previously seen environments to minimize the required adaptation when revisiting them, while mitigating overfitting to any of the environments. Formally, continual SLAM can be defined as a potentially infinite sequence of scenes $\mathcal{S} = (s_1 \rightarrow s_2 \rightarrow \dots)$ from a set of different environments $s_i \in \{E_a, E_b, \dots\}$, where $s$ denotes a scene and $E$ denotes an environment. In particular, $\mathcal{S}$ can contain multiple scenes from the same environment and the scenes in $\mathcal{S}$ can occur in any possible fixed order. A continual SLAM algorithm $\mathcal{A}$ can be defined as

$$\mathcal{A} : < \theta_{i-1}, (s_1, \dots, s_i) > \mapsto < \theta_i >, \tag{1}$$

where $(s_1, \dots, s_i)$ refers to the seen scenes in the specified order and $\theta_i$ denotes the corresponding state of the learnable parameters of the algorithm. During deployment, the algorithm $\mathcal{A}$ has to update $\theta_{i-1}$ based on the newly encountered scene $s_i$. For instance, given two environments $E_a = \{s_a^1, s_a^2\}$ and $E_b = \{s_b^1\}$, which comprise two and one scenes, respectively, examples of feasible sequences are

$$\mathcal{S}_1 = (s_a^1 \rightarrow s_b^1 \rightarrow s_a^2), \quad \mathcal{S}_2 = (s_a^2 \rightarrow s_a^1 \rightarrow s_b^1), \quad \mathcal{S}_3 = (s_b^1 \rightarrow s_a^2 \rightarrow s_a^1), \tag{2}$$

where the scene subscripts denote the corresponding environment and the superscripts refer to the scene ID in this environment. As described in Sec. 1, the task of continual SLAM is substantially different from lifelong SLAM or unidirectional domain adaptation as previously addressed by Luo *et al.* [23] and Li *et al.* [19, 20].

To conclude, we identify the following main challenges: 1) large number of different environments, 2) huge number of chained scenes, 3) scenes can occur in any possible order, and 4) environments can contain multiple scenes. Therefore, following the spirit of continual learning (CL), a continual SLAM algorithm has to balance between short-term adaptation to the current scene and long-term knowledge retention. This trade-off is also commonly referred to as avoiding catastrophic forgetting with respect to previous tasks without sacrificing performance on the new task at hand.

**Performance Metrics:** To address the aforementioned challenges, we propose two novel metrics, namely adaptation quality (AQ), which measures the short-term adaptation capability when being deployed in a new environment, and retention quality (RQ), which captures the long-term memory retention when revisiting a previously encountered environment. In principle, these metrics can be applied to any given base metric $M_d$ that can be mapped to the interval $[0, 1]$, where 0 and 1 are the lowest and highest performances, respectively. The subscript $d$ denotes the given sequence, where the error is computed on the final scene.

*Base Metrics:* For continual SLAM, we leverage the translation error $t_{err}$ (in %) and the rotation error $r_{err}$ (in °/m), proposed by Geiger *et al.* [5], that evaluate the error as a function of the trajectory length. To obtain scores in the interval $[0, 1]$, we apply the following remapping:

$$\widehat{t}_{err} = \max\left(0, 1 - \frac{t_{err}}{100}\right), \quad \widehat{r}_{err} = 1 - \frac{r_{err}}{180}, \tag{3}$$

where we clamp $\widehat{t}_{err}$ to 0 for $t_{err} > 100\%$. The resulting $\widehat{t}_{err}$ and $\widehat{r}_{err}$ are then used as the base metric $M$ to compute $\mathrm{AQ}_{\mathrm{trans}}$ / $\mathrm{RQ}_{\mathrm{trans}}$ and $\mathrm{AQ}_{\mathrm{rot}}$ / $\mathrm{RQ}_{\mathrm{rot}}$, respectively.

*Adaptation Quality:* The adaptation quality (AQ) measures the ability of a method to effectively adapt to a new environment based on experiences from previously seen environments. It is inspired by the concept of forward transfer (FWT) [22] in traditional CL, which describes how learning a current task influences the performance of a future task. Particularly, positive FWT enables zero-shot learning, i.e., performing well on a future task without explicit training on it. On the other hand, negative FWT refers to sacrificing performance on a future task by learning the current task. In our context, a task refers to performing SLAM in a given environment. Consequently, the AQ is intended to report how well a continual SLAM algorithm is able to minimize negative FWT, e.g., by performing online adaptation.

To illustrate the AQ, we consider the simplified example of a set of two environments $\{E_a, E_b\}$ consisting of different numbers of scenes. We further assume that the algorithm has been initialized in a separate environment $E_p$. Since the AQ focuses on the cross-environment adaptation, we sample one random scene from each environment $s_a \in E_a$ and $s_b \in E_b$ and hold them fixed. Now, we construct the set of all possible deployment

sequences $\mathcal{D} = \big\{(s_p \rightarrow s_a), (s_p \rightarrow s_b), (s_p \rightarrow s_a \rightarrow s_b), (s_p \rightarrow s_b \rightarrow s_a)\big\}$, where $s_p \in E_p$ is the data used for initialization. The AQ is then defined as:

$$\text{AQ} = \frac{1}{|\mathcal{D}|} \sum_{d \in \mathcal{D}} M_d. \tag{4}$$

*Retention Quality:* To further account for the opposing challenge of the continual SLAM setting, we propose the retention quality (RQ) metric. It measures the ability of an algorithm to preserve long-term knowledge when being redeployed in a previously encountered environment. It is inspired by the concept of backward transfer (BWT) [22] in CL settings, which describes how learning a current task influences the performance on a previously learned task. While positive BWT refers to improving the performance on prior tasks, negative BWT indicates a decrease in the performance of the preceding task. The extreme case of a large negative BWT is often referred to as catastrophic forgetting. Different from classical BWT, we further allow renewed online adaptation when revisiting a previously seen environment, i.e., performing a previous task, as such a setting is more sensible for a robotic setup. It further avoids the necessity to differentiate between new and already seen environments, which would require the concept of environment classification in the open world.

To illustrate the RQ, we consider a set of two environments $\{E_a, E_b\}$ consisting of different numbers of scenes. We further assume that the algorithm has been initialized on data $s_p$ of a separate environment $E_p$. We sample two random scenes from each environment, i.e., $s_a^1$, $s_a^2$, $s_b^1$, and $s_b^2$. To evaluate the RQ, we need to construct a deployment sequence $S$ that consists of alternating scenes from the two considered environments. In this example, we consider the following fixed sequence:

$$S = (s_p \rightarrow s_a^1 \rightarrow s_b^1 \rightarrow s_a^2 \rightarrow s_b^2). \tag{5}$$

We then consider all the subsequences $\mathcal{D}$ of $S$ in which the last scene comes from an environment already visited prior to a deployment in a scene of a different environment. In this example, $\mathcal{D} = \big\{(s_p \rightarrow s_a^1 \rightarrow s_b^1 \rightarrow s_a^2), (s_p \rightarrow s_a^1 \rightarrow s_b^1 \rightarrow s_a^2 \rightarrow s_b^2)\big\}$.

