Road detection in adversarial weather conditions using RGB and LiDAR data

Miika Lehtimäki

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 23.07.2021

Supervisor

Prof. Ville Kyrki

Advisor

D.Sc. (Tech.) Antti Hietanen

Aalto University
School of Electrical
Engineering

Aalto University
School of Electrical
Engineering

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi
Abstract of the master's thesis

Author Miika Lehtimäki

Title Road detection in adversarial weather conditions using RGB and LiDAR data

Degree programme Automation and Electrical Engineering

Major Translational Engineering **Code of major** ELEC3023

Supervisor Prof. Ville Kyrki

Advisor D.Sc. (Tech.) Antti Hietanen

Date 23.07.2021 **Number of pages** 72 **Language** English

Abstract

Autonomous vehicles have previously used road markings as a reference for drivable area detection. For autonomous driving to be possible in weather conditions where these markings are not visible (for example snow cover, ice, heavy rain), the drivable area needs to be determined by other means.

In this work the use of machine learning models based on fully convolutional neural networks is evaluated for this task. The models use data projected into bird's eye view format, enabling detections agnostic to variations in sensor properties, such as resolution. In order to find an optimal model architecture, two hyperparameter searches are performed on different resolutions and parameter ranges.

A tool for partially automating the process of such dataset creation is described and implemented. The tool enables a human labeler to label a single global bird's eye view image for each driving sequence, improving the labeling efficiency by a factor of thousands compared to individually labeling the camera images. The dataset tool also demonstrates real-time production of LiDAR and RGB bird's eye view frames that are used as inputs for the neural network models.

The dataset tool was able to produce a dataset of sufficient quality for the machine learning model training and evaluation. Minor defects were observed that can most likely be rectified with relatively simple modifications. The models were able to successfully predict the general shape of surrounding roads, but exhibited noise on road edges and uncertainty in areas of little input data.

Keywords Autonomous Driving, Weather, Drivable Area Detection, Machine Learning, Bird's Eye View

Aalto University
School of Electrical
Engineering

Aalto-yliopisto, PL 11000, 00076 AALTO
www.aalto.fi
Diplomityön tiivistelmä

Tekijä Miika Lehtimäki

Työn nimi Tien havaitseminen vaikeissa sääolosuhteissa RGB- ja LiDAR-dataa hyödyntäen

Koulutusohjelma Automaatio ja sähkötekniikka

Pääaine Translationaalinen tekniikka **Pääaineen koodi** ELEC3023

Työn valvoja Prof. Ville Kyrki

Työn ohjaaja TkT Antti Hietanen

Päivämäärä 23.07.2021 **Sivumäärä** 72 **Kieli** Englanti

Tiivistelmä

Autonomiset autot ovat perinteisesti havainneet ajokelpoista aluetta tiemerkintöjen avulla. Jotta autonominen ajaminen olisi mahdollista sääolosuhteissa, joissa tiemerkinnät eivät ole näkyvissä (esimerkiksi lumipeitteen, jään tai kovan sateen takia), tulee ajokelpoinen alue havaita muilla tavoin.

Tässä työssä arvioidaan täyskonvoluutioneuroverkkoihin perustuvien koneoppimismallien soveltuvutta em. tarkoitukseen. Mallit hyödyntävät lintuperspektiivin projisoitua tietoformaattia, mahdollistaen sensoriominaisuuksista, kuten resoluutiosta riippumattoman havainnoinnin. Kaksi hyperparametrihakua eri resoluutioilla ja parametriavaruuksilla suoritetaan optimaalisen mallin arkkitehtuurin löytämiseksi.

Työkalu tällaisen tietojoukon luomisprosessin osittaiseksi automatisoinniksi kuvataan ja implementoidaan. Työkalun ansiosta ihmisen tarvitsee merkitä tie vain yhteen ajosekvenssiä vastaavaan, globaliin lintuperspektiivikuvaan, parantaen ihmistyön tehokkuutta monituhatkertaisesti yksittäisten kamerakuvien merkintään verrattuna. Työkalun implementaatiossa myös demonstroidaan neuroverkkomallien syötteenä käytettävien LiDAR ja RGB -lintuperspektiivikuvien reaaliajassa tuottaminen.

Työkalu kykeni tuottamaan riittävän laadukkaan tietojoukon koneoppimismallin kouluttamiseksi sekä arvioimiseksi. Pieniä vikoja oli havaittavissa, mutta ne ovat luultavasti korjattavissa verrattaen yksinkertaisilla muutoksilla. Mallit onnistuivat havaitsemaan ympäröivien teiden yleismuodon, mutta havainnoissa oli kohinaa erityisesti teiden reunoilla sekä epävarmuutta alueilla, joilla syötedata oli vähäistä.

Avainsanat Autonominen ajaminen, Sää, Ajokelpoinen alue, Tunnistaminen, Koneoppiminen, Lintuperspektiivi

Preface

I would like to thank Antti Hietanen and Antti Kangasrääsiö for their excellent thesis writing advice. Thanks for Viljami Lyytikäinen for understanding and support during the process.

I am most grateful for my friends, my mother Katri and my partner Karoliina for love, support and life worth living over the past (and hopefully future) years. This accomplishent would not have been possible or worth anything without you.

Espoo, 23.07.2021

Miika E. J. Lehtimäki

Contents

Symbols and abbreviations

Abbreviations

AABB	Axis-Aligned Bounding Box
AI	Artificial Intelligence
API	Application Programming Interface
BEV	Bird's Eye View
BVH	Bounding Volume Hierarchy
CNN	Convolutional Neural Network
CRF	Conditional Random Field
DORN	Deep Ordinal Regression Network
FCN	Fully Convolutional Network
GNN	Graph Neural Network
GNSS	Global Navigation Satellite System
GPU	Graphics Processing Unit
ICP	Iterative Closest Point
IMU	Inertial Measurement Unit
INS	Inertial Navigation System
IoU	Intersection over Union
KDE	Kernel Density Estimation
LiDAR	Light Detection and Ranging
ML	Machine Learning
MLP	Multi-Layer Perceptron
MSE	Mean Square Error
NDT	Normal Distributions Transform
ReLU	Rectified Linear Unit
ROI	Region of Interest
RoI	Region of Interest
RPN	Region Proposal Network
SVD	Singular Value Decomposition

1 Introduction

Over the last decade, autonomous driving has become one of the most dominant and researched topics in the field of transportation, particularly in automotive industry. Transportation companies and consumers are recognizing the value proposition of vehicular locomotion free of human labour; driver salaries account for a significant segment of cost structures of cargo, taxi and bus companies, and numerous hours of personal life are wasted on daily commute instead of recreation and recovery. Another, even more important factor of consideration is road safety — in 2016, 18.2 deaths per 100 000 population were caused by road accidents and road traffic injuries are the leading cause of death for people aged 5–29 years globally [1]. Such numbers can be partially explained by constraints of a human driver, especially limited sensory input, focus and reaction time.

The concept of replacing the human driver with technology was first conceived in the 1920's. After the 1980's, with the advent of microprocessors, progress on the subject started and then accelerated in the 2000's by the USA Defense Advanced Research Projects Administration's (DARPA) Grand Challenge on autonomous vehicles [2]. Contemporary acceleration of progress can be attributed to *machine learning* (ML) — a field of computer science driving the current *artificial intelligence* (AI) boom. Machine learning methods rely upon vast amounts of high quality data and computing power, both of which are available in quantities vastly exceeding those of past decades. Whereas AI methods developed in the past were built by encapsulating expertise of leading engineers and researchers, machine learning methods can find useful abstractions and structures directly from the data presented to them, without any human input. Such approach leaves the heavy lifting of algorithm development to computers, enabling human experts to focus on model architectures and data representations.

1.1 Sensing and Modeling the Environment

The basis for operating any kind of vehicle is defined by situational awareness — the ability of modeling the environment and the relationship of the vehicle with respect to it, at present and in immediate future. A human driver has developed this ability due to years of interaction with the world and concepts within it. Recreating such semantical understanding with means of technology remains one of the core challenges facing developers of autonomous driving systems.

The formation of a situational representation begins at sensing the environment. This is the natural strong point for robotics, as human drivers cannot be augmented with additional sensors. Sensors for autonomous vehicles are numerous and developing at fast pace. For example, cameras, *light detection and ranging* (LiDAR) scanners and radars are widely used sensors for sensing the environment, whereas *global navigation satellite systems* (GNSS), *inertial measurement units* (IMU) and wheel odometry are sensors and systems for gathering information about the state of the vehicle. Such wide spectrum of data modalities enables autonomous vehicles to gather information not available to humans, who are bounded by frontal stereo RGB vision and limited

auditory perception.

In contrast, processing the sensor data into a situational model for making vehicle control decisions is a significantly more difficult than gathering data. As mentioned, people have the advantage of immensely capable prosessing "hardware", optimized by evolution over billions of years. Replicating the human semantic understanding requires splitting the problem into subproblems, including drivable space detection, traffic signal detection and interpretation, object and obstacle detection and environment dynamics prediction.

Traditionally (before 2010's) these tasks have been approached by hand-crafting algorithms for feature detection and extraction [3]. Such algorithms inherently exhibit multiple layers of human bias: they are result of humans mimicking human perception using mathematics — a set of tools invented by humans. Therefore, these methods are prone to fragility — erroneus or unstable behaviour in presence of extreme mapping nonlinearities or sensory noise, both of which are common in real-world environments. Furthermore handcrafted algorithms might not use the computing resources optimally, resulting in weak performance in terms of latency and/or quality for given amount of computing power. For these reasons, contemporary techniques for situational modeling have almost entirely shifted to utilize machine learning methods.

1.2 Machine Learning and Data

The strength of machine learning is also its greatest weakness: while ML methods can create models capable of mapping extremely nonlinear relationships in the data, they impose significant requirements for the dataset used in the training process. In the beginning of the training process, the model is initialized randomly and thus does not have any understanding of the structures of the data. As such, the upper bound for the performance of the model is defined by the training dataset. In practice this implies that the dataset has to contain representative examples over the entire domain of the desired functionality of the model.

The vast majority of current ML models are trained in a configuration called *supervised learning*, in which parameters of the model are changed so that it maps input-output data pairs by minimizing the difference of model outputs to desired outputs defined by the dataset. Therefore, the method relies on existence of these output examples, which are often referred to as *labels*. These labels usually are produced by a human labeler, imposing a significant labour demand.

Supervised learning is the most readily applicable and the most researched method for the perception (sub)problems presented in Section 1.1, making it the most promising currently available approach for the problem of road detection. However, the problem imposes very particular requirements for the training data.

1.3 Thesis Contribution

The goal of this thesis is to find a solution for the subproblem of drivable space detection, particularly in weather conditions where strong features and traffic indicators

(signs, street markings) are not clearly visible due to ice, snow and/or precipitation. Drivable space refers to any surfaces allowed for vehiclular traffic: for example, a sidewalk is not considered a drivable area but the opposite lane is. To train machine learning models for the task, a sufficiently large dataset with representative scenarios and labels is required.

Numerous autonomous driving datasets featuring dense road labels (in this context, dense labels refer to per-pixel semantic labels for RGB images) are available, such as the popular KITTI dataset [4]. However, currently no publicly available dataset does feature both dense road labeling and adverse weather conditions like described above. Therefore a significant portion of the thesis is dedicated to development and implementation of a tool for creating such dataset.

A naive approach would be to manually label drivable areas in a dataset offering the desired weather conditions. However, a minute worth of footage on 8-camera setup running at 10Hz would consist of 4800 images. Given that numerous such sequences would be required for training a reliable model, manual labeling would clearly be very labour-intensive. Instead, in this work a semiautomatic tool for dataset creation is developed. The tool requires human labeler to label a single bird's eye view image for each driving sequence, reducing the labeling work required for at least a factor of several thousands.

In current state-of-the-art *convolutional neural networks* (CNNs) are widely used for various image processing tasks, such as image classification, feature detection and segmentation. As the thesis is focused on the dataset tool, for machine learning models, adaptations of two well-established CNN-based meta-architectures are chosen for evaluation. A hyperparameter search is performed on the architectures to find the optimal model configuration for this specific problem.

No comparable tool to the dataset tool presented in this paper is publicly available, so the dataset tool output is constrained to be evaluated only qualitatively. Reliability is assessed by looking for any kind of distortions and discontinuities in the output data. Road detection models are evaluated qualitatively and quantitatively. Quantitative analysis is performed by splitting a portion of the produced dataset into an evaluation set which is not seen by the model during training. The evaluation set is used as a ground truth reference, against which the model performance can be evaluated. Hyperparameter searches are done according to this quantitative metric.

1.4 Thesis Structure

The thesis is structured as follows: in Section 2 current state-of-the-art methods are presented. The importance of the bird's eye view representation, traditional sensor fusion, sensor fusion with neural networks and image segmentation with neural networks and related methods are discussed.

Section 3 is split into two parts: Section 3.1 presents the dataset tool, its phases and novel algorithms developed for it. General algorithms for point cloud processing are presented first and the following sections describe the dataset tool pipeline in detail. Section 3.2 discusses the machine learning models, their architectures and hyperparameters.

Experiments (including the hyperparameter search) are defined in Section 4. Results for both the dataset tool and the road detection model are presented in Section 5. Conclusions are discussed in the section 6.

2 Background

2.1 Bird's Eye View

Convolutional neural networks are commonly used for feature and object detection in visual data. One crucial limitation of CNNs is that despite the transformation invariance they are still variant to scale, meaning that the network has to learn a distinct detector for all scales for each object. Images captured by perspective cameras (i.e. regular cameras) naturally exhibit objects of varying scale due to perspective projection: objects closer to the camera appear larger. Therefore it is beneficial to present the CNN input data in a format that reduces the scale variability [5, 6].

In the context of autonomous driving, vast majority of the objects of interest lie on an approximately horizontal plane around the vehicle. Thus, a top-down orthographic projection (commonly in the literature known as the *Bird's Eye View* or BEV) provides a presentation format with the desired property of constant-scale objects without losing significant amount of useful information due to projective occlusion. For these properties, methods utilizing BEV projection as a preprosessing step have attracted widespread research interest in recent years [7].

LiDAR scanners provide direction and distance measures for series of light rays, enabling their output to be processed into a 3D point cloud in straightforward manner. In case two of the three axes lie parallel to the aforementioned horizontal plane, such point clouds can trivially be projected into BEV format by collapsing the vertical dimension.