The RQ is then defined as the sum over all differences of the base metric in a known environment before and after deployment in a new environment, divided by the size of $\mathcal{D}$. For instance, given the sequence in Eq. 5:

$$\text{RQ} = \frac{1}{2}\left(M_{s_p \rightarrow s_a^1 \rightarrow s_b^1 \rightarrow s_a^2} - M_{s_p \rightarrow s_a^1 \rightarrow s_a^2} + M_{s_p \rightarrow s_a^1 \rightarrow s_b^1 \rightarrow s_a^2 \rightarrow s_b^2} - M_{s_p \rightarrow s_a^1 \rightarrow s_b^1 \rightarrow s_b^2}\right). \tag{6}$$

## 4 Technical Approach

**Framework Overview:** The core of CL-SLAM is the dual-network architecture of the visual odometry (VO) model that consists of an *expert* that produces myopic online odometry estimates and a *generalizer* that focuses on the long-term learning across environments (see Fig. 2). We train both networks in a self-supervised manner where the weights of the expert are updated only based on online data, whereas the weights of the generalizer are updated based on a combination of data from both the online stream

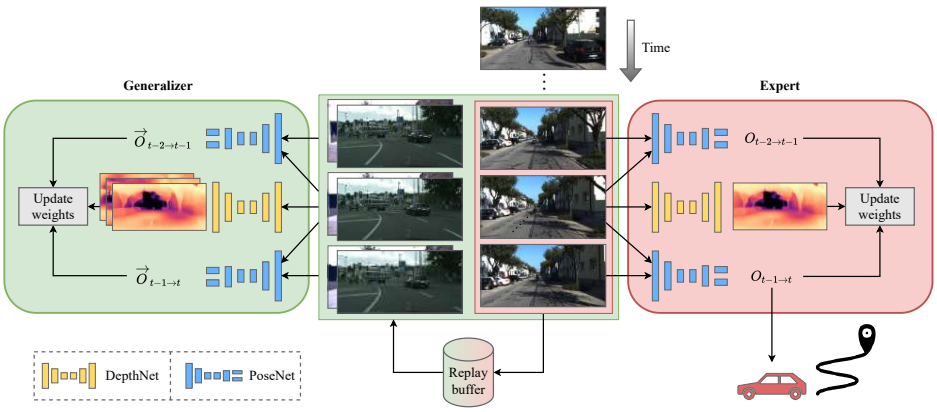



Fig. 2: Online adaptation scheme of our proposed CL-SLAM that is constructed as a dual-network architecture including a generalizer (left) and an expert (right). While the expert focuses on the short-term adaptation to the current scene, the generalizer avoids catastrophic forgetting by employing a replay buffer comprising samples from the past and the present. Note that both subnetworks contain a single PoseNet, shown twice to reflect pose estimation at different steps. The predicted odometry $O_{t-1\to t}$ is sent to the SLAM framework as shown in Fig. 3.

and a replay buffer. We use the VO estimates of the expert to construct a pose graph (see Fig. 3). To reduce drift, we detect global loop closures and add them to the graph, which is then optimized. Finally, we can create a dense 3D map using the depth predicted by the expert and the optimized path.

**Visual Odometry:** We generate VO estimates following the commonly used approach of using a trainable pose network [2, 6, 8, 18] for self-supervised depth estimation with a stream of monocular images. The basic idea behind this approach is to synthesize a novel view of an input image using image warping as reviewed in the supplementary material.

In this work, we use Monodepth2 [6] to jointly predict the depth map of an image and the camera motion from the previous timestep to the current. To recover metric scaling of both depth and the odometry estimates, we adapt the original loss function with a velocity supervision term as proposed by Guizilini *et al.* [8]. As scalar velocity measurements are commonly available in robotic systems, e.g., by wheel odometry, this does not pose an additional burden. Our total loss is composed of the photometric reprojection loss $\mathcal{L}_{pr}$, the image smoothness loss $\mathcal{L}_{sm}$, and the velocity supervision loss $\mathcal{L}_{vel}$:

$$\mathcal{L} = \mathcal{L}_{pr} + \gamma \mathcal{L}_{sm} + \lambda \mathcal{L}_{vel}. \tag{7}$$

Following the common methodology, we compute the loss based on an image triplet $\{\mathbf{I_{t-2}}, \mathbf{I_{t-1}}, \mathbf{I_t}\}$ using depth and odometry predictions $\mathbf{D_{t-1}}$, $\mathbf{O_{t-2\to t-1}}$, and $\mathbf{O_{t-1\to t}}$. We provide more details on the individual losses in the supplementary material.

**Loop Closure Detection and Pose Graph Optimization:** In order to reduce drift over time, we include global loop closure detection and pose graph optimization (see Fig. 3). We perform place recognition using a pre-trained and frozen CNN, referred to as LoopNet. In particular, we map every frame to a feature vector using MobileNetV3 small [10], trained on ImageNet, and store them in a dedicated memory. Then, we compute the

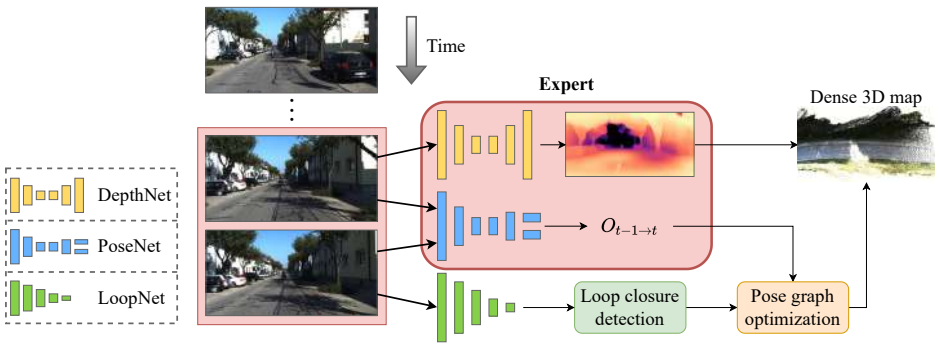



Fig. 3: Full SLAM framework of our proposed CL-SLAM. Global loop closures are detected with a pre-trained CNN. Visual odometry estimates between both consecutive frames and loop closure frames are generated by the PoseNet and added to a pose graph, which is optimized upon the detection of a new loop closure. Finally, a dense 3D map can be created using the predicted depth and the optimized path.

cosine similarity of the current feature map with all preceding feature maps:

$$\text{sim}_{\cos} = \cos(f_{current}, f_{previous}). \tag{8}$$

If $\text{sim}_{\cos}$ is above a given threshold, we use the PoseNet to compute the transformation between the corresponding images. During deployment, we continuously build a pose graph [17] consisting of both local and global connections, i.e., consecutive VO estimates and loop closures. Whenever a new loop closure is detected, the pose graph is optimized.

**Online Adaptation:** In this section, we describe the dual-network architecture of the VO predictor in CL-SLAM that effectively addresses the trade-off between short-term adaptation and long-term memory retention, a problem also known as catastrophic forgetting. Subsequently, we detail the training scheme including the utilized replay buffer.

*Architecture:* The dual-network architecture consists of two instances of both the DepthNet and the PoseNet. In the following, we refer to these instances as *expert* and *generalizer*. We build upon the architecture of Monodepth2 [6]. The DepthNet has an encoder-decoder topology, comprising a ResNet-18 [9] encoder and a CNN-based decoder with skip connections, and predicts disparity values for each pixel in the input image. The PoseNet consists of a similar structure using a separate ResNet-18 encoder followed by additional convolutional layers to generate the final output that represents translation and rotation between two input images. Further implementation details are provided in Sec. 5.

*Training Scheme:* Before deployment, i.e., performing continual adaptation, we pre-train the DepthNet and the PoseNet using the standard self-supervised training procedure based on the loss functions described in Sec. 4. When deployed in a new environment, we continuously update the weights of both the expert and the generalizer in an online manner, following a similar scheme as Kuznietsov *et al.* [16]:

(1) Create an image triplet composed of the latest frame $\mathbf{I_t}$ and the two previous frames $\mathbf{I_{t-1}}$ and $\mathbf{I_{t-2}}$. Similarly, batch the corresponding velocity measurements.
(2) Estimate the camera motion between both pairs of subsequent images, i.e., $\mathbf{O_{t-2\to t-1}}$ and $\mathbf{O_{t-1\to t}}$ with the PoseNet.

(3) Generate the depth estimate $\mathbf{D_{t-1}}$ of the previous image with the DepthNet.
(4) Compute the loss according to Eq. 7 and use backpropagation to update the weights of the DepthNet and PoseNet.
(5) Loop over steps (2) to (4) for $c$ iterations.
(6) Repeat the previous steps for the next image triplet.

Upon deployment, both the expert and the generalizer are initialized with the same set of parameter weights, initially obtained from pre-training and later replaced by the memory of the generalizer. As illustrated in Fig. 2, the weights of the expert are updated according to the aforementioned algorithm. Additionally, every new frame from the online image stream is added to a replay buffer along with the corresponding velocity reading. Using only the online images, the expert will quickly adapt to the current environment. This behavior can be described as a desired form of overfitting for a myopic increase in performance. On the other hand, the generalizer acts as the long-term memory of CL-SLAM circumventing the problem of catastrophic forgetting in continual learning settings. Here, in step (1), we augment the online data by adding image triplets from the replay buffer to rehearse experiences made in the past, as depicted in Fig. 2. After deployment, the weights of the stored parameters used for initialization are replaced by the weights of the generalizer, thus preserving the continuous learning process of CL-SLAM. The weights of the expert are then discarded.

## 5 Experimental Evaluation

**Implementation Details:** We adopt the Monodepth2 [6] architecture using separate ResNet-18 [9] encoders for our DepthNet and PoseNet. We implement CL-SLAM in PyTorch [29] and train on a single NVIDIA TITAN X GPU. We pre-train both sub-networks in a self-supervised manner on the Cityscapes dataset [4] for 25 epochs with a batch size of 18. We employ the Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$ and an initial learning rate of $10^{-4}$, which is reduced to $10^{-5}$ after 15 epochs. Further, we resize all images during both pre-training and adaptation to $192 \times 640$ pixels. Additionally, during the pre-training phase, we mask all potentially dynamic objects using bounding boxes generated by YOLOv5m [12], which was trained on the COCO dataset. We observe that on Cityscapes this procedure yields a smaller validation loss than without masking. We set the minimum predictable depth to 0.1 m without specifying an upper bound. To balance the separate terms in the loss, we set the disparity smoothness weight $\gamma = 0.001$ and the velocity loss weight $\lambda = 0.05$.

During adaptation, we utilize the same hyperparameters as listed above. Inspired by the findings of McCraith *et al.* [26], we freeze the weights of the encoders. Based on the ablation study in Sec. 5.2, we set the number of update cycles $c = 5$. To enhance the unsupervised guidance, we use the velocity readings to skip new incoming images if the driven distance is less than 0.2 m. We construct the training batch for the generalizer by concatenating the online data with a randomly sampled image triplet of each environment except for the current environment as this is already represented by the online data. Finally, we add the online data to the replay buffer.

Table 1: Path accuracy on the KITTI dataset

| KITTI sequence | Online adaptation to KITTI | | | | Trained on KITTI seq. {0, 1, 2, 8, 9} | | | | No training | |
|---|---|---|---|---|---|---|---|---|---|---|
| | CL-SLAM | | CL-SLAM (w/o loops) | | DeepSLAM [18] | | VO+vel [6, 8] (w/o loops) | | ORB-SLAM | |
| | $t_{err}$ | $r_{err}$ | $t_{err}$ | $r_{err}$ | $t_{err}$ | $r_{err}$ | $t_{err}$ | $r_{err}$ | $t_{err}$ | $r_{err}$ |
| 4 | – | – | **4.37** | **0.51** | 5.22 | 2.27 | 10.72 | 1.69 | 0.62 | 0.11 |
| 5 | 4.30 | **1.01** | 4.41 | 1.33 | **4.04** | 1.40 | 34.55 | 11.88 | 2.51 | 0.25 |
| 6 | **2.53** | **0.63** | 3.07 | 0.73 | 5.99 | 1.54 | 15.20 | 5.62 | 7.80 | 0.35 |
| 7 | **2.10** | **0.83** | 3.74 | 1.91 | 4.88 | 2.14 | 12.77 | 6.80 | 1.53 | 0.35 |
| 10 | – | – | **2.22** | **0.34** | 10.77 | 4.45 | 55.27 | 9.50 | 2.96 | 0.52 |

Translation error $t_{err}$ in [%] and rotation error $r_{err}$ in [°/100m]. Sequences 4 and 10 do not contain loops. CL-SLAM is pre-trained on the Cityscapes dataset. The paths computed by ORB-SLAM use median scaling [34] as they are not metric scale. The smallest errors among the learning-based methods are shown in bold.

**Datasets:** To simulate scenes from a diverse set of environments, we employ our method on three relevant datasets, namely Cityscapes [4], Oxford RobotCar [24], and KITTI [5], posing the additional challenge of adapting to changing camera characteristics.

*Cityscapes:* The Cityscapes Dataset [4] includes images and vehicle metadata recorded in 50 cities across Germany and bordering regions. Due to the unsupervised training scheme of our VO method, we can leverage the included 30-frame snippets to pre-train our networks despite the lack of ground truth poses.

*Oxford RobotCar:* The Oxford RobotCar Dataset [24] focuses on repeated data recordings of a consistent route, captured over the period of one year in Oxford, UK. Besides RGB images, it also contains GNSS and IMU data, which we use for velocity supervision. To compute the trajectory error, we leverage the released RTK ground truth positions.

*KITTI:* The KITTI Dataset [5] provides various sensor recordings taken in Karlsruhe, Germany. We utilize the training data from the odometry benchmark, which includes images and ground truth poses for multiple routes. We further leverage the corresponding IMU data from the released raw dataset to obtain the velocity of the vehicle.