In recent years, numerous methods based on BEV-projected LiDAR data have emerged. These methods utilize wide variety of techniques developed in past decade to increase their accuracy and computational efficiency. Yang et al. utilize CNNs constructed from *residual blocks* [8] alongside skip connections similar to *U-Net* [9] for object detection in *PIXOR* [10]. In *HDNET* [11], they extend the architecture to perform road labeling. In *BirdNet* [12] Beltrán et al. use 2D BEV projection and incorporate Faster R-CNN *Region Proposal Network* (RPN)[13] meta-architecture for object detection. Furthermore, they propose a post-processing step for aligning object bounding boxes to a ground plane estimation to achieve object bounding box detection in 3 dimensions. In *FAF* [14] Luo et al. also use CNNs but instead of flattening the vertical dimension of the input data they discretize the point cloud into a 3D voxel grid. As convolutional operations on high resolution voxel grids are computationally expensive, Yan et al. propose *SECOND* [15], an optimization on convolutional operations exploiting the sparse nature of such grids. In *PointPillars* [16], Lang et al. propose an different sparsity optimization for 2D BEV projections: as a preprosessing step, a linear feature network is run on a dense, indexed tensor of augmented point projection bins (called pillars in the paper). PointPillars performs in KITTI test BEV and 3D detection benchmarks similarly or better than methods above, while providing 2-6 times higher speed, making it the most powerful method for single-LiDAR object detection. To completely eliminate the requirement for sparsity optimization, Shi et al. propose a *Point-GNN* [17]: a network architecture

utilizing *Graph Neural Networks* (GNNs) as their backbone.

2.2 Multimodal Data and Sensor Fusion

All the methods discussed in the previous section depend solely on the LiDAR data. Despite the impressive results, it is clear that point clouds do not always contain enough information to reliably identify objects due to their relative spatial sparsity [18]. Compared to models fusing multi-modal sensory input, these models have reduced available information and therefore they lack potential to reduce state uncertainty beyond what can be deduced from the LiDAR data. In addition, single-modal approaches suffer from loss in redundancy — in the presence of an enviromental disturbance (e.g. sensor getting blocked by snow or dust) or a sensor failure the system is rendered unable to perform reliable detections [19]. Thus, autonomous systems often perform *sensor fusion* to address the aforementioned issues.

2.2.1 Classical Sensor Fusion

In the context of robotics and control systems, sensor fusion has traditionally been achieved by reconstructing unknown variables in a hand-crafted model from available sensor data [20]. Other widely used techniques derive from Bayesian inference or Monte Carlo methods.

Developed in 1960 by R. Kalman, the *Kalman Filter* [21] has became the de facto method for linear state estimation in the presence of statistical and systematical errors, which are inherent to practically every real-world sensing task. Such popularity has been achieved due to the fact that Kalman Filter provides an optimal unbiased mininum mean square error (MSE) linear state estimate, given that process and measurement covariances are known [22]. As standard Kalman Filter applies only for linear systems, many extensions and generalizations for nonlinear systems have been developed, such as *Extended Kalman Filter* and *Unscented Kalman Filter* [23].

Methods based on *Bayesian inference* exploit the rule known as Bayes' theorem:

$$\mathbb{P}(H|E) = \frac{\mathbb{P}(E|H)\mathbb{P}(H)}{\mathbb{P}(E)} \tag{1}$$

where $\mathbb{P}(H|E)$ denotes probability for hypothesis H given that event E has occurred and $\mathbb{P}(E|H)$ the probability that E has happened if H is true. Thus, $\mathbb{P}(H)$ and $\mathbb{P}(H|E)$ denote the *a priori* and *a posteriori* probabilities for the hypothesis H. [20] The Kalman Filter is in fact a closed form for an optimal linear Gaussian distribution Bayesian filter. Other Bayesian inference based algorithms include Rauch-Tung-Striebel smoothers (RTSS) and grid filters and smoothers, which are solutions to probabilistic Markov sequence models [24, pp. 12-14].

Monte Carlo methods have been adapted into sensor fusion problems in a form of *Particle Filters*. More specifically, Particle filters are *Sequential Monte Carlo* (SMC) methods for solving the optimal filtering problem in a recursive manner. This is achieved by tracking a large number of weighted state hypotheses called particles. The algorithm consists of three steps, executed sequentially [25, 26]:

1. Update: propagate particles according to the model dynamics
2. Importance sampling: weigh particles based on agreement with sensor measurements
3. Resampling: Discard particles with smallest weights and clone particles with largest weights

Particle filters enable greater flexibility over Kalman Filters as they can produce accurate estimations in presence of nonlinear model dynamics and error distributions. However, to achieve sufficiently small estimation variance, a potentially large number of particles is required, posing a significant computational load. Furthermore, the number of required particles can be difficult to predict, presenting an application-specific hyperparameter [27].

2.2.2 Sensor Fusion with Neural Networks

In contrast to traditional sensor fusion methods, machine learning based sensor fusion does not rely on hand-crafted dynamics models or assumptions about process probabilities; instead, feature distributions and model parameters are learnt directly from a dataset. To accomplish this, the dataset and learning model are required to be prepared and configured accordingly. In general, architectures for neural networks performing sensor fusion fall into one of three categories: early fusion, late fusion and deep/cross fusion — each with their respective benefits and drawbacks [19].

In *early fusion* inputs are first merged and then passed through all layers of the network. Usual merging strategies include tensor concatenation, addition and gating. Early fusion approach is well suited for data allowing for low abstraction level feature-wise alignment; this is typically achieved by projecting the data into an intermediate format (such as the BEV). If such alignment is unattainable, the network might not be able to find meaningful structure from the data and thus suffer from underfitting [28]. Early fusion network architecture is illustrated in Fig. 1.

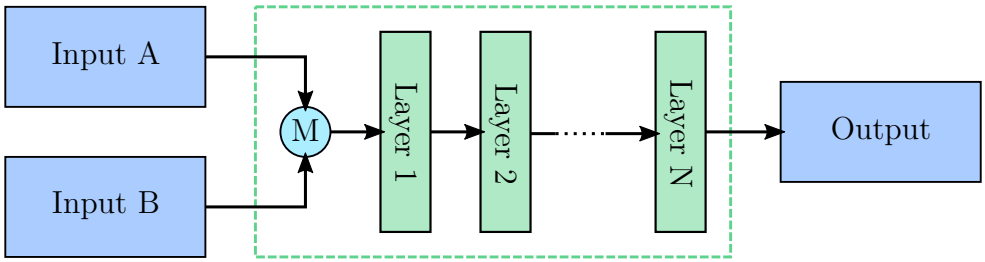


Figure 1: Early fusion neural network architecture, M denotes the merging operation

In contrast to the early fusion scheme, in *late fusion* individual, mutually detached networks for each input are utilized and their outputs are merged afterwards. A strong point of the late fusion is the individuality of the networks; it allows each detector branch to be designed and tested separately. Furthermore, it enables practitioners to use existing and pretrained networks, reducing computational resource requirements

for network training. However, each network only has access to a single modality data and thus, is subject to the weaknesses discussed in Section 2.2 — namely increased uncertainty and lack of redundancy. In consequence, benefits of this family of methods rely solely on implementation of the data merging operation [5] [28]. Late fusion network architecture is illustrated in Fig. 2.

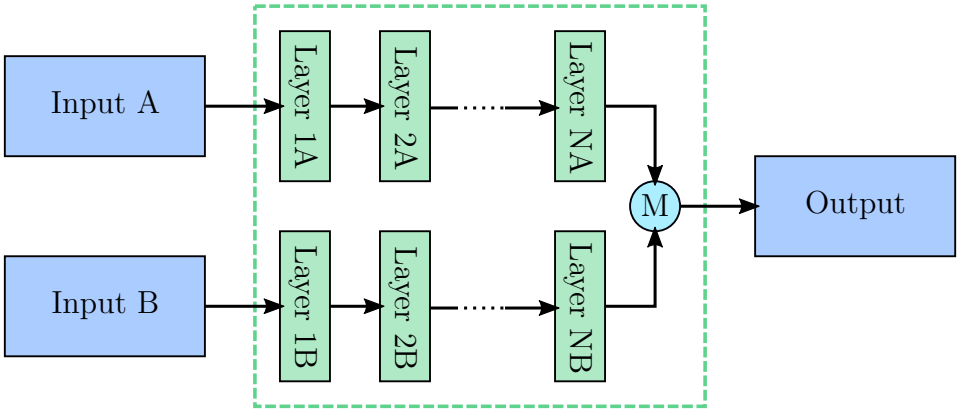


Figure 2: Late fusion neural network architecture

Neither of the early and late fusion approaches take full advantage of flexibility of neural networks due to the fusion being done on a very low or high abstraction level; it is often beneficial to allow the network to learn suitable abstractions for fusion by itself in a *deep fusion* scheme. This can be achieved by performing the merge operations in middle of the network architecture, as illustrated in the Fig. 3. In practice, the network "layers" can be individual subnetworks with heterogeneous inputs and outputs, providing practically infinite amount of architectural possibilities. Deep fusion methods are demanding in terms of computational resources, since no (sub)network can be trained separately due to the inherently intertwined nature of such schemes. Despite this, deep fusion methods have proved to be powerful and flexible tools for multimodal sensor fusion [19]. Different deep fusion approaches include Deeply-Fused Nets [29], cross fusion [28] and FractalNet [30].

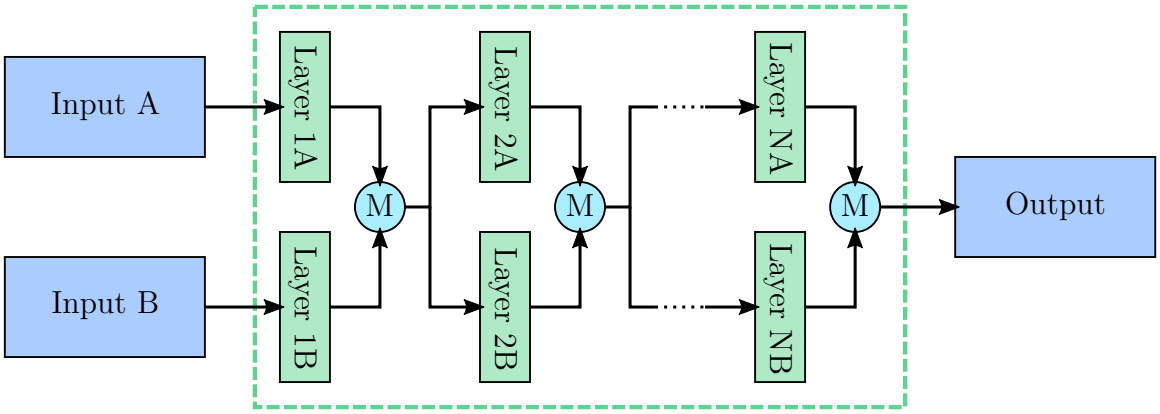


Figure 3: Deep fusion neural network architecture

2.2.3 Neural Network Sensor Fusion for Autonomous Driving

MV3D [5] by Chen et al. is one of the first successful methods utilizing neural network with multimodal inputs for autonomous driving. In similar fashion to the BirdNet [12], the architecture uses the bird's eye view as an input and RPN as an intermediate processing phase. The BEV projection of the point cloud is represented as an image with three channels encoding height, density and intensity. In addition to the BEV, the network also has two front view inputs: projected point cloud and an RGB image. The RPN forms proposals for object locations, which are then pooled with each view to produce vectors of identical size, enabling merge operation and subsequent passing to a deep fusion network. The deep fusion network detects 3D object bounding boxes, from which orientation of the object is deduced.

Ku et al. improve upon MV3D by introducing *AVOD* [31], a similar method relying on RPNs to perform *Region of Interest* (RoI) proposals. Full-resolution feature maps are used to enable more reliable region proposal recall for small object classes. To generate the feature maps, an encoder-decoder architecture with skip connections similar to U-Net [9] is used. To allow for efficient processing of full-resolution feature maps, convolutions with 1×1 kernels are utilized to effectively select important features for given class. Furthermore, AVOD proposes improved bounding box and object orientation encoding, reducing ambiguity and increasing efficiency.

MV3D and AVOD are object-centric approaches with separate backbones for each modality, performing fusion on high level after RoI-cropping. This forces the network to learn elaborate abstractions, requiring extensive computation. As discussed in Section 2.1, performing detections in the BEV representation format helps to alleviate the computational load. With this motivation, a family of methods transforming camera image features to BEV space has been developed.

Liang et al. [32] propose a fusion method for projecting RGB image features into BEV format using continuous(standard) convolution with learned *Multi-Layer Perceptron* (MLP) kernel. This approach enables use of deep fusion architecture over feature pyramid resolutions levels and object detection directly from the BEV representation. For mathematically more direct approach, Wang et al. [33] propose a pooling layer with non-homogeneous sparse mapping (i.e. sparse matrix multiplication) between the BEV and camera processing pipelines, enabling seamless communication between the two modalities. Roddick et al. [6] propose a similar method, projecting image-space features directly to a 3D voxel lattice.

Instead of projecting camera image features, series of works have explored the potential of projecting the original RGB images into the BEV domain. By using corresponding predicted depth image and inverse camera matrix, a 3D point cloud referred to as *pseudo-LiDAR* can be produced. Both Wang et al. [34] and Weng et al. [35] utilize *Deep Ordinal Regression Network* (DORN) [36] for predicting depth image from a monocular RGB image. In addition to the expected object bounding box and orientation prediction network Weng et al. propose a self-supervision mechanism by introducing a *Bounding Box Consistency Loss* term: the *Intersection over Union* (IoU) between the 3D bounding box and corresponding 2D segmentation bounding

box. Wang et al. [37] propose improvements to the naive pseudo-LiDAR by employing *Pyramid Stereo Matching Network* [38] — a network based on a stereo camera input - for depth estimation. For further estimation precision, they propose usage of depth cost volume for depth map prediction and pseudo-LiDAR point cloud correction with a real LiDAR point cloud. This sensor fusion approach is achieved with a graph-based two-phase quadratic optimization algorithm, achieving alignment of the dense, predicted pseudo-LiDAR with the sparse, real LiDAR point cloud. Vora et al. propose a method called *PointPainting* [18]: labels from an image segmentation network are projected into the LiDAR points, providing the point cloud network with additional data related to the points. The technique can be incorporated to any method with raw LiDAR point cloud as an input (such as the ones presented in Section 2.1); only modification required being the introduction of additional channels for segmentation classes.

The fusion methods presented above seek to solve the problem of object detection — even if some of the techniques are partially applicaple in the road detection domain, it is clear that methods involving strong assumptions about the spatial structure of detections (such as the RPN-based approaches) might not provide advantage to the road detection problem. Cheng et al. [39] propose a solution exactly for road detection; similar fashion to PIXOR [10] and AVOD [31], an approach based on U-Net [9] architecture is used. The method falls under the category of early fusion: input consisting of a precomputed occupancy grid and projected camera images are fed to the network, output being a binary road classification map. The neural network architecture presented in this thesis is heavily inspired by the method proposed by Cheng et al.

2.3 Image Segmentation with Neural Networks

Most of the methods presented previously are designed to detect parametric representations of objects (bounding boxes or position/orientation). However, such representations are not useful when detecting objects or features without well defined shape and/or if the object shape is to be determined. Both of these conditions are true for the road detection task, making such models unsuitable for the task specifically as they are proposed.

Fortunately, when using the BEV representation format for the model input data, the problem of road detection and heightmap production can be reframed as *image segmentation* — a problem that has been extensively studied and for which numerous deep learning based methods have been developed over the recent years [40]. Image segmentation considers the problem of assigning a label (binary or one of multiple classes) for each pixel in an image. Strictly speaking the heightmap production does not fall in this category as instead of a class label, a continuous value for each pixel is assigned. However, in this particular case only minimal modifications to segmentation models are required to make them suitable for the problem.

Convolutional Neural Networks

In 1998 LeCun et al. [41] demonstrated the superiority of CNNs over all other ML methods for image recognition. In 2010's, with introduction of massively parallellized computing hardware (such as GPUs) CNNs have become regarded as the de facto basis for machine vision tasks. *AlexNet* by Krizhevsky et al. [42] is one of the most recognized evolutions of LeCun's work, achieving state of the art performance by wide margin on ImageNet Large Scale Visual Recognition Challenge 2012. The essential contributions include usage of *Rectified Linear Unit* (ReLU) activation functions, dropout regularization and GPU acceleration for training. Comparison of LeNet and AlexNet is presented in Fig. 4

In VGGNet [43], Simoyan et al. replace the large convolution kernels with small ones of sizes 3×3 and 1×1 in favour of deeper network architecture with more layers. This increases the nonlinear inference capabilities of the network without affecting the receptive field of the kernels.

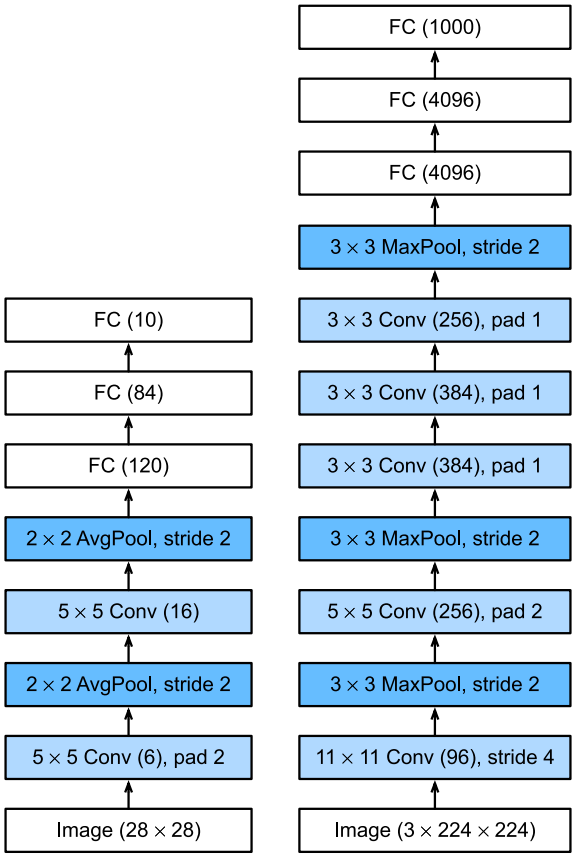


Figure 4: Comparison of LeNet [41](left) and AlexNet [42] (right), as presented in [44, Chapter 7].

Fully Convolutional Networks

Despite the fact that the works above address the problem of image classification, the convolutional backbone serves as a key building block for segmentation networks.

Long et al. [45] propose a *Fully Convolutional Network* (FCN) meta-architecture, a class of models capable of producing a segmentation for an input image of arbitrary size. They achieve this by transforming previous networks into convolutional ones. This is done by effectively running the network on translated and padded versions of the image and by replacing the classification head with yet another convolutional layer. FCN architecture is presented in Fig. 5.

While achieving good results, the FCN approach leads to the significant disadvantage of the method — such "naively" implemented FCN models do not allow for real-time inference [40]. Furthermore, partial observation of the image results into lack of global contextual information, leading to label inconsistency, localization errors and resolution imprecision. The latter issue was addressed in ParseNet [46] by Liu et al. by directly pooling global context features from the hierarchical feature maps. Chen et al. [47] address the label imprecision problem by introducing *Conditional Random Fields* (CRF) on top of the prediction model. However, since these works introduce additions to FCN models, they further reduce the inference time performance.

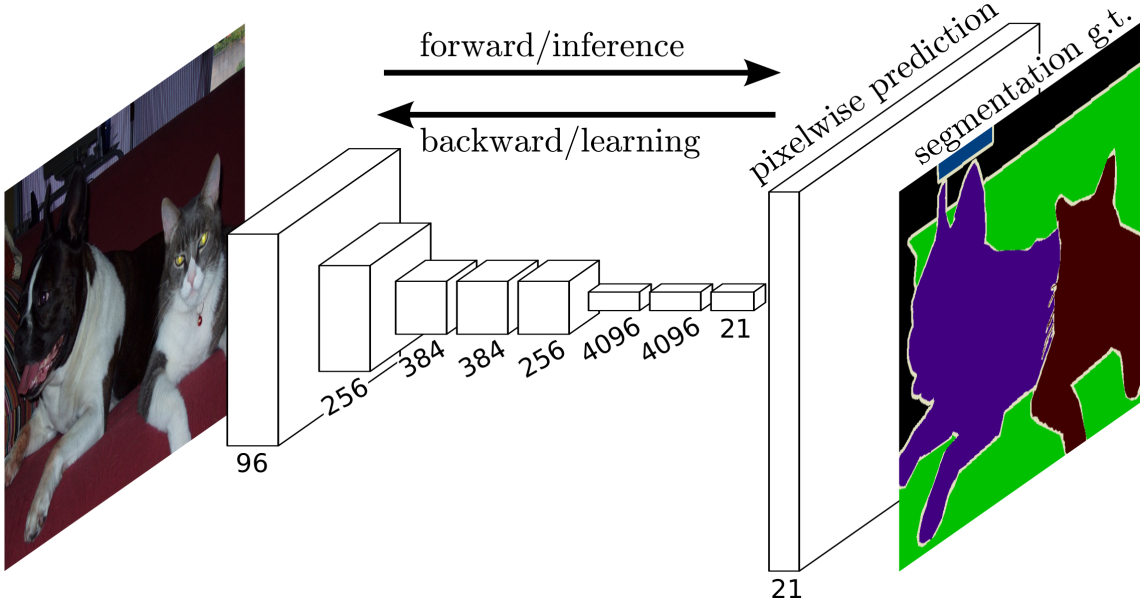


Figure 5: Fully convolutional network architecture [45].

Encoder-Decoder Networks

A significant oversight in naive FCN design is the fact that the forward inference is required to be performed for each pixel in the target image. This introduces large computational overhead, resulting to the performance issues discussed above. A widely utilized solution for the problem is to extend the network with deconvolutional (convolution transpose) layers to "decompress" a global representation of the image into a labeled image [48]. Such group of network architectures are referred to as *Encoder-Decoder Networks* and are used across wide range of data transformation,

fusion and compression problems. A typical encoder-decoder network is presented in Fig. 6

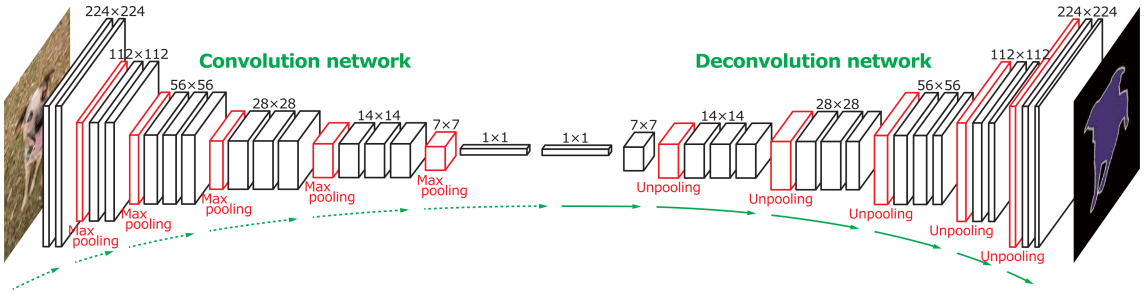


Figure 6: Encoder-Decoder network architecture [48].

Despite the increased computational efficiency, the encoder-decoder model suffers from similar (and sometimes even worse) localization errors and label "blurriness" than FCNs. This is due to the fact that the decoder is required to form a complete image only from the global, compressed information. In U-Net [9] Ronneberger et al. address this drawback by introducing skip connections between each resolution level. These connections provide high-resolution local information which assists production of high quality, spatially accurate label predictions. Zhang et al. [49] successfully predict roads from summertime satellite photos, demonstrating the feasibility of the architecture for such task. U-Net architecture has also inspired further research into deeper network architectures: Fu et al. [50] propose stacking arbitrary number of networks with skip connections and hierarchical supervision, forming very deep and powerful segmentation models.

Neural network based approaches for autonomous driving have been subject to extensive development for a past few years. Wide variety of approaches providing impressive results have been conceived. However, evaluating every method would be infeasible within the scope of this thesis — especially given the focus on the dataset tool. Therefore the relatively simple and extensively studied encoder-decoder and U-Net meta-architectures are chosen for evaluation.

3 Research Material and Methods

In Section 3.1 the methodology for creating dataset tool for semi-automating the process of BEV adverse driving conditions dataset is presented and discussed. Neural network models performing the road detection are presented in Section 3.2.

3.1 Dataset tool

At the time of the thesis writing, no publicly available dataset featuring winter conditions and road labels existed. Thus, a tool for producing such dataset was required to apply machine learning methods for road detection.

Largely for same reasons as discussed in Section 2.1, road labeling task greatly benefits from bird's eye view presentation. In addition, compared to labeled camera images, BEV format is more readily usable for subsequent tasks related to autonomous driving, such as path planning. However, translation from raw LiDAR and camera data directly to BEV maps with a neural network requires the model to learn unnecessarily high-level abstractions of the data. Such domain translation would most likely be computationally prohibitely expensive. Furthermore, this approach would make the model specific for the particular sensor configuration used for the dataset capture; changing sensors or their calibration will most likely result in unreliability or complete failures in model outputs.

To avoid aforementioned domain translation, inputs to the network should also be presented in BEV format. For this approach to be viable, input preprossing is required to be performed in real-time, preferably under 50 milliseconds to leave sufficient time budget for neural network model so that detection frequency of 10Hz can be achieved.

Traditionally segmentation labeling for road labels have been done for camera images. This requires considerable amount of human labour, most of which is redundant in theory: a particular batch of road is typically detected during multiple time points and potentially by multiple cameras(especially with multi-camera setups sharing a imaging direction, such as stereo cameras). In this work a solution is proposed to significantly reduce the human workload for labeling: by projecting globally aligned camera images on an ground level estimate a global BEV image resembling a satellite image can be produced. By labeling this single global image, a road label for any viewpoint (including the real cameras) can be virtually recreated, given sufficiently reliable ground level estimate (heightmap). This reduces the total labeling workload from potentially thousands of images to one.

In summary, the dataset tool should be able to produce 1) global BEV heightmap and corresponding BEV RGB image and 2) real-time BEV representations of a LiDAR scan and RGB camera images.

The dataset tool is divided into two phases: the first phase merges data over a driving sequence to form the global BEV images (heightmap and RGB). The second phase 1) crops and orients target (output) frames from the global images for each time point in sequence and 2) forms LiDAR and RGB source (input) frames while demonstrating the real-time feasibility described above.

Block diagram for the first phase is presented in Fig. 7. The process is structured into transformation registration, heightmap formation and RGB image projection. In transformation registration, relative transformations for each frame are computed by refining the provided a priori GNSS+INS localization data with point cloud alignment. This is achieved by forming a global reference point cloud from GNSS+INS data and then using extended ICP algorithm to align individual LiDAR point clouds against it. Heightmap is created by accumulating aligned point clouds into a data structure based on discrete grid of vertical point columns. For such structure, a multitude of useful algorithms related to heightmap estimation and its effective usage can be defined. Finally, with the refined vehicle transformations and camera calibration parameters the RGB camera images can be projected against the heightmap estimate, producing the global BEV RGB image. Details of the first phase implementation are presented in Section 3.1.1.

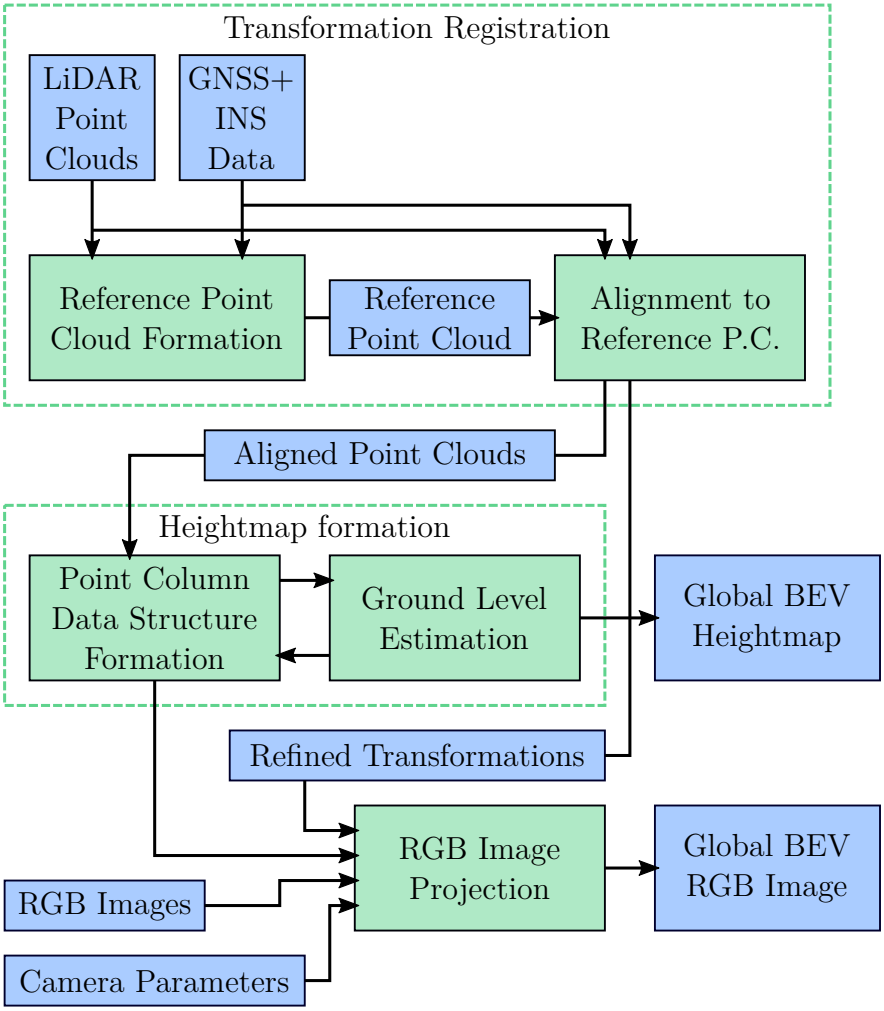


Figure 7: Dataset tool first phase block diagram

In the dataset tool second phase, the final BEV frames are produced. Source frames are inputs to the neural network model and target frames represent the desired

outputs. Source frames are to be produced in real-time to provide sufficiently low latency for the autonomous driving application.