## 5.1 Evaluation of Pose Accuracy of CL-SLAM

Before analyzing how CL-SLAM addresses the task of continual SLAM, we compare its performance to existing SLAM framework. In particular, in Table 1 we report the translation and rotation errors on sequences 4, 5, 6, 7, and 10 of the KITTI Odometry dataset [5] following Li *et al.* [18]. Since the IMU data of sequence 3 has not been released, we omit this sequence. We compare CL-SLAM to two learning-based and one feature-based approach. DeepSLAM [18] uses a similar unsupervised learning-based approach consisting of VO and graph optimization but does not perform online adaptation. VO+vel refers to Monodepth2 [6] with velocity supervision [8], i.e., it corresponds to the base VO estimator of CL-SLAM without adaptation and loop closure detection. Both learning-based methods produce metric scale paths and are trained on the sequences 0, 1, 2, 8, and 9. Further, we report the results of monocular ORB-SLAM [28] after median scaling [34].

Table 2: Translation and rotation error for computing the AQ and RQ metrics

| Used for | Previous scenes | Current scene | $\mathcal{B}_{fixed}$ | | $\mathcal{B}_{expert}$ | | $\mathcal{B}_{general}$ | | CL-SLAM | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | $t_{err}$ | $r_{err}$ | $t_{err}$ | $r_{err}$ | $t_{err}$ | $r_{err}$ | $t_{err}$ | $r_{err}$ |
| AQ | $c_t$ | $k_1$ | 130.74 | 26.35 | **2.50** | **0.37** | 7.21 | 1.26 | **2.50** | **0.37** |
| | $c_t$ | $r_1$ | 170.76 | 13.37 | **28.94** | 5.63 | 29.05 | **5.49** | **28.94** | 5.63 |
| | $c_t \rightarrow r_1$ | $k_1$ | – | – | 3.66 | 0.73 | 14.14 | 1.79 | **3.24** | **0.54** |
| | $c_t \rightarrow k_1$ | $r_1$ | – | – | 32.56 | 6.08 | 34.79 | 6.64 | **30.13** | **5.87** |
| RQ | $c_t \rightarrow k_1 \rightarrow r_1$ | $k_2$ | 164.77 | 25.07 | 45.20 | 5.62 | 8.48 | 1.79 | **4.85** | **1.59** |
| | $c_t \rightarrow k_1 \rightarrow r_1 \rightarrow k_2$ | $r_2$ | 200.14 | 28.94 | **15.91** | 4.93 | 16.02 | 4.98 | 20.50 | **4.77** |
| | $c_t \rightarrow k_1$ | $k_2$ | – | – | 15.82 | 2.50 | 9.37 | 2.21 | **7.48** | **1.63** |
| | $c_t \rightarrow k_1 \rightarrow r_1$ | $r_2$ | – | – | 14.89 | 4.62 | **12.24** | **4.38** | 16.41 | 4.58 |

The *previous scenes* denote the scenes that have been used for previous training of the algorithm, the *current scene* denotes the evaluation scene to compute both errors $t_{err}$ in [%] and $r_{err}$ in [°/100m]. $c_t$ refers to the Cityscapes training set. $r_i$ and $k_i$ are sequences from KITTI and the Oxford RobotCar dataset. Bold and underlined values indicate the best and second best scores on each sequence.

CL-SLAM outperforms DeepSLAM on the majority of sequences highlighting the advantage of online adaptation. Note that CL-SLAM was not trained on KITTI data but was only exposed to Cityscapes before deployment. To show the effect of global loop closure detection, we report the error on sequences 5 to 7 both with and without graph optimization enabled. Note that sequences 4 and 10 do not contain loops. Compared to ORB-SLAM, CL-SLAM suffers from a higher rotation error but can improve the translation error in sequences 6 and 10. The overall results indicate that general SLAM methods would benefit from leveraging online information to enhance performance.

## 5.2 Evaluation of Continual SLAM

**Experimental Setup:** In order to quantitatively evaluate the performance of our proposed approach, we compute both the adaptation quality (AQ) and the retention quality (RQ) by deploying CL-SLAM and the baseline methods on a fixed sequence of scenes. In particular, we use the official training split of the Cityscapes dataset to initialize the DepthNet and PoseNet, using the parameters detailed in Sec. 5. The pre-training step is followed by a total of four scenes of the Oxford RobotCar dataset and the KITTI dataset.

$$(c_t \rightarrow k_1 \rightarrow r_1 \rightarrow k_2 \rightarrow r_2), \tag{9}$$

where $c_t$ refers to the Cityscapes training set.

Following the setup of Li *et al.* [19], we set $k_1$ and $k_2$ to be sequences 9 and 10 of the KITTI Odometry dataset. Note that we omit loop closure detection for this evaluation to prevent graph optimization from masking the effect of the respective adaptation technique. From the Oxford RobotCar dataset, we select the recording of August 12, 2015, at 15:04:18 GMT due to sunny weather and good GNSS signal reception. In detail, we set $r_1$ to be the scene between frames 750 and 4,750 taking every second frame to increase the driven distance between two consecutive frames. Analogously, we set $r_2$ to be the scene between frames 22,100 and 26,100. We use a scene length of 2,000 frames in order to be similar to the length of KITTI sequences: 1,584 frames for $k_1$ and 1,196 for $k_2$.

Table 3: Comparison of the Adaptation Quality (AQ)

| | ↑ $AQ_{trans}$ | ↑ $AQ_{rot}$ |
|---|---|---|
| $\mathcal{B}_{fixed}$ | 0.000 | 0.890 |
| $\mathcal{B}_{expert}$ | <u>0.831</u> | <u>0.982</u> |
| $\mathcal{B}_{general}$ | 0.787 | 0.979 |
| CL-SLAM | **0.848** | **0.983** |

$AQ_{trans}$ refers to adaptation quality with respect to the translation error, $AQ_{rot}$ is based on the rotation error. Bold and underlined values denote the best and second best scores.

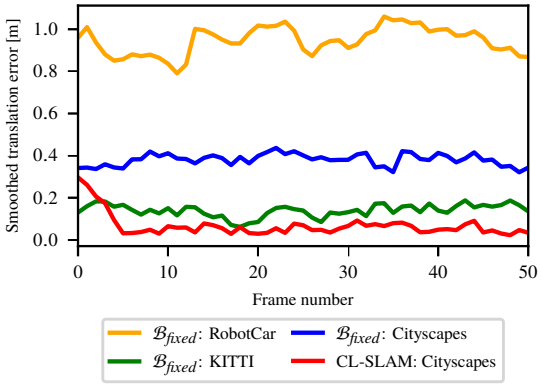



Fig. 4: The translation error on the initial frames of KITTI sequence 4. $\mathcal{B}_{fixed}$ is trained on the different environments indicating the domain gap between them. CL-SLAM overcomes this issue by performing online adaptation.

*Baselines:* We compare CL-SLAM to three baselines that are inspired by previous works towards online adaptation to a different environment compared to the environment used during training. As noted in Sec. 3, continual SLAM differentiates from such a setting in the sense that it considers a sequence of different environments. First, $\mathcal{B}_{expert}$ imitates the strategy employed by Li *et al.* [19], using a single set of network weights that is continuously updated based on the current data. This corresponds to only using the expert network in our architecture without resetting the weights. Second, $\mathcal{B}_{general}$ follows CoMoDA [16] leveraging a replay buffer built from previously seen environments. This method corresponds to only using the generalizer network. Finally, we compute the error without performing any adaptation, i.e., $\mathcal{B}_{fixed}$ utilizes network weights fixed after the pre-training stage. To further illustrate forward and backward transfer and to close the gap to classical CL, we provide results on an additional baseline $\mathcal{B}_{offline}$ in the supplementary material. This baseline is initialized with the same network weights as CL-SLAM but does not perform online adaptation to avoid masking backward transfer. In reality, it resembles data collection followed by offline training after every new environment.