The two source frames represent LiDAR and RGB data projected into the bird's eye view. As the LiDAR point cloud readily provides 3D structure, formation of LiDAR BEV frame is relatively trivial. On the contrary, RGB camera image projection requires a dense 3D structure, which can be generated from the lidar data by triangulation. The algorithm is discussed in detail in Section 3.1.2.2. Second phase block diagram for source frames is presented in the Fig. 8

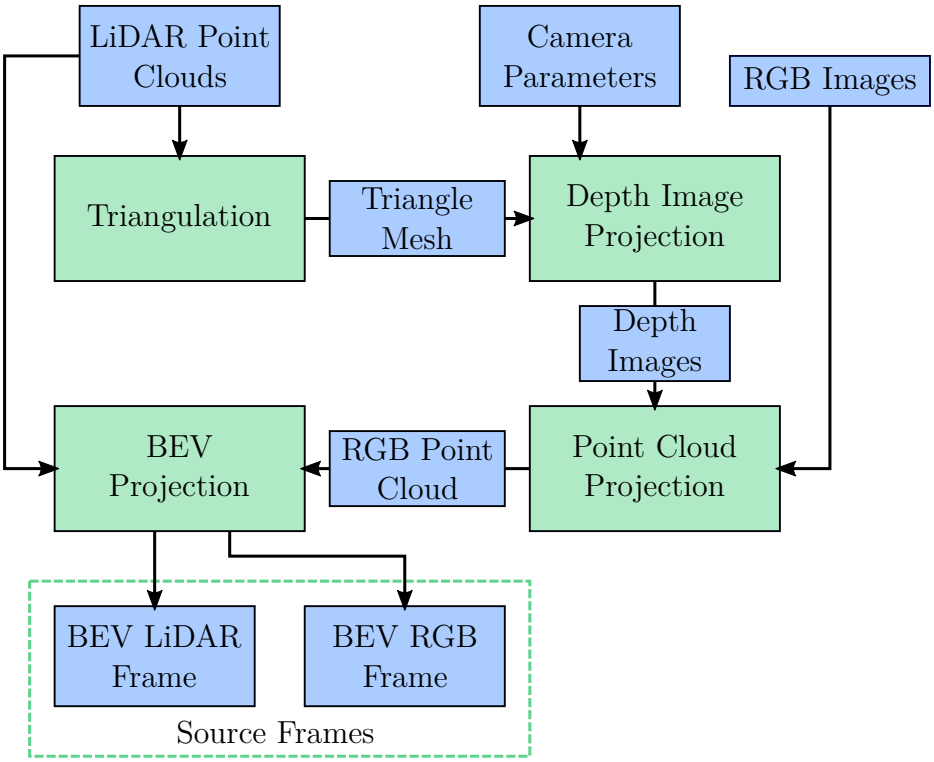


Figure 8: Dataset tool second phase block diagram for source frames

To produce the target frames, a human labelel needs to create a road label corresponding the global BEV RGB image. Using the refined transformations produced in the first phase, the global BEV heightmap and the road label can be cropped and oriented to produce target frames corresponding to the source frames. Second phase block diagram for target frames is presented in Fig. 9 An example of dataset tool output is presented in Fig. 10.

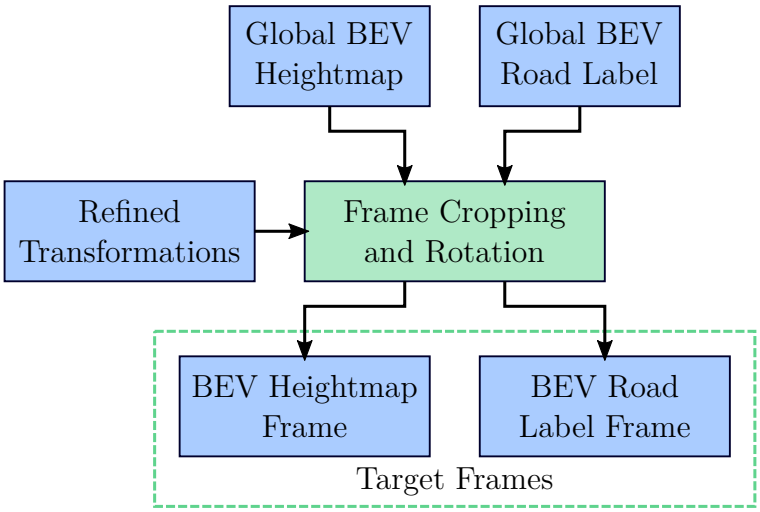
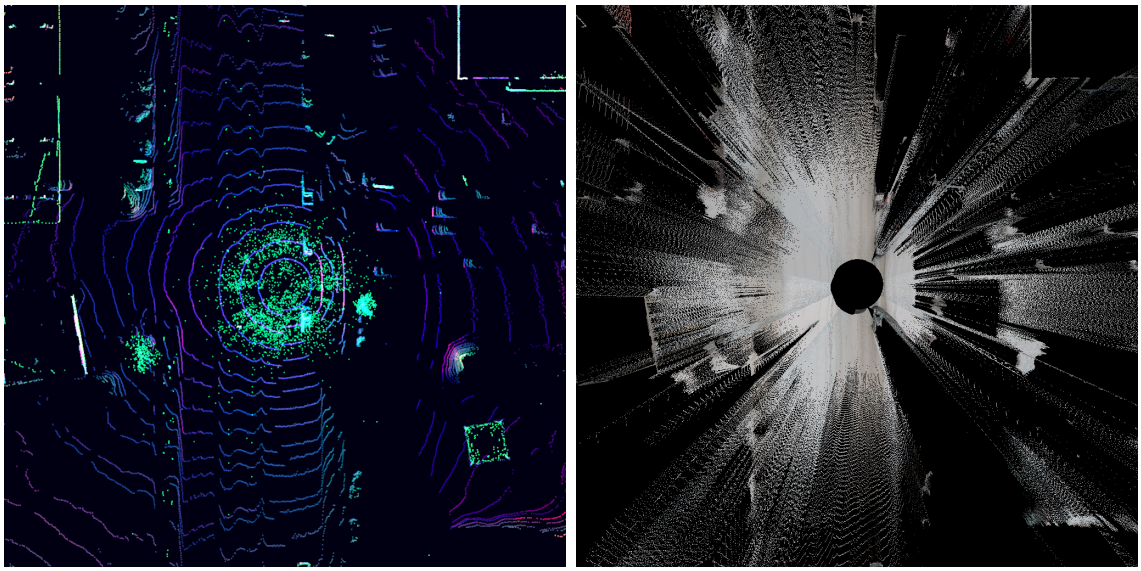


Figure 9: Dataset tool second phase block diagram for target frames



(a) LiDAR Frame

(b) RGB Frame

(c) Heightmap Frame

(d) Road Label Frame

Figure 10: Example of frames created by the dataset tool. (a), (b): source frames, (c), (d): target frames

3.1.1 First Phase

3.1.1.1 Point Cloud Algorithms

As the dataset tool relies extensively on usage of point clouds, several novel algorithms were developed to meet requirements of the dataset processing pipeline. Most of the proposed algorithms modify the point clouds with the exception of extended ICP, which provides an algorithm for point cloud aligment.

Outlier Stripping

LiDAR scans in presence of precipitation exhibit noise close to the sensor due to light reflecting from airborne particles. For the dataset tool to function correctly, LiDAR point clouds are to consist exclusively of points representing solid surfaces. To strip the scans of weather-related outliers, a relatively simple algorithm was developed.

For a point $\boldsymbol{p}$ to be preserved by the algorithm, it is required to have N_{min} other points in its neighbourhood, defined as space within Euclidean distance τ_d of point $\boldsymbol{p}$. τ_d is defined by

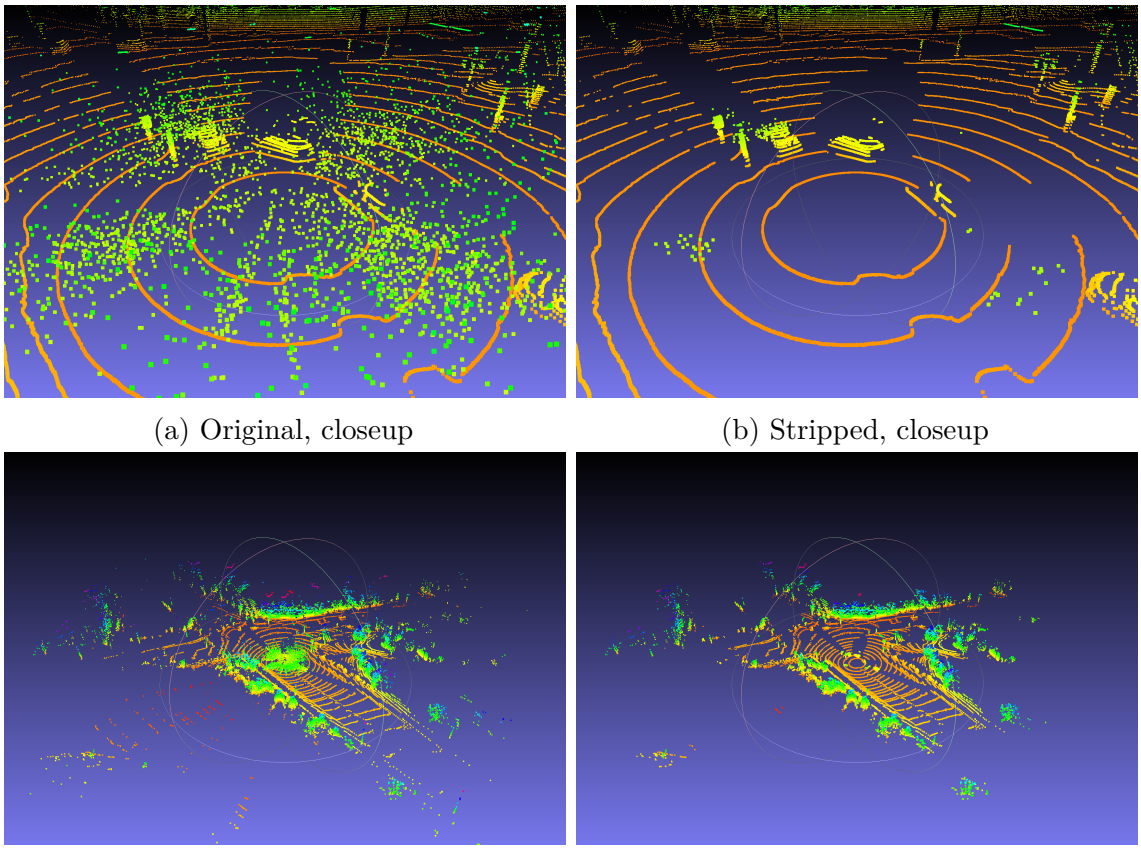
$$\tau_d = \tau + \alpha\|\boldsymbol{p}\| \tag{2}$$

where τ is the neighbourhood range threshold, α the distance modifier and $\|\boldsymbol{p}\|$ the point's distance from the origin(LiDAR sensor is defined to be positioned to the origin). Because distance between points increases linearly with the distance from origin, a distance correction term $\alpha\|\boldsymbol{p}\|$ is used to compensate for the increasing volumetric sparsity. The outlier stripping algorithm is described in Alg. 1. Effects of the algorithm are presented in Fig. 11. Note the effective reduction of precipitation noise and slight loss of information in distance.

Algorithm 1 LiDAR point cloud outlier stripping

Inputs:
$\mathcal{P}$: Point cloud to be processed
τ : Neighbourhood range threshold
α : Neighbourhood threshold distance modifier
N_{min} : Number of required neighbour points
Output:
$\mathcal{P}_{new}$: Processed point cloud

procedure StripOutliers($\mathcal{P}, \tau, \alpha, N_{\text{min}}$)
  $\mathcal{P}_{new} \leftarrow \{\}$ ▷ Stripped point cloud, initially empty
  for each $\boldsymbol{p} \in \mathcal{P}$ **do**
    $\tau_d \leftarrow \tau + \alpha\|\boldsymbol{p}\|$ ▷ Distance-corrected neighbourhood threshold
    $N \leftarrow 0$
    for each $\boldsymbol{q} \in \mathcal{P} \setminus \{\boldsymbol{p}\}$ **do** ▷ Count neighbours for point $\boldsymbol{p}$
      if $\|\boldsymbol{p} - \boldsymbol{q}\| < \tau_d$ **then**
        $N \leftarrow N + 1$
      end if
    end for
    if $N \geq N_{\text{min}}$ **then** ▷ Sufficient number of neighbours, point $\boldsymbol{p}$ is preserved
      $\mathcal{P}_{new} \leftarrow \mathcal{P}_{new} \cup \{\boldsymbol{p}\}$
    end if
  end for
  return $\mathcal{P}_{new}$
end procedure



(a) Original, closeup

(b) Stripped, closeup

(c) Original, full cloud

(d) Stripped, full cloud

Figure 11: Example of effects of the outlier stripping algorithm with parameters $\tau = 0, \alpha = 0.08, N_{\text{min}} = 25$.