**Adapting to New Environments:** In the initial part of the evaluation sequence (Eq. 9), the algorithm has to adapt to unseen environments. In accordance to the definition of the AQ in Sec. 3, we construct four sequences listed in the upper four rows of Table 2. Next, we deploy CL-SLAM and the baselines, initialized with the same set of model weights pre-trained on Cityscapes, on each of these sequences and compute the translation and rotation errors. Note that we do not apply median scaling since the PoseNet in our work produces metric estimates due to the velocity supervision term. Further note that for the first deployment after pre-training, $\mathcal{B}_{expert}$ corresponds to CL-SLAM. We observe that $\mathcal{B}_{expert}$ yields smaller errors than $\mathcal{B}_{general}$. This indicates the importance of online adaptation without diluting the updates with data from unrelated environments, if a high performance on the current deployment is the desideratum, and, thus, supports using the expert network in our approach. To compute the AQ score, after remapping using Eq. 3

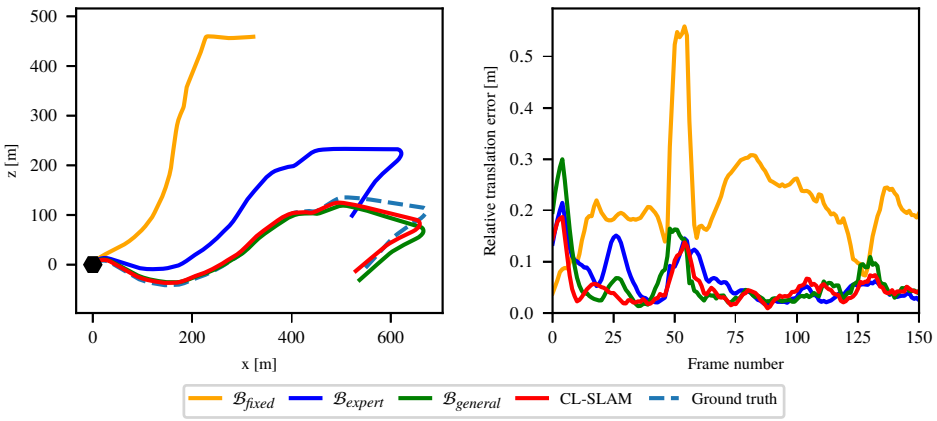



Fig. 5: Comparison of the trajectory on $k_2$ after previous deployment on $k_1$ and $r_1$ predicted by CL-SLAM and the baseline methods. The hexagon indicates the starting point.

Fig. 6: Relative translation error of the first 150 frames along $k_2$. Compared to $\mathcal{B}_{expert}$, CL-SLAM reduces the error more quickly due to initialization with the weights of its generalizer network.

we sum the errors and divide by the number of sequences:

$$\text{AQ} = \frac{1}{4}\left(M_{c_t \to k_1} + M_{c_t \to r_1} + M_{c_t \to r_1 \to k_1} + M_{c_t \to k_1 \to r_1}\right). \tag{10}$$

Comparing the AQ (see Table 3) for all experiments further endorses the previous findings in a single metric. Notably, continual adaptation is strictly necessary to obtain any meaningful trajectory.

Finally, we discuss the effect of consecutive deployments to different environments. In Fig. 4, we plot the translation error of the VO estimates on KITTI sequence 4 without online adaptation, separately trained on the considered datasets, and with adaptation, pre-trained on Cityscapes. As expected, without adaptation, the error is substantially higher if the system was trained on a different dataset showing the domain gap between the environments. By leveraging online adaptation, CL-SLAM reduces the initial error and yields even smaller errors than training on KITTI without further adaptation. Having established the existence of a domain gap, we analyze how the deployment to the current environment effects the future deployment to another environment, resembling the concept of forward transfer (FTW) in continual learning (CL). In detail, Table 2 reveals that the performances of all adaptation-based methods decrease when deploying them to an intermediate environment, e.g., $(c_t \to k_1)$ versus $(c_t \to r_1 \to k_1)$, where the effect is most pronounced for $\mathcal{B}_{general}$. In CL, such behavior is referred to as negative FWT.

**Remembering Previous Environments:** In the subsequent phase of the evaluation sequence (Eq. 9), the algorithm is redeployed in a new scene taken from a previously encountered environment. In accordance to the definition of the RQ in Sec. 3, we construct four sequences listed in the lower four rows of Table 2. Note that the first two sequences are part of the original evaluation sequence (Eq. 9) and the other two sequences are used as a reference to measure the effect of mixed environments.

Following the same procedure as in the previous section, we compute the translation and rotation errors. The resulting scores (see Table 2) demonstrate the benefit of employing a replay buffer to leverage previously learned knowledge, $\mathcal{B}_{general}$ yields smaller errors than $\mathcal{B}_{expert}$ on the majority of sequences. To compute the RQ, we follow Eq. 6:

$$\mathrm{RQ} = \frac{1}{2}\left[\left(M_{c_t \to k_1 \to r_1 \to k_2} - M_{c_t \to k_1 \to k_2}\right) + \left(M_{c_t \to k_1 \to r_1 \to k_2 \to r_2} - M_{c_t \to k_1 \to r_1 \to r_2}\right)\right]. \tag{11}$$

Comparing the RQ scores in Table 4 clearly shows that the drop in performance when mixing environments is less pronounced for $\mathcal{B}_{general}$. Our proposed CL-SLAM leverages this advantage due to its generalizer, while the expert still focuses on the current scene, achieving the highest RQ across the board.

To bridge the gap to classical CL, we also qualitatively compare the consecutive deployment to scenes from the same environment with introducing an intermediate scene from another environment, e.g., $(c_t \to k_1 \to k_2)$ versus $(c_t \to k_1 \to r_1 \to k_2)$. In CL, an increase/decrease in performance is known as positive/negative backward transfer (BWT). Whereas we observe positive BWT for $\mathcal{B}_{general}$ and CL-SLAM on the KITTI dataset, the sequence with final deployment on RobotCar suffers from negative BWT. A possible explanation for this inconsistent behavior is structural differences between the sequences of the same dataset inducing small domain gaps within a dataset that require a potentially more fine-grained scene classification. However, by always performing online adaptation independent of previous deployments, CL-SLAM circumvents such issues.

In Fig. 5, we visualize the generated trajectories in $k_2$ given previous deployment in $k_1$ and $r_1$ from our method and the evaluated baselines. Although $\mathcal{B}_{expert}$ can reproduce the general shape of the trajectory, it requires a warm-up time causing an initial drift, visible up to frame 40 in Fig. 6. On the other hand, $\mathcal{B}_{general}$ can leverage the experience from $k_1$ due to the rehearsal of the KITTI data from its replay buffer during the previous deployment in $r_1$. By following this idea, our proposed CL-SLAM combines the advantages of both baseline strategies.