Spherical Augmentation

LiDAR scanners designed for automotive applications typically have uneven beam distribution: more vertical resolution is allocated around the horizontal angle parallel to the ground. This is justifiable to enable greater resolution for obstacle and object detection but due to sparsely populated ground plane point clouds produced by such scanners are poorly suited for terrain reconstruction. To increase point cloud density in sparse areas, a method for augmenting the point clouds via spherical coordinate projection was developed. The algorithm is outlined as follows:

1. Project point cloud $\mathcal{P}$ into equirectangular radial distance map $\boldsymbol{R} \in \mathbb{R}^{s_h, s_v}$ and weight map $\boldsymbol{W} \in \mathbb{R}^{s_h, s_v}$. Points projected on the same element are weighted using kernel $k(\boldsymbol{q}, \boldsymbol{q_c})$, with projected point $\boldsymbol{q}$ and element centroid $\boldsymbol{q_c}$. s_h and s_v indicate the horizontal and vertical resolution, respectively.

2. Project each element in $\boldsymbol{R}$ back to a 3D point, unless the corresponding polar angle θ is outside the limits $[\theta_{\text{min}}, \theta_{max}]$. If the element is unassigned(corresponding weight in $\boldsymbol{W}$ is 0), distance for the element is interpolated between nearest assigned elements along the vertical dimension.

Each element $\boldsymbol{R_{i,j}}$ in the radial distance map is calculated as a weighted sum of the radial distance $r(q)$ of the projected points $\boldsymbol{q} \in \mathcal{Q}_{i,j}$ on the element. Each element $\boldsymbol{W_{i,j}}$ in the weight map is calculated as the sum of the weights $k(\boldsymbol{q}, \boldsymbol{q_c})$. Calculation of the radial distance and weight maps are presented in Eq. 3 and Eq. 4.

$$\boldsymbol{R_{i,j}} = \sum_{\boldsymbol{q} \in \mathcal{Q}_{i,j}} k(\boldsymbol{q}, \boldsymbol{q_c}) r(\boldsymbol{q}) \tag{3}$$

$$\boldsymbol{W_{i,j}} = \sum_{\boldsymbol{q} \in \mathcal{Q}_{i,j}} k(\boldsymbol{q}, \boldsymbol{q_c}) \tag{4}$$

For the weight kernel $k(\boldsymbol{q}, \boldsymbol{q_c})$, an inverse distance function defined by Eq. 5 is used. Intuitive rationale behind such kernel rises from the assumption that LiDAR measurements are accurate, so a point landing precisely in the middle of an element should get affected very little by surrounding points. ϵ is an non-zero stability term to prevent division by zero and exceedingly large weights.

$$k(\boldsymbol{q}, \boldsymbol{q_c}) = \frac{1}{\|\boldsymbol{q} - \boldsymbol{q_c}\| + \epsilon}, \{\boldsymbol{q}, \boldsymbol{q_c}\} \in \mathbb{R}^2, \epsilon > 0 \tag{5}$$

To create interpolated points along linear line between two vertically aligned elements corresponding to polar angles θ_1, θ_2 and distances d_1, d_2, a geometric problem illustrated in Fig. 12 needs to be solved. θ_3 is the polar angle of an element that lies within range $[\theta_1, \theta_2]$ and d_3 the unknown, corresponding distance required for 3D point projection. The problem can be expressed by forming an equation using the side-angle-side area formula for triangles. With minimal manipulation a formula for d_3 can be attained, as described by Eq. 9

$$\Delta\theta_{13} = \theta_3 - \theta_1 \tag{6}$$

$$\Delta\theta_{32} = \theta_2 - \theta_3 \tag{7}$$

$$\Delta\theta_{12} = \Delta\theta_{13} + \Delta\theta_{32} = \theta_2 - \theta_1 \tag{8}$$

$$\begin{aligned} \frac{1}{2} d_1 d_2 \sin\Delta\theta_{12} &= \frac{1}{2} d_1 d_3 \sin\Delta\theta_{13} + \frac{1}{2} d_3 d_2 \sin\Delta\theta_{32} \\ d_3 &= \frac{d_1 d_2 \sin\Delta\theta_{12}}{d_1 \sin\Delta\theta_{13} + d_2 \sin\Delta\theta_{32}} \end{aligned} \tag{9}$$

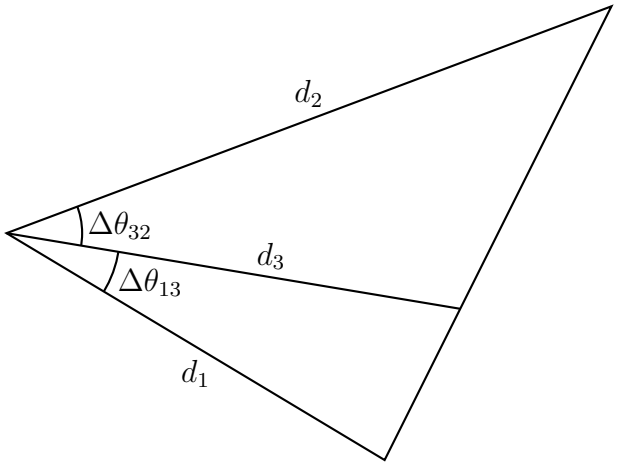


Figure 12: Geometric setup of point cloud spherical augmentation interpolation

Point projection with linear polar angle with respect to the element's vertical index results in a very dense point cloud for ground and objects near the vehicle. Furthermore, as most of the LiDAR beams are concentrated around the horizontal angle, vast majority of the projective resolution is used on areas void of points. Thus, details of the original point cloud will not be preserved without usage of prohibitively large resolution. (This is not particularly critical for the ground projection problem, but could be an issue for other problems, such as object detection) To address these issues, a nonlinear function was used for the projection polar angle. The nonlinear function $n(\theta|s_v, \gamma)$ and its inverse $n^{-1}(q_y|s_v, \gamma)$ are defined by Eq. 10 and Eq. 11 and plotted in Fig. 13. Effects of the nonlinearization scheme are illustrated in Fig. 14. Note the densely packed projected points in ground plane when using linear polar angle range and lack thereof when nonlinear angle range is used.

$$n(\theta|s_v, \gamma) = \begin{cases} \frac{s_v}{2} + \frac{s_v}{2}\left(\frac{\theta}{\pi/2}\right)^{\gamma} & , \theta > 0 \\ \frac{s_v}{2} - \frac{s_v}{2}\left(\frac{-\theta}{\pi/2}\right)^{\gamma} & , \theta < 0 \end{cases} \tag{10}$$

$$n^{-1}(q_y|s_v, \gamma) = \begin{cases} \frac{\pi}{2}\left(\frac{2q_y}{s_v} - 1\right)^{\frac{1}{\gamma}} & , q_y > \frac{s_v}{2} \\ -\frac{\pi}{2}\left(1 - \frac{2q_y}{s_v}\right)^{\frac{1}{\gamma}} & , q_y < \frac{s_v}{2} \end{cases} \tag{11}$$

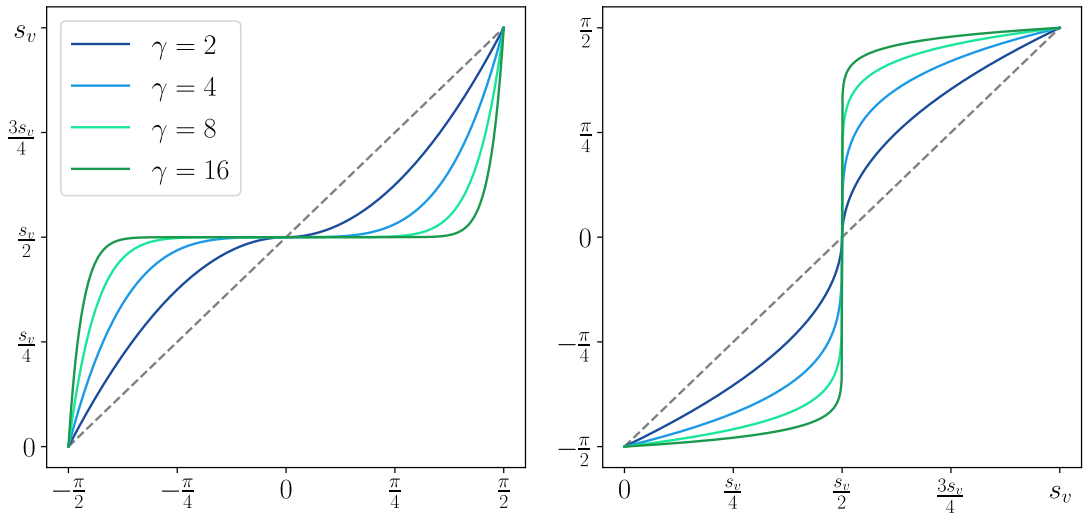


Figure 13: Polar angle nonlinearization function(left) and it's inverse(right) for point cloud spherical augmentation

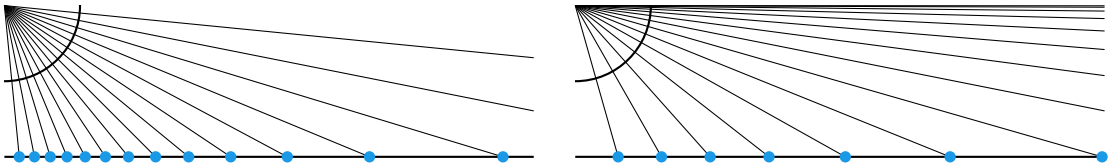


Figure 14: Effects of polar angle projection nonlinearization. Left: Constant angle difference, Right: Varying angle difference

Extended ICP

Iterative closest point (ICP)[51] is a widely used algorithm for aligning two 3D point clouds with rigid transformation (rotation and translation). A key aspect behind the algorithm's success is the fact that the rotation minimizing least-squares error between two point clouds sharing a centroid can be computed with singular value decomposition (SVD) of a single 3×3 matrix [52]. This gives the rotation alignment a computational complexity of $O(n)$, where n is the number of points.

The rotation alignment assumes paired point sets $\mathcal{P} = \{\boldsymbol{p_1}, \boldsymbol{p_2}, \ldots, \boldsymbol{p_N}\}$ and $\mathcal{P}' = \{\boldsymbol{p_1}, \boldsymbol{p_2}, \ldots, \boldsymbol{p_N}\}$ so that

$$\boldsymbol{p_i'} = \boldsymbol{R}\boldsymbol{p_i} + \boldsymbol{t} + \boldsymbol{\sigma_i} \tag{12}$$

where $\boldsymbol{R} \in \mathbb{R}^{3\times3}$ is the rotation matrix, $\boldsymbol{t} \in \mathbb{R}^3$ a translation vector, $\boldsymbol{\sigma_i}$ pointwise noise and $\boldsymbol{p_i}, \boldsymbol{p_i'} \in \mathbb{R}^3$ are the points before and after the transformation, respectively. $\boldsymbol{R}$ and $\boldsymbol{t}$ are computed to minimize the least-squares error e:

$$e = \sum_{i=1}^{N} \|\boldsymbol{p_i'} - (\boldsymbol{R}\boldsymbol{p_i} + \boldsymbol{t})\|^2 \tag{13}$$

However, in most practical cases, point clouds are not pre-paired and therefore point pairing is to be considered to belong in the problem domain. That is, matching point set $\mathcal{P}^{'}$ is sampled from the target point cloud $\mathcal{Q}$. For this purpose, Besl and McKay formalized an extension to the aligment algorithm titled iterative closest point. The full ICP algorithm is specified as follows:

1. On iteration j, for each point $\boldsymbol{p_{i,j}}$, find a matching(closest) target point from target point cloud $\mathcal{Q}$:

$$\boldsymbol{p}^{'}_{\boldsymbol{i,j}} = \arg\min_{\boldsymbol{q}\in\mathcal{Q}} \|\boldsymbol{q} - \boldsymbol{p_{i,j}}\| \tag{14}$$

2. Compute the alignment parameters $\boldsymbol{R}$ and $\boldsymbol{t}$ so that alignment error e_j (Eq. 13) is minimized.

3. Apply the alignment:

$$\begin{aligned} \boldsymbol{p_{i,j+1}} &= \boldsymbol{R}\boldsymbol{p_{i,j}} + \boldsymbol{t} \\ i &\in 1, 2, \ldots, N \end{aligned} \tag{15}$$

4. Terminate the iteration if sufficiently small change in aligment error has been reached: $\Delta e_j < \epsilon$, where $\Delta e_j = e_{j-1} - e_j$ and $\epsilon > 0$ is sufficiently small constant indicating the desired iteration precision. Otherwise, proceed to next iteration $j + 1$.

The standard ICP algorithm potentially fails to converge to optimal alignment unless sufficient initial conditions are met. These conditions include low noise, cloud point distribution similarity and initial alignment. The reference point clouds are noisy due to the aggregation of LiDAR scans. Furthermore, they exhibit high variability in density as the resolution of the point cloud drops with respect to distance from the sensor according to the inverse-square law.

It was noticed that alignment to the reference cloud(section 3.1.1.2) had high convergence failure rate when using standard ICP algorithm due to the limitations discussed above. To address these issues, an extension to the algorithm was developed. The extension proposes replacement for point matching step defined by Eq. 14. Instead of selecting matching point $\boldsymbol{p}^{'}_{\boldsymbol{i,j}}$ directly from target point cloud $\mathcal{Q}$, it is computed by averaging points in neighbourhood $\mathcal{S}_{i,j}$ defined by space within radius r from the point $\boldsymbol{p_{i,j}}$. In case this neighbourhood does not contain any points, the point is selected as in the standard ICP algorithm. The extension is formulated in Eq. 16 and its effectiveness in the presence of noise, density distribution dissimilarity and poor initial alignment is presented in the Fig. 15, Fig. 16 and Fig. 17 respectively.

$$\begin{aligned} \mathcal{S}_{i,j} &= \{\boldsymbol{q} \in \mathcal{Q} : \|\boldsymbol{q} - \boldsymbol{p_{i,j}}\| < r\} \\ \boldsymbol{p}^{'}_{\boldsymbol{i,j}} &= \begin{cases} \dfrac{1}{|\mathcal{S}_{i,j}|} \displaystyle\sum_{\boldsymbol{s}\in\mathcal{S}_{i,j}} \boldsymbol{s}, & |\mathcal{S}_{i,j}| > 0 \\ \arg\min\limits_{\boldsymbol{q}\in\mathcal{Q}} \|\boldsymbol{q} - \boldsymbol{p_{i,j}}\|, & |\mathcal{S}_{i,j}| = 0 \end{cases} \end{aligned} \tag{16}$$

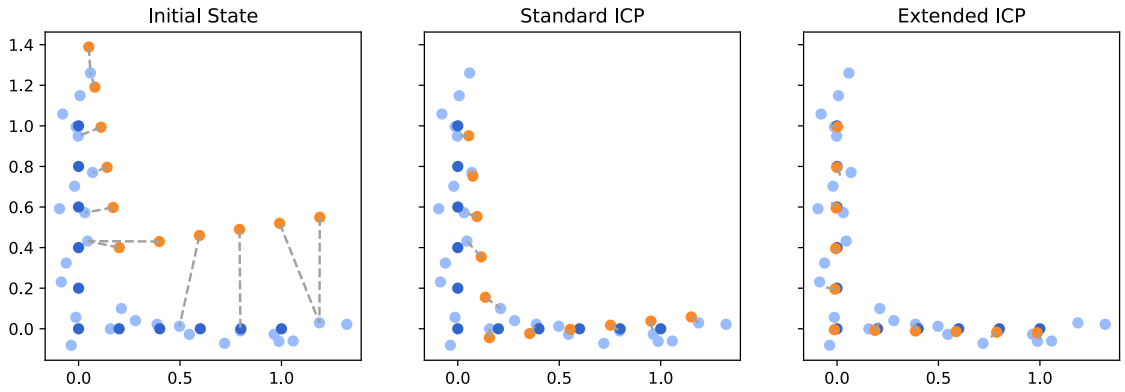


Figure 15: ICP alignment in the presence of noise in the target point cloud. Point cloud $\mathcal{P}$ to be aligned (orange), target point cloud $\mathcal{Q}$ (light blue) and optimally aligned point cloud (dark blue).

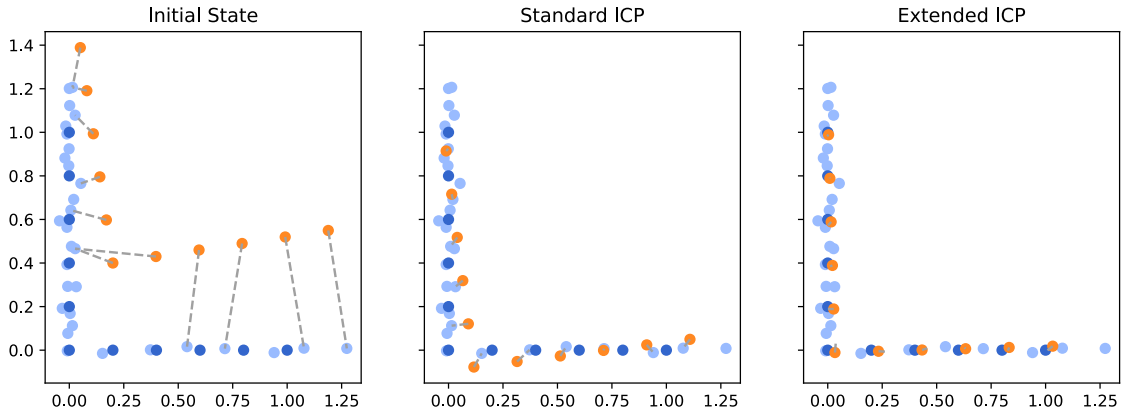


Figure 16: ICP aligment in the presence of density distribution dissimilarity

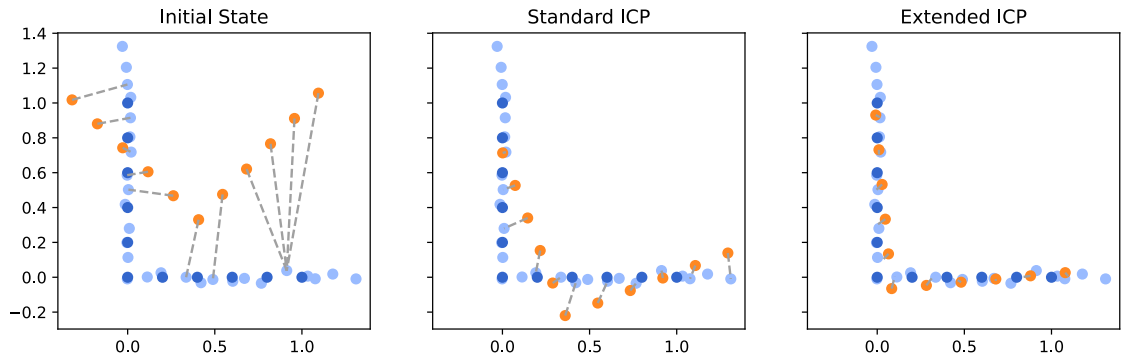


Figure 17: ICP aligment in case of poor initial alignment

Query Optimization with k-d Trees

k-d tree is a data structure for partitioning data of dimensionality k. Primary objective of the k-d tree partitioning is to enable efficient queries of logarithmic complexity on the data. k-d tree was invented by Bentley [53] and is frequently utilized when handling sparse data of relatively low dimensionality, such as 3D point clouds.

In the dataset tool a custom k-d tree is implemented to achieve efficient subcloud queries required by the proposed ICP extension. More specifically, the k-d tree enables cache-efficient computation of averaged neighbourhood point $\boldsymbol{p}'_{i,j}$, as specified in Eq. 16. This is achieved by not keeping neighbourhood points in the memory but performing the point summation inline with the tree iteration instead.

Stochastic Density Adjustment

Merging augmented LiDAR point clouds results in large point clouds with dense local clusters of points. Such clusters contain redundant points and thus cause distortion on methods operating on local regions of points(such as the extended ICP). Furthermore, large point clouds consisting of hundreds of millions of points are prohibitely slow to process on current PC hardware. A relatively straightforward method for reducing local density variance is to remove points stochastically with probability proportional to the number of neighbouring points. Stochastic approach was chosen due to its simplicity and robustness against potential aliasing of deterministic algorithms.

The density adjustment algorithm is described in Alg. 2: first, in similar fashion to outlier stripping, number of neighbourhood points are counted. After this a removal probability $\phi_{\boldsymbol{p}}$ for each point $\boldsymbol{p}$ is calculated; it is determined by largest initial probability Φ_{max}(number of points in densest neighbourhood), base removal probability ϕ_0 and nonlinearity factor γ. With values $\gamma > 1$ the removal process will concentrate on the densest clusters whereas values of $\gamma \in (0, 1)$ will leave only the sparsest areas untouched. $\phi_{\boldsymbol{p}}$ is computed as presented in Eq. 17.

$$\phi_{\boldsymbol{p}} = \phi_0 + (1 - \phi_0) \left(\frac{N}{\Phi_{max}} \right)^{\gamma} \tag{17}$$

Algorithm 2 Stochastic point cloud density adjustment

Inputs:
$\mathcal{P}$: Point cloud to be processed
τ : Neighbourhood range threshold
ϕ_0 : Base removal probability
γ : Removal probability nonlinearity
Output:
$\mathcal{P}_{new}$: Processed point cloud

procedure AdjustDensity($\mathcal{P}, \tau, \phi_0, \gamma$)
  $\Phi \leftarrow \{\}$ ▷ Set of removal probabilities
  for each $\boldsymbol{p} \in \mathcal{P}$ **do**
    $N \leftarrow 0$
    for each $\boldsymbol{q} \in \mathcal{P} \setminus \{\boldsymbol{p}\}$ **do** ▷ Count neighbours for point $\boldsymbol{p}$
      if $\|\boldsymbol{p} - \boldsymbol{q}\| < \tau$ **then**
        $N \leftarrow N + 1$
      end if
    end for
    $\phi_{\boldsymbol{p}} \leftarrow N$ ▷ Probability of removal of point $\boldsymbol{p}$
    $\Phi \leftarrow \{\Phi, \phi_{\boldsymbol{p}}\}$
  end for
  $\Phi_{max} \leftarrow \max(\Phi)$
  for each $\boldsymbol{p} \in \mathcal{P}$ **do**
    $\phi_{\boldsymbol{p}} \leftarrow \phi_0 + (1 - \phi_0)\left(\frac{\phi_{\boldsymbol{p}}}{\Phi_{max}}\right)^{\gamma}$ ▷ Update $\phi_{\boldsymbol{p}}$ to final value
  end for
  $\mathcal{P}_{new} \leftarrow \{\}$ ▷ Density adjusted point cloud, initially empty
  for each $\boldsymbol{p} \in \mathcal{P}$ **do**
    if $\phi_{\boldsymbol{p}} < (X \sim U(0,1))$ **then** ▷ $U(0,1)$ denotes an uniform pr. distribution
      $\mathcal{P}_{new} \leftarrow \mathcal{P}_{new} \cup \{\boldsymbol{p}\}$
    end if
  end for
  return $\mathcal{P}_{new}$
end procedure

3.1.1.2 Transformation Registration

Producing an accurate heightmap requires precise transformation for each LiDAR scan in a driving sequence. A naive implementation would use a local alignment algorithm (such as the ICP) to obtain transformations between consecutive scans. However, such approach would cause drifting - accumulation of error on subsequent transformations. Fortunately, the CADC dataset provides good quality GNSS+INS localization data. Despite precise localization, using raw transformations based on GNSS+INS data alone would not provide sufficient point cloud aligment further away from the vehicle due to the orientation inaccuracies.

To compute accurate transformations for each frame, a two-step approach is

proposed: first, a reference point cloud is formed based on the GNSS+INS localization data. Second, each LiDAR point cloud frame is aligned to the reference point cloud using the extended ICP algorithm. This method combines the benefits of the two naive approaches, providing precise alignment on entire extent of each frame without drifting error.

Reference Point Cloud Formation

To merge LiDAR point clouds of all frames into a global reference point clouds, relative transformations between the point clouds need to be acquired. By treating the initial vehicle transformation as origin, transformation $\boldsymbol{Q_i} \in \mathbb{R}^{4\times4}$ for frame i can be expressed with initial orientation $\boldsymbol{R_0} \in \mathbb{R}^{4\times4}$, frame i orientation $\boldsymbol{R_i} \in \mathbb{R}^{4\times4}$ and initial frame to frame i translation $\boldsymbol{T_{0i}} \in \mathbb{R}^{4\times4}$, as presented in Eq. 18

$$\boldsymbol{Q_i} = \boldsymbol{R_0^{-1}} \boldsymbol{T_{0i}} \boldsymbol{R_i} \tag{18}$$

$$\boldsymbol{R_i} = \boldsymbol{R}_z(\alpha)\boldsymbol{R}_y(\beta)\boldsymbol{R}_x(\gamma) \tag{19}$$

$$\boldsymbol{T_{0i}} = \begin{bmatrix} 1 & 0 & 0 & \Delta x \\ 0 & 1 & 0 & \Delta y \\ 0 & 0 & 1 & \Delta z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{20}$$

The transformations are expressed as 4×4 homogeneous matrices, enabling compact notation that includes translations. $\boldsymbol{R}_z$, $\boldsymbol{R}_y$ and $\boldsymbol{R}_x$ denote rotations along the x, y and z axes, respectively. α, β and γ denote yaw(azimuthal angle), pitch and roll, respectively. Δx, Δy and Δz denote translation along each axis. Angles and z-position (altitude) are readily available in the GNSS+INS data provided by the dataset. Δx and Δy need to be attained by a projection from latitude / longitude coordinates. This is done by planar projection with initial location as reference, using WGS84 coordinate system standard and an open source library for cartesian conversion [1]. Curvature of the earth is negligible for distances present in the dataset, justifying the approximation.

If LiDAR scans were to be merged unprocessed, the resulting point cloud would exhibit density variance to the extent of making subsequent alignment unstable. Particularly the ground would suffer from point scarcity due to the fact that most of the LiDAR scan lines are focused around horizontal angle. To address this issue, point clouds are first processed with the spherical augmentation algorithm.

However, due to the interpolative nature of the spherical augmentation algorithm, it is extremely sensitive to airborne noise caused by precipitation. Therefore the noise is to be first filtered out with the outlier stripping algorithm.

After the merging, the global point cloud will again suffer from uneven density caused by the vehicle trajectory and the projective nature of the original point clouds. The merged point cloud is denser around areas where the vehicle has moved slowly or remained stationary due to accumulation of scans 10 times per second,

[1] https://github.com/chrberger/WGS84toCartesian

regardless of the vehicle speed. Additionally, points cloud sections closer to origin are inherently denser than those further away. Furthermore, the resulting point clouds are prohibitively large to be used in the alignment phase. These problems can be synergistically solved by removing points from the densest sections with the stochastic density adjustment algorithm. The algorithm is repeatedly with increasing range threshold τ and nonlinearity factor γ until sufficiently low point count has been reached.

Aligment to Reference Point Cloud

Due to the uneven distribution of points in LiDAR scans, they are not readily suitable for alignment with the Extended ICP algorithm. In similar fashion to the processing done in the reference point cloud formation phase, the clouds are first stripped of noise and spherically augmented. In addition, stochastic density adjustment is applied to increase alignment stability.

After alignment, the transformations are stored for future use. In principle, the alignment phase could be executed in iterative manner by forming a new reference point cloud from the aligned clouds. However in this application, it was found to be sufficient to perform the alignment once.

3.1.1.3 Heightmap Formation

Ground level estimator for a driving sequence s is a function defined by $z_g = h(x, y|s)$, which gives a ground level estimate z_g (w.r.t. sequence origin) for point (x, y) on the horizontal plane. For the estimator to be feasible, its computational cost is required to be sufficienty low to accomodate for ground area covered by a single driving sequence.

If a 3D point cloud were to be used directly as the estimator, evaluation of h would involve point search over the cloud and processing of multiple points, giving such algorithm (with acceleration structure such as k-d tree) a lower bound complexity of $O(log(n) + m)$, where n is the number of points in the cloud and m the number of points involved in the ground level estimate processing. To project the RGB images and form the heightmap frames for a single sequence, the estimator needs to be evaluated $\sim 2 \cdot 10^7$ times. With a ordinary, modern PC computer, this approach would not yield results in reasonable time frame. Instead, the point clouds are stored into a 2D column sampling data structure that can be used to effectively produce a ground level estimate and subsequent heightmap.

Column Sampling Data Structure

As the ground estimator h is a function of a 2D point, a reasonable starting point for its formation is to reduce the 3D point cloud into a 2D form. To achieve this, the point cloud is first discretized into a data structure consisting of point columns of size $l \times l$. A column is implemented as a dynamic array of point z-coordinates (x and y are omitted in discretization). l denotes the desired cell edge size of the resulting heightmap in real-world units. To allow for efficient and flexible memory usage, the

data is structured into blocks of $W \times H$ columns. This enables the structure grow spatially indefinitely without reallocation or wasting memory on empty areas.

To produce a reliable ground level estimate, a couple of factors are to be considered. Firstly, point distribution along z-dimension might be multimodal and/or not centered around the ground level; such is the case when overhead (traffic signs, power lines, trees, bridges) or non-stationary (vehicles, pedestrians) objects are present. Secondly, for desired heightmap resolution there might be little or no points occuring inside each column, resulting in severe noise or "holes" in the heightmap.

To address these issues, spatial filtering methods are used. Inside a column, point distribution is modeled with *kernel density estimation* (KDE) [54, 55]. The resulting distributions are horizontally filtered to fill and smooth the defects caused by insufficient number of column points.

Ground Level Estimation with KDE

First vertical point density distributions are formed using gaussian kernels with constant bandwidth σ. The vertical kernel density estimator $f_c(z)$ for column c is presented in Eq. 21. ($\mathcal{P}_c$ denotes points inside a column c) As evaluation of the density function involves computing kernel with respect to each point in the column, horizontal filtering would become computationally infeasible. Therefore an approximation method for the density function was developed.

$$f_c(z) = \frac{1}{|\mathcal{P}_c|} \sum_{\boldsymbol{p} \in \mathcal{P}_c} k(p_z - z) \tag{21}$$

$$k_v(z|\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{z}{\sigma}\right)^2} \tag{22}$$

Key insight for developing the density function approximator is the fact that column points tend to form clusters representing ground and objects in the scene. These clusters are approximately normally distributed and can therefore modeled as macroscopic, (Larger than the kernel $k_v(z|\sigma)$)parametrized gaussians. This reduces the number of gaussian evaluations per column from the number of points ($|\mathcal{P}_c|$) to number of clusters (< 5 in vast majority of cases).

Parametrized gaussian $g(z|a, \mu, \sigma)$ is presented in Eq. 23 with a as peak height, μ as center of the peak and σ as the width of the gaussian. The quadratic error function is defined as a squared sum of differences of approximation $\hat{f}_c(z)$ and original density function $f_c(z)$ at evaluation points $\mathcal{Z}_E$. It is presented in Eq. 24.

$$g(z|a, \mu, \sigma) = a e^{-\frac{1}{2}\left(\frac{z-\mu}{\sigma}\right)^2} \tag{23}$$

$$E_c = \sum_{z \in \mathcal{Z}_E} (\hat{f}_c(z) - f_c(z))^2 \tag{24}$$

The outline for the approximator formation algorithm is defined as follows:

1. Find arguments of the local maxima $\mathcal{Z}_{max}$ of $f_c(z)$ using Newton's method and fixed interval sampling for initial points.

2. Initialize parametrized gaussians according to the maxima (with $\mu \in \mathcal{Z}_{max}$ and $a \in f(\mathcal{Z}_{max})$).

3. Fit parametrized gaussians into the density function $f_c(z)$ using gradient descent for a quadratic error function E_c.

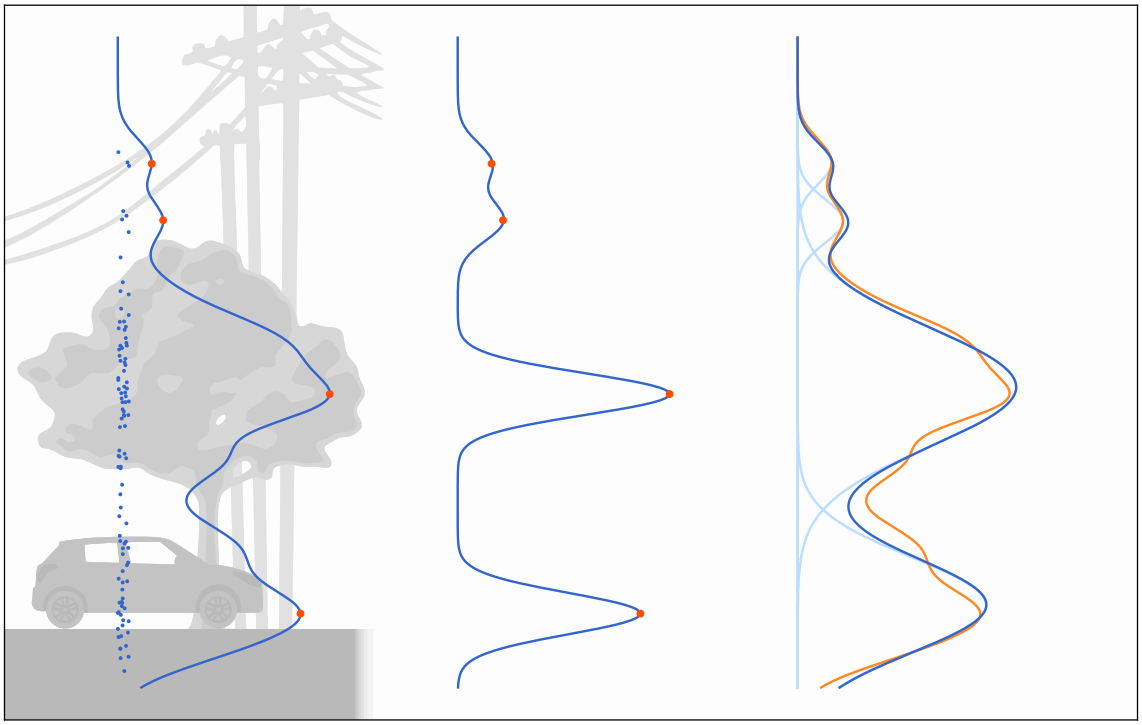


Figure 18: KDE approximation process: Point sample column and full KDE $f_c(z)$ (left), initial approximation with parametrized gaussians based on peaks only (middle) and final fitted approximation $\hat{f}_c(z)$ (right, blue) compared to full KDE (right, orange)

As a result of its computationally optimized nature, the KDE approximator function $\hat{f}_c(x)$ is suitable for horizontal filtering. Horizontal filtering is implemented as a weighted sum of columns $\mathcal{C}$ (Eq. 26) using a 2D gaussian weight kernel $k_h(\Delta x, \Delta y)$, presented in Eq. 25.

$$k_h(\Delta x, \Delta y|\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{\Delta x^2+\Delta y^2}{\sigma^2}\right)} \tag{25}$$

$$\tilde{f}_c(z) = \sum_{d\in\mathcal{C}} k_h(x_d - x_c, y_d - y_c|\sigma) f_d(z) \tag{26}$$

The filtered density estimate $\tilde{f}_c(z)$ is used for ground level estimation. First, maxima for the filtered estimate $\tilde{f}_c(z)$ are found in similar manner to the original KDE $f_c(z)$. The ground level estimate for a column c is chosen to be the lowest argument of a maximum corresponding to a value of at least of 10% of the global maximum max $\tilde{f}_c(z)$. The soft selection is justified by the fact that there are unlikely

any significant peaks under the ground level besides outliers caused by noise or point cloud misalignment.

3.1.1.4 RGB Image Projection

To form the global BEV RGB image as a reference for human labeler, the RGB camera images are to be projected against the heightmap. Image distortion correction, camera parameters and corresponding depth image are necessary for successful projection. Distortion correction calibration parameters as well as intrinsic and extrinsic parameters for each camera are readily provided in the CADC dataset. Depth image is formed by virtually imaging the 3D heightmap using ray tracing and storing length-to-contact for each ray.

Camera Parameters and Distortion Correction

A homogeneous 3D world point $\boldsymbol{p_w} \in \mathbb{R}^4$ can be projected into point in pixel coordinates $\boldsymbol{p_p} \in \mathbb{R}^3$ with a *pinhole camera model*, in which camera is modeled with two matrices. *Extrinsic parameter matrix* $\left[\boldsymbol{R}|\boldsymbol{t}\right]$ encodes the camera's rotation in matrix $\boldsymbol{R} \in \mathbb{R}^{3\times3}$ and its translation in vector $\boldsymbol{t} \in \mathbb{R}^3$. With the extrinsic parameter matrix point in world space is transformed into point in camera space $\boldsymbol{p_c}$, as presented in Eq. 27. This point is projected into pixel coordinates with *intrinsic parameter matrix* $\boldsymbol{A}$ formed from focal lengths f_x, f_y and principal point location (c_x, c_y) — presented in Eq. 30. The projection is presented in Eq. 28. These operations can be combined to transform world point directly into pixel coordinates as presented in Eq. 29.

$$\boldsymbol{p_c} = \left[\boldsymbol{R}|\boldsymbol{t}\right] \boldsymbol{p_w} \tag{27}$$

$$\boldsymbol{p_p} = \boldsymbol{A}\boldsymbol{p_c} \tag{28}$$

$$\boldsymbol{p_p} = \boldsymbol{A} \left[\boldsymbol{R}|\boldsymbol{t}\right] \boldsymbol{p_w} \tag{29}$$

$$\boldsymbol{A} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{30}$$

$$\tag{31}$$

With $p_{wz} \neq 0$, the post-depth-division projection can be reformed into Eq. 32. In this form the distortion correction can be applied by replacing p_{wx}/p_{wz} and p_{wy}/p_{wz} with x'' and y'' as presented in Eq. 33. For the distortion correction, a popular open source computer vision library, OpenCV [56], was used. OpenCV uses a distortion correction model presented in Eq. 34, Eq. 35 and Eq. 36 [57]. $k_{1\cdots6}$ are the radial distortion coefficients, q_1, q_2 the tangential distortion coefficients and $s_{1\cdots4}$ the thin prism distortion coefficients.

$$\begin{bmatrix} p_{px} \\ p_{py} \end{bmatrix} = \begin{bmatrix} f_x(p_{wx}/p_{wz}) + c_x \\ f_y(p_{wy}/p_{wz}) + c_y \end{bmatrix} \tag{32}$$

$$\begin{bmatrix} p_{px} \\ p_{py} \end{bmatrix} = \begin{bmatrix} f_x x'' + c_x \\ f_y y'' + c_y \end{bmatrix} \tag{33}$$

$$\begin{bmatrix} x'' \\ y'' \end{bmatrix} = \begin{bmatrix} x' \frac{1+k_1 r^2+k_2 r^4+k_3 r^6}{1+k_4 r^2+k_5 r^4+k_6 r^6} + 2q_1 x' y' + q_2(r^2+2x'^2) + s_1 r^2 + s_2 r^4 \\ y' \frac{1+k_1 r^2+k_2 r^4+k_3 r^6}{1+k_4 r^2+k_5 r^4+k_6 r^6} + q_1(r^2+2y'^2) + 2q_2 x' y' + s_3 r^2 + s_4 r^4 \end{bmatrix} \tag{34}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} p_{wx}/p_{wz} \\ p_{wy}/p_{wz} \end{bmatrix} \tag{35}$$

$$r^2 = x'^2 + y'^2 \tag{36}$$

In practice, the distorted images are undistorted to enable world points to be transformed into pixel positions, and pixel positions to be transformed into projective lines with a single linear matrix $\boldsymbol{A}\left[\boldsymbol{R}|\boldsymbol{t}\right]$ (and its inverse). The latter is paramount for production of corresponding depth images by ray tracing.

Ray tracing

Ray tracing (and especially its Monte Carlo variant, path tracing) is a popular method for offline(non-realtime) rendering. It is used in numerous computer graphics applications, including architectural rendering, cinematic visual effects and physics simulation rendering. Ray tracing is based on the concept of simulating light transport in reverse manner; instead of simulating rays of light emitted from a light source, they are "emitted" from the camera into the scene geometry [58].

Generally a ray tracing algorithm is composed of three phases: 1) ray-geometry intersections, 2) transmission on geometry contact(reflection, refraction, dispersion) and 3) material modeling. In the context of this work it is not of an interest to produce photorealistic imagery but to render a matching depth image corresponding to an undistorted camera view, so phases 2) and 3) can be omitted.

The ray is modeled with a point of origin $\boldsymbol{o}$, a normalized direction vector $\hat{\boldsymbol{d}}$ and ray length t, as presented in Eq. 37. Geometry for each column is modeled with bilinear interpolation, with corner heights $\boldsymbol{Q} \in \mathbb{R}^{2\times 2}$, as presented in Eq. 38 and Eq. 39. For a local point $\boldsymbol{s}$ inside a column($s_x, s_y \in [0,1]$), height difference to the bilinear surface can be calculated with $g(\boldsymbol{s}|\boldsymbol{Q})$, presented in Eq. 40. Every point s with $g(\boldsymbol{s}|\boldsymbol{Q}) < 0$ is considered to be below the ground and therefore a geometry contact. By replacing s with the parametrization of a localized ray $\boldsymbol{r}(t|\tilde{\boldsymbol{o}}, \tilde{\boldsymbol{d}})$ and finding roots for ray length t the equation presented in Eq. 41, the contact point can be resolved. Ray normalization is done with the equations Eq. 42, Eq. 43, Eq. 44 and Eq. 45.

$$\boldsymbol{r}(t|\boldsymbol{o},\hat{\boldsymbol{d}}) = \boldsymbol{o} + t\hat{\boldsymbol{d}}, \qquad \boldsymbol{o},\hat{\boldsymbol{d}} \in \mathbb{R}^3, \|\hat{\boldsymbol{d}}\| = 1 \tag{37}$$

$$i(x|a,b) = a + x(b-a), \qquad x \in [0,1] \tag{38}$$

$$i_2(x,y|\boldsymbol{Q}) = i(y|i(x|\boldsymbol{Q_{11}},\boldsymbol{Q_{12}}), i(x|\boldsymbol{Q_{21}},\boldsymbol{Q_{22}})), \qquad x,y \in [0,1], \boldsymbol{Q} \in \mathbb{R}^{2\times 2} \tag{39}$$

$$g(\boldsymbol{s}|\boldsymbol{Q}) = s_z - i_2(s_x,s_y|\boldsymbol{Q}), \qquad \boldsymbol{s} \in \mathbb{R}^3, s_x, s_y \in [0,1] \tag{40}$$

$$g(\boldsymbol{r}(t|\tilde{\boldsymbol{o}},\tilde{\boldsymbol{d}})|\boldsymbol{Q}) = 0 \tag{41}$$

$$\tilde{\boldsymbol{o}}_{\boldsymbol{x}} = (o_x - c_{x1})/(c_{x2} - c_{x1}) \tag{42}$$

$$\tilde{\boldsymbol{o}}_{\boldsymbol{y}} = (o_y - c_{y1})/(c_{y2} - c_{y1}) \tag{43}$$

$$\tilde{\boldsymbol{d}}_{\boldsymbol{x}} = d_x/(c_{x2} - c_{x1}) \tag{44}$$

$$\tilde{\boldsymbol{d}}_{\boldsymbol{y}} = d_y/(c_{y2} - c_{y1}) \tag{45}$$

With respect to t, $g(\boldsymbol{r}(t|\tilde{\boldsymbol{o}},\tilde{\boldsymbol{d}})|\boldsymbol{Q})$ can be refactored into a second order polynomial, as presented in Eqs. 46, 47, 48 and 49. Therefore, contact points are trivially resolvable with the quadratic formula.

$$g(\boldsymbol{r}(t|\tilde{\boldsymbol{o}},\tilde{\boldsymbol{d}})|\boldsymbol{Q}) = at^2 + bt + c, \tag{46}$$

$$a = \hat{d}_x\hat{d}_y(\boldsymbol{Q_{12}} + \boldsymbol{Q_{21}} - \boldsymbol{Q_{11}} - \boldsymbol{Q_{22}}) \tag{47}$$

$$\begin{aligned} b = {} & \hat{d}_z + \boldsymbol{Q_{11}}(\hat{d}_x + \hat{d}_y - \hat{d}_y\tilde{o}_x - \hat{d}_x\tilde{o}_y) - \boldsymbol{Q_{22}}(\hat{d}_y\tilde{o}_x + \hat{d}_x\tilde{o}_y) \\ & + \boldsymbol{Q_{12}}(\hat{d}_y\tilde{o}_x + \hat{d}_x\tilde{o}_y - \hat{d}_x) + \boldsymbol{Q_{21}}(\hat{d}_y\tilde{o}_x + \hat{d}_x\tilde{o}_y - \hat{d}_y) \end{aligned} \tag{48}$$

$$\begin{aligned} c = {} & \tilde{o}_z - \boldsymbol{Q_{11}}(1 + \tilde{o}_x + \tilde{o}_y - \tilde{o}_x\tilde{o}_y) - \tilde{o}_x\boldsymbol{Q_{12}} - \tilde{o}_y\boldsymbol{Q_{21}} \\ & + \tilde{o}_x\tilde{o}_y(\boldsymbol{Q_{12}} + \boldsymbol{Q_{21}} - \boldsymbol{Q_{22}}) \end{aligned} \tag{49}$$

To create a ray, its origin is chosen to be the position of the camera "pinhole" and its direction a vector in the direction of negative z axis, projected with inverse of the camera matrix. This produces a ray for each pixel "traveling" to negative direction with respect to light rays in undistorted RGB image. Each ray may contact the ground geometry at multiple points, of which the nearest (smallest t) is chosen for the corresponding pixel.