Table 4: Comparison of the Retention Quality (RQ)

| | $\uparrow \mathrm{RQ}_{\mathrm{trans}} \times 10^{-3}$ | $\uparrow \mathrm{RQ}_{\mathrm{rot}} \times 10^{-3}$ |
|---|---|---|
| $\mathcal{B}_{fixed}$ | – | – |
| $\mathcal{B}_{expert}$ | -152.0 | -9.5 |
| $\mathcal{B}_{general}$ | <u>-14.4</u> | <u>-0.5</u> |
| CL-SLAM | **-7.3** | **-0.4** |

$\mathrm{RQ}_{\mathrm{trans}}$ refers to the retention quality with respect to the translation error, $\mathrm{RQ}_{\mathrm{rot}}$ is based on the rotation error. $\mathcal{B}_{fixed}$ does not perform adaptation, hence computing the RQ is meaningless. Bold and underlined values denote the best and second best scores.

Table 5: Ablation study on the number of adaptation cycles

| | | $c_t \to k_1$ | | $c_t \to k_2$ | |
|---|---|---|---|---|---|
| Updates | Relative FPS | $t_{err}$ | $r_{err}$ | $t_{err}$ | $r_{err}$ |
| 1 | 1.00 | 34.37 | 6.70 | 86.36 | 11.71 |
| 2 | 0.56 | 31.37 | 5.83 | 39.72 | 7.16 |
| 3 | 0.40 | 24.21 | 4.21 | **11.15** | 4.63 |
| 4 | 0.30 | 3.24 | 0.54 | 13.51 | 2.03 |
| 5 | 0.24 | **2.50** | **0.37** | <u>11.18</u> | <u>1.74</u> |
| 6 | 0.20 | <u>2.84</u> | <u>0.40</u> | 12.97 | **1.51** |

Translation error $t_{err}$ in [%] and rotation error $r_{err}$ in [°/100m] for varying number of weight updates $c$ performed during online adaptation. We use $c = 5$ in CL-SLAM. Bold and underlined values denote the best and second best scores.

**Number of Update Cycles:** We perform a brief ablation study on the number of update cycles performed during online adaptation, i.e., how often steps (2) to (4) are repeated for a given batch of data (see Sec. 4). For this, we deploy CL-SLAM to both KITTI sequences $k_1$ and $k_2$ and compute the translation and rotation error. As shown in Table 5, using five update cycles yields the most accurate trajectory while resulting in a 75% reduction in speed compared to a single cycle. However, please note that in this work, we do not focus on adaptation speed but on showing the efficacy of the proposed dual-network approach to balance the common continual learning trade-off between quick adaptation and memory retention.

## 6 Conclusion

In this paper, we introduced the task of continual SLAM, which requires the SLAM algorithm to continuously adapt to new environments while retaining the knowledge learned in previously visited environments. To evaluate the capability of a given model to meet these opposing objectives, we defined two new metrics based on the commonly used translation and rotation errors, namely the adaptation quality and the retention quality. As a potential solution, we propose CL-SLAM, a deep learning-based visual SLAM approach that predicts metric scale trajectories from monocular videos and detects global loop closures. To balance short-term adaptation and long-term memory retention, CL-SLAM is designed as a dual-network architecture comprising an expert and a generalizer, which leverages experience replay. Through extensive experimental evaluations, we demonstrated the efficacy of our method compared to baselines using previously proposed continual learning strategies for online adaptation of visual odometry. Future work will focus on transferring the proposed design scheme to more advanced visual odometry methods, e.g., using point matching via optical flow. We further plan to address the currently infinite replay buffer to mitigate the scaling problem, e.g., by storing more abstract representations or keeping only the most representative images.

## References

1. Bešić, B., Gosala, N., Cattaneo, D., Valada, A.: Unsupervised domain adaptation for lidar panoptic segmentation. IEEE Robotics and Automation Letters **7**(2), 3404–3411 (2022)
2. Bešić, B., Valada, A.: Dynamic object removal and spatio-temporal rgb-d inpainting via geometry-aware adversarial learning. IEEE Transactions on Intelligent Vehicles (2022)
3. Cattaneo, D., Vaghi, M., Valada, A.: Lcdnet: Deep loop closure detection and point cloud registration for lidar slam. IEEE Transactions on Robotics pp. 1–20 (2022)
4. Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., Schiele, B.: The Cityscapes dataset for semantic urban scene understanding. In: IEEE Conf. on Computer Vision and Pattern Recognition, pp. 3213–3223 (2016)
5. Geiger, A., Lenz, P., Urtasun, R.: Are we ready for autonomous driving? the KITTI vision benchmark suite. In: IEEE Conf. on Computer Vision and Pattern Recognition (2012)
6. Godard, C., Aodha, O.M., Firman, M., Brostow, G.: Digging into self-supervised monocular depth estimation. In: Int. Conf. on Computer Vision, pp. 3827–3837 (2019)
7. Gopalakrishnan, S., Singh, P.R., Fayek, H., Ambikapathi, A.: Knowledge capture and replay for continual learning. In: IEEE Winter Conf. on Applications of Computer Vision (2022)

8. Guizilini, V., Ambruș, R., Pillai, S., Raventos, A., Gaidon, A.: 3D packing for self-supervised monocular depth estimation. In: IEEE Conf. on Computer Vision and Pattern Recognition (2020)
9. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: IEEE Conf. on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
10. Howard, A., Sandler, M., Chen, B., Wang, W., Chen, L.C., Tan, M., Chu, G., Vasudevan, V., Zhu, Y., Pang, R., Adam, H., Le, Q.: Searching for mobilenetv3. In: Int. Conf. on Computer Vision, pp. 1314–1324 (2019)
11. Hurtado, J.V., Londoño, L., Valada, A.: From learning to relearning: A framework for diminishing bias in social robot navigation. Frontiers in Robotics and AI **8**, 69 (2021)
12. Jocher, G.: Yolov5 (2022). URL https://github.com/ultralytics/yolov5
13. Kemker, R., Kanan, C.: Fearnet: Brain-inspired model for incremental learning. In: Int. Conf. on Learning Representations (2018)
14. Kretzschmar, H., Grisetti, G., Stachniss, C.: Lifelong map learning for graph-based slam in static environments. KI - Künstliche Intelligenz **24**(3), 199–206 (2010)
15. Kurz, G., Holoch, M., Biber, P.: Geometry-based graph pruning for lifelong SLAM. In: Int. Conf. on Intelligent Robots and Systems, pp. 3313–3320 (2021)
16. Kuznietsov, Y., Proesmans, M., Van Gool, L.: CoMoDA: Continuous monocular depth adaptation using past experiences. In: IEEE Winter Conf. on Applications of Computer Vision (2021)
17. Kümmerle, R., Grisetti, G., Strasdat, H., Konolige, K., Burgard, W.: G2o: A general framework for graph optimization. In: Int. Conf. on Robotics and Automation, pp. 3607–3613 (2011)
18. Li, R., Wang, S., Gu, D.: DeepSLAM: A robust monocular SLAM system with unsupervised deep learning. IEEE Transactions on Industrial Electronics **68**(4), 3577–3587 (2021)
19. Li, S., Wang, X., Cao, Y., Xue, F., Yan, Z., Zha, H.: Self-supervised deep visual odometry with online adaptation. In: IEEE Conf. on Computer Vision and Pattern Recognition (2020)
20. Li, S., Wu, X., Cao, Y., Zha, H.: Generalizing to the open world: Deep visual odometry with online adaptation. In: IEEE Conf. on Computer Vision and Pattern Recognition, pp. 13179–13188 (2021)
21. Li, Z., Hoiem, D.: Learning without forgetting. IEEE Trans. on Pattern Analysis and Machine Intelligence **40**(12), 2935–2947 (2018)
22. Lopez-Paz, D., Ranzato, M.A.: Gradient episodic memory for continual learning. In: Adv. in neural information processing systems, vol. 30 (2017)
23. Luo, H., Gao, Y., Wu, Y., Liao, C., Yang, X., Cheng, K.T.: Real-time dense monocular SLAM with online adapted depth prediction network. IEEE Transactions on Multimedia **21**(2), 470–483 (2019)
24. Maddern, W., Pascoe, G., Linegar, C., Newman, P.: 1 year, 1000km: The Oxford RobotCar dataset. Int. Journal of Robotics Research **36**(1), 3–15 (2017)
25. McClelland, J.L., McNaughton, B.L., O'Reilly, R.C.: Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. Psychological review **102**(3) (1995)
26. McCraith, R., Neumann, L., Zisserman, A., Vedaldi, A.: Monocular depth estimation with self-supervised instance adaptation. arXiv preprint arXiv:2004.05821 (2020)
27. Mittal, M., Mohan, R., Burgard, W., Valada, A.: Vision-based autonomous UAV navigation and landing for urban search and rescue. In: Robotics Research, pp. 575–592. Springer (2022)
28. Mur-Artal, R., Montiel, J.M.M., Tardós, J.D.: ORB-SLAM: A versatile and accurate monocular SLAM system. IEEE Transactions on Robotics **31**(5), 1147–1163 (2015)
29. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., et al.: PyTorch: An imperative style, high-performance deep learning library. In: Adv. in neural information processing systems (2019)
30. Shin, H., Lee, J.K., Kim, J., Kim, J.: Continual learning with deep generative replay. In: Adv. in neural information processing systems, vol. 30 (2017)
31. Thrun, S.: Is learning the n-th thing any easier than learning the first? In: Adv. in neural information processing systems, vol. 8 (1995)
32. Zhan, H., Weerasekera, C.S., Bian, J.W., Reid, I.: Visual odometry revisited: What should be learnt? In: Int. Conf. on Robotics and Automation, pp. 4203–4210 (2020)
33. Zhang, Z., Lathuilière, S., Ricci, E., Sebe, N., Yang, J.: Online depth learning against forgetting in monocular videos. In: IEEE Conf. on Computer Vision and Pattern Recognition (2020)
34. Zhou, T., Brown, M., Snavely, N., Lowe, D.G.: Unsupervised learning of depth and ego-motion from video. In: IEEE Conf. on Computer Vision and Pattern Recognition, pp. 6612–6619 (2017)

# A Supplementary Material

## A.1 Technical Approach: Visual Odometry

As elaborated in the main paper, we generate VO estimates following the commonly used approach of using a trainable pose network [1, 3, 4, 5] for self-supervised depth estimation with a stream of monocular images. In this section, we first review the basic idea behind this approach and then describe the losses that we employ in more detail.

The core intuition is that given a source image $\mathbf{I_s}$ and the camera motion $\mathbf{O_{s\to t}}$, it is possible to generate a reconstructed view $\hat{\mathbf{I}}_{\mathbf{s\to t}}$ for a target image $\mathbf{I_t}$ using image warping. In detail, a 2D pixel $\mathbf{p_t}$ can be projected to the 3D point $\mathbf{P_t}$ using the camera matrix $\mathbf{K}$ and depth information $\mathbf{d_t}$ at this pixel. In monocular depth estimation, $\mathbf{d_t}$ is predicted by a neural network. Next, the camera motion $\mathbf{O_{s\to t}}$ is used to transform $\mathbf{P_t}$ to $\mathbf{P_s}$, which can be projected onto the plane of image $\mathbf{I_s}$ yielding the 2D pixel $\hat{\mathbf{p}}_\mathbf{s}$:

$$\hat{\mathbf{p}}_\mathbf{s} \sim \mathbf{K}\mathbf{O}_{\mathbf{t}\to\mathbf{s}} \underbrace{\mathbf{d_t}\mathbf{K}^{-1}\mathbf{p_t}}_{\mathbf{P_t}}. \tag{12}$$

Repeating this procedure for every pixel in $\mathbf{I_t}$, we obtain a mapping $\mathbf{p_t} \mapsto \hat{\mathbf{p}}_\mathbf{s}$ to reproject image coordinates:

$$\hat{\mathbf{I}}_{\mathbf{s}\to\mathbf{t}}(\mathbf{p_t}) = \mathbf{I_s}(\hat{\mathbf{p}}_\mathbf{s}). \tag{13}$$

Finally, using bilinear interpolation over these coordinates, the reconstructed view $\hat{\mathbf{I}}_{\mathbf{s}\to\mathbf{t}}$ can be generated and compared to the target image $\mathbf{I_t}$ to compute a loss value.

Our total loss is composed of the photometric reprojection loss $\mathcal{L}_{pr}$, the image smoothness loss $\mathcal{L}_{sm}$, and the velocity supervision loss $\mathcal{L}_{vel}$:

$$\mathcal{L} = \mathcal{L}_{pr} + \gamma\mathcal{L}_{sm} + \lambda\mathcal{L}_{vel}. \tag{14}$$

*Photometric Consistency:* To minimize the photometric error between the true target image and the reconstructed view, we compute the structural dissimilarity $\mathcal{L}_{sim}$ [2]:

$$\mathcal{L}_{sim}(\mathbf{I}, \hat{\mathbf{I}}) = \alpha\frac{1 - SSIM(\mathbf{I}, \hat{\mathbf{I}})}{2} + (1-\alpha)||\mathbf{I} - \hat{\mathbf{I}}||_1, \tag{15}$$

where $SSIM$ denotes the structure similarity image matching index [6]. To mitigate the effect of objects that are present in the target image $\mathbf{I_t}$ but not in the source images, Godard *et al.* [3] proposed to take the pixel-wise minimum over $\mathcal{L}_{sim}(\mathbf{I_t}, \hat{\mathbf{I}}_{\mathbf{s}\to\mathbf{t}})$ for all source images:

$$\mathcal{L}_p = \min_s \mathcal{L}_{sim}(\mathbf{I_t}, \hat{\mathbf{I}}_{\mathbf{s}\to\mathbf{t}}). \tag{16}$$

To further suppress the signal from static scenes or objects moving at a similar speed as the ego-robot, the same authors introduced the concept of auto-masking. The idea is to compute the loss only on those pixels, where the photometric error $\mathcal{L}_{sim}(\mathbf{I_t}, \hat{\mathbf{I}}_{\mathbf{s}\to\mathbf{t}})$ of the reconstructed image is smaller than the error $\mathcal{L}_{sim}(\mathbf{I_t}, \hat{\mathbf{I}}_\mathbf{s})$ of the original source image:

$$\mu_{mask} = \left[ \min_s \mathcal{L}_{sim}(\mathbf{I_t}, \hat{\mathbf{I}}_{\mathbf{s}\to\mathbf{t}}) < \min_s \mathcal{L}_{sim}(\mathbf{I_t}, \mathbf{I_s}) \right], \tag{17}$$

where $\mu_{mask}$ has the same width and height as $\mathbf{I_t}$.