In practice, computing every intersection between multiple megapixel size images and 10^7 columns ($\approx 10^{14}$ intersections for each frame) is a computationally unfeasible task, raising requirement for optimization. Typically an acceleration structure is used, reducing the complexity of the rendering algorithm from $O(n)$ to $O(\log(n))$, where n is the number of geometry elements. Most common such structure is the *bounding volume hierarchy* (BVH) [58, Chapter 4.3], in which the geometry elements are grouped into hierarchically growing groups with bounding volumes surrounding them. In such structure each ray is required to be intersected at most with number of volumes equal to the depth of the hierarchy.

However, general bounding volumes can be computationally demanding to construct to enable reasonable performance for all kinds of geometries. Fortunately, in this work the geometry is well defined pseudo-dense grid pattern in x and y

directions, allowing for a different approach. The square nature of each column block is exploited to form a 2.5D quadtree [59], each level of the hierarchy being an *axis-aligned bounding box* (AABB) with minimum and maximum height values chosen so that all successive levels are contained within the limits.

3.1.2 Second Phase

RGB and LiDAR BEV frames are both tensors with dimensions of $N \times N \times 3$. As such, they can be represented as regular RGB images (with a floating point data type). With the vehicle origin centered at $(N/2, N/2)$, a frame maps to a surrounding area of $Nl \times Nl$, where l is the cell edge length in real-world units.

The LiDAR frame channels encode density, height and intensity of the LiDAR point cloud. Density is simply the number of points inside the $l \times l$ column. Height and intensity are the average z-position and reflection intensity of the points.

For the RGB frame BEV projection, the RGB images are required to be projected against 3D environment geometry. Naively the environment could be modeled as a plane below the vehicle. However, such approach would cause severe projective distortions in environments with height variations and obstacles. In the dataset tool a proposed algorithm presented in detail in section 3.1.2.2 is used; taking advantage of the line sweep structure of the LiDAR point clouds, the point cloud can be triangulated into a triangle mesh in real-time. GPU:s are designed primarily for effectively rasterizing triangle meshes and therefore the resulting mesh can be projected into the camera views to form depth images corresponding to the RGB images. Finally with the help of the depth images, the RGB images can be projected into 3D points to form the BEV image. For the BEV image color of the topmost point is chosen.

3.1.2.1 LiDAR Frame

BEV LiDAR frames are formed directly from the LiDAR scan point clouds by projection and simple averaging operations for maximum efficiency. Each cell (pixel) $p_{\text{LiDAR,c}} \in \mathbb{R}^2$ corresponding to column c in the frame encodes a column point density ρ_c, average height $z_{\text{c,avg}}$ and average reflective intesity $\Phi_{\text{c,avg}}$, as presented in Eq. 51.

$$p_{\text{LiDAR,c}} = \begin{bmatrix} x_{\text{LiDAR,c}} \\ y_{\text{LiDAR,c}} \end{bmatrix} \tag{50}$$

$$d_{\text{LiDAR,c}} = \begin{bmatrix} \rho_c \\ z_{\text{c,avg}} \\ \Phi_{\text{c,avg}} \end{bmatrix} \tag{51}$$

$$\rho_c = |\mathcal{P}_c| \tag{52}$$

$$z_{\text{c,avg}} = \frac{1}{\rho_c} \sum_{\boldsymbol{p} \in \mathcal{P}_c} z_{\boldsymbol{p}} \tag{53}$$

$$\Phi_{\text{c,avg}} = \frac{1}{\rho_c} \sum_{\boldsymbol{p} \in \mathcal{P}_c} \Phi_{\boldsymbol{p}} \tag{54}$$

3.1.2.2 RGB Frame

Producing a BEV RGB frame is vastly less trivial than producing a BEV LiDAR frame due to the fact that there exists no 3D structure in 2D RGB images. Therefore, the RGB images have to be projected into 3D points. To achieve this, corresponding depth images providing distance from the camera origin for each pixel are required.

Furthermore, timeframe for the depth image creation process is limited due to the real-time constraints discussed in Section 3.1. The proposed solution takes advantage of hardware rasterization capabilities of a modern *graphics processing unit* (GPU). By triangulating the LiDAR point cloud into a triangle mesh, it can be rendered into depth images by utilizing the camera parameters provided in the CADC dataset.

Finally each RGB image is back-projected into a 3D point cloud and projected into BEV image. For a RGB pixel value $d_{\mathrm{RGB,c}} = \begin{bmatrix} r_{\mathrm{RGB,c}} & g_{\mathrm{RGB,c}} & b_{\mathrm{RGB,c}} \end{bmatrix}^T$ corresponding to column c the RGB value of the topmost (highest z value) point is chosen.

LIDAR Point Cloud Triangulation

General algorithms for reconstructing a triangle mesh provide high quality results but require additional spatial analysis for coherent results. One such method is a widely acclaimed implicit method based on isosurface polygonization of a signed distance field, proposed by Hoppe et al. [60]. Extensions and optimizations to the algorithm have been proposed, but still manifest computational complexity of $O(n \log n)$, where n is the number of points [61].

The proposed method for reconstructing triangle mesh from LiDAR scans exploits the inherent scanline structure of the point clouds. Points beloning to scanlines are separated at specific polar angles, according to the specifications of the LiDAR sensor. (Automatic separation method based on polar angle histogram was developed, but it suffered from unreliability issues.) Subsequent scanlines are then connected by addressing point pairs in sequence determined by their azimuthal angle. One of the lines is treated as a pivot, and points from the other line are connected to the active point until it is surpassed, in which case the line designations are swapped. This results into an algorithm of linear complexity $O(n)$. The full triangulation algorithm is outlined as follows:

1. Convert each point $\boldsymbol{p} = \begin{bmatrix} x_p \\ y_p \\ z_p \end{bmatrix}$ in LiDAR point cloud $\mathcal{P}$ to spherical coordinates $\boldsymbol{p_s} = \begin{bmatrix} r_p \\ \theta_p \\ \phi_p \end{bmatrix}$ and save them into a new array $\mathcal{P}_s$.
2. Sort $\mathcal{P}_s$ according to polar angle θ_p.
3. Split the sorted $\mathcal{P}_s$ into lines $\mathcal{L}_{1\cdots N_l}$ by finding indices of points splitting the lines. Split angles are chosen according to the LiDAR sensor specification.
4. Sort lines $\mathcal{L}_{1\cdots N_l}$ according to azimuthal angle ϕ_p.

5. Connect two consecutive line points $\boldsymbol{p_{s,i,j}}, \boldsymbol{p_{s,i,j+1}} \in \mathcal{L}_i$ to form an edge and add their indices to $\mathcal{E}$ in case the difference $\phi_{p_{s,i,j+1}} - \phi_{p_{s,i,j}}$ is larger than minimum required difference $\Delta\phi_{\min}$.

6. Connect two lines $\mathcal{L}_i$ and $\mathcal{L}_{i+1}$ with Alg. 3.

Algorithm 3 Connect two spherical coordinate point lines

Inputs:
$\mathcal{L}_i$: First sorted line of points
$\mathcal{L}_{i+1}$: Second sorted line of points
$\Delta\phi_{\min}$: Minimum required azimuthal angle difference
$\mathcal{E}$: Pre-existing edges
Output:
$\mathcal{E}$: Updated edges with added connective edges

procedure ConnectLines($\mathcal{L}_i, \mathcal{L}_{i+1}, \Delta\phi_{\min}, \mathcal{E}$)
  $\mathcal{P}_{new} \leftarrow \{\}$ ▷ Stripped point cloud, initially empty
  $j \leftarrow \begin{bmatrix}1\\1\end{bmatrix}$ ▷ Indices of active points in lines
  $a \leftarrow 1$ ▷ Connecting line index
  $b \leftarrow 2$ ▷ Pivot line index
  while $j_1 < |\mathcal{L}_i| \vee j_2 < |\mathcal{L}_{i+1}|$ **do**
    $\Delta\phi \leftarrow \phi_{p_{s,i+b-1,j_b}} - \phi_{p_{s,i+a-1,j_a}}$ ▷ Difference between active points az. angles
    if $\Delta\phi < 0$ **then** ▷ Swap pivot and connecting line designations
      $c \leftarrow a$
      $a \leftarrow b$
      $b \leftarrow c$
    else
      if $|\Delta\phi| < \Delta\phi_{\min}$ **then** ▷ Create and store the edge
        $\mathcal{E} \leftarrow \mathcal{E} \cup \{\{\{i+a-1, j_a\}, \{i+b-1, j_b\}\}\}$
      end if
      $j_a \leftarrow j_a + 1$ ▷ Increase point index of the connecting line
    end if
  end while
  return $\mathcal{E}$
end procedure

7. Form triangles $\mathcal{T}$ from edges $\mathcal{E}$ by finding cycles of length 3.

8. Transform each triangle $t = \{\boldsymbol{p_{t,1}}, \boldsymbol{p_{t,2}}, \boldsymbol{p_{t,3}}\} \in \mathcal{T}$ to counterclockwise indexing w.r.t. the origin by swapping $\boldsymbol{p_{t,2}}$ and $\boldsymbol{p_{t,3}}$ in case $((\boldsymbol{p_{t,2}} - \boldsymbol{p_{t,1}}) \times (\boldsymbol{p_{t,3}} - \boldsymbol{p_{t,1}})) \cdot \boldsymbol{p_{t,1}} > 0$.

9. Filter out all triangles with normal-to-origin angle γ exceeding a chosen thresh-

old $\gamma_{\max}$. Formula for γ is presented in Eq. 55

$$\gamma = \left(\frac{(\boldsymbol{p_{t,2}} - \boldsymbol{p_{t,1}}) \times (\boldsymbol{p_{t,3}} - \boldsymbol{p_{t,1}})}{\|(\boldsymbol{p_{t,2}} - \boldsymbol{p_{t,1}}) \times (\boldsymbol{p_{t,3}} - \boldsymbol{p_{t,1}})\|} \right) \cdot \left(-\frac{\boldsymbol{p_{t,1}}}{\|\boldsymbol{p_{t,1}}\|} \right) \tag{55}$$

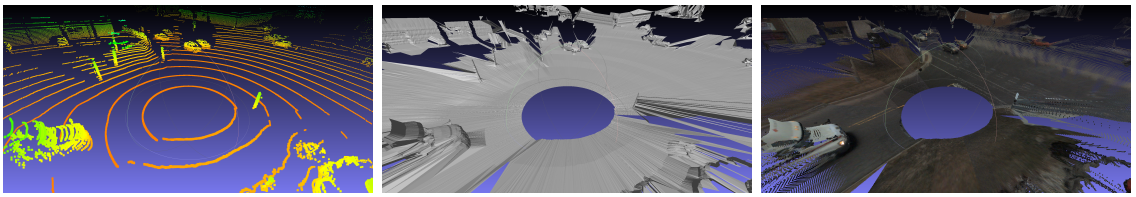
Triangle Mesh Rasterization

The triangle mesh is rasterized with OpenGL graphics *application programming interface* (API), enabling efficient formation of depth images from a triangle mesh. Rasterization is achieved with a virtual camera encompassing a frustum-shaped rendering volume. This can be achieved by forming a *projection matrix* from the camera intrinsic matrix $\boldsymbol{A}$, near and far clipping plane distances d_1, d_2 and image width and height w, h, as described by Eq. 56 [62, Chapter 2].

$$\boldsymbol{A}_{\text{OpenGL}} = \begin{bmatrix} \frac{2\boldsymbol{A_{1,1}}}{w} & \frac{-2\boldsymbol{A_{1,2}}}{w} & \frac{w-2\boldsymbol{A_{1,3}}}{w} & 0 \\ 0 & \frac{2\boldsymbol{A_{2,2}}}{h} & \frac{-h+2\boldsymbol{A_{2,3}}}{h} & 0 \\ 0 & 0 & \frac{-d_2-d_1}{d_2-d_1} & \frac{-2d_2d_1}{d_2-d_1} \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{56}$$

Depth image is created by rendering the world coordinate for each pixel. This was made possible by passing camera extrinsic parameter matrix to the vertex shader and transforming each vertex with it. In result, the rasterizer interpolator efficiently assigns the correct world position for each rendered pixel. Thus the depth image can be created by calculating the euclidean distance from each pixel's rendered world position to the camera origin.

With the rendered depth images, the camera RGB images can be projected into a 3D point cloud, which consequently can be trivially BEV-projected to create the RGB BEV frame. Fig. 19 demonstrates the results of LiDAR point cloud triangulation and image projection.



(a) Original