The total photometric reprojection loss $\mathcal{L}_{pr}$ is defined as:

$$\mathcal{L}_{pr} = \mu_{mask} \cdot \mathcal{L}_p. \tag{18}$$

*Image Smoothness:* To regularize the depth prediction in image regions with less texture, we use an edge-aware smoothness term [2] computed for the predicted depth map $\mathbf{D_t}$. It encourages the DepthNet to generate continuous depth values in continuous image areas.

$$\mathcal{L}_{sm} = |\partial_x \mathbf{S}_\mathbf{t}^*| e^{-|\partial_x \mathbf{I_t}|} + |\partial_y \mathbf{S}_\mathbf{t}^*| e^{-|\partial_y \mathbf{I_t}|}, \tag{19}$$

where $\partial_i$ indicates the partial derivative with respect to axis $i$ and $\mathbf{S}_\mathbf{t}^* = \mathbf{S_t}/\bar{\mathbf{S}}_\mathbf{t}$ denotes the inverse depth (disparity) $\mathbf{S_t} = {\mathbf{D_t}}^{-1}$ normalized with its mean.

*Velocity Supervision:* To enforce metric scaling of the predicted odometry, we leverage the speed or velocity measurements from the robot. While such a rough measurement can be obtained inexpensively, e.g., by wheel odometry, it teaches the network to predict scale-aware depth and pose estimates. The velocity supervision term $\mathcal{L}_{vel}$ [4] imposes a loss between the magnitude of the predicted translation $T_{t\to s}$ and the distance traveled by the robot based on the velocity reading $v_{t\to s}$:

$$\mathcal{L}_{vel} = \sum_s \Big| \, ||T_{t\to s}||_2 - |v_{t\to s}| \Delta\tau_{t\to s} \, \Big|, \tag{20}$$

where $\Delta\tau_{t\to s}$ denotes the time between images $\mathbf{I_s}$ and $\mathbf{I_t}$.

## A.2 Additional Experimental Evaluation

In Table 6, we provide the results of an additional baseline that we call $\mathcal{B}_{offline}$. It does not leverage online adaptation but is initialized with the same network parameters as CL-SLAM. Note that in a practical setting, to some extent this corresponds to data collection followed by offline training with a replay buffer for every new environment. Although such a setup does not completely align with the core idea of continual SLAM, $\mathcal{B}_{offline}$ aims to close the gap to classical continual learning by not performing re-adaptation to previously seen environments, i.e., conducting a previously learned task, to avoid masking backward transfer.

Analogously to the adaptation-based methods, the performance of $\mathcal{B}_{offline}$ on $k_1$ degrades with an intermediate deployment to $r_1$, which is referred to as positive forward transfer (FWT) in classical continual learning (CL). Unlike the other methods, $\mathcal{B}_{offline}$ yields smaller errors on $r_1$ if it was previously deployed to $k_1$, known as positive FWT. A possible explanation for this inconsistent behavior is structural differences between the sequences of the same dataset inducing small domain gaps within a dataset that require a potentially more fine-grained scene classification.

Table 6: Translation and rotation error for computing the AQ and RQ metrics

| Previous scenes | Current scene | $\mathcal{B}_{offline}$ | | CL-SLAM | |
|---|---|---|---|---|---|
| | | $t_{err}$ | $r_{err}$ | $t_{err}$ | $r_{err}$ |
| $c_t$ | $k_1$ | 130.74 | 26.35 | 2.50 | 0.37 |
| $c_t$ | $r_1$ | 170.76 | 13.37 | 28.94 | 5.63 |
| $c_t \rightarrow r_1$ | $k_1$ | 182.62 | 38.38 | 3.24 | 0.54 |
| $c_t \rightarrow k_1$ | $r_1$ | 52.23 | 8.49 | 30.13 | 5.87 |
| $c_t \rightarrow k_1 \rightarrow r_1$ | $k_2$ | 23.77 | 4.76 | 4.85 | 1.59 |
| $c_t \rightarrow k_1 \rightarrow r_1 \rightarrow k_2$ | $r_2$ | 105.54 | 25.64 | 20.50 | 4.77 |
| $c_t \rightarrow k_1$ | $k_2$ | 69.48 | 7.45 | 7.48 | 1.63 |
| $c_t \rightarrow k_1 \rightarrow r_1$ | $r_2$ | 153.77 | 35.21 | 16.41 | 4.58 |

The *previous scenes* denote the scenes that have been used for previous training of the algorithm, the *current scene* denotes the evaluation scene to compute both errors $t_{err}$ in [%] and $r_{err}$ in [°/100m].

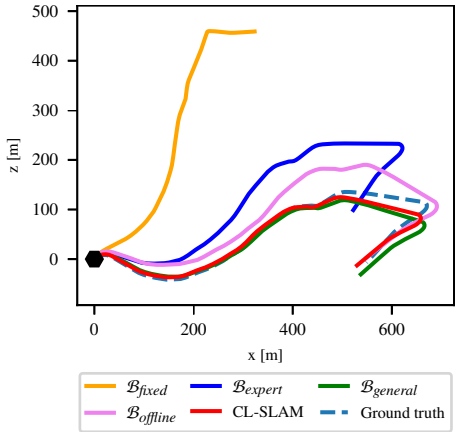



Fig. 7: A modified version of Fig. 5 including the additional baseline $\mathcal{B}_{offline}$. Comparison of the trajectory on $k_2$ after previous deployment on $k_1$ and $r_1$ predicted by CL-SLAM and the baseline methods.

## Supplementary Material References

1. Bešić, B., Valada, A.: Dynamic object removal and spatio-temporal rgb-d inpainting via geometry-aware adversarial learning. IEEE Transactions on Intelligent Vehicles (2022)
2. Godard, C., Aodha, O.M., Brostow, G.J.: Unsupervised monocular depth estimation with left-right consistency. In: IEEE Conf. on Computer Vision and Pattern Recognition (2017)
3. Godard, C., Aodha, O.M., Firman, M., Brostow, G.: Digging into self-supervised monocular depth estimation. In: Int. Conf. on Computer Vision, pp. 3827–3837 (2019)
4. Guizilini, V., Ambruș, R., Pillai, S., Raventos, A., Gaidon, A.: 3D packing for self-supervised monocular depth estimation. In: IEEE Conf. on Computer Vision and Pattern Recognition (2020)
5. Li, R., Wang, S., Gu, D.: DeepSLAM: A robust monocular SLAM system with unsupervised deep learning. IEEE Transactions on Industrial Electronics **68**(4), 3577–3587 (2021)
6. Wang, Z., Bovik, A., Sheikh, H., Simoncelli, E.: Image quality assessment: from error visibility to structural similarity. IEEE Transactions on Image Processing **13**(4), 600–612 (2004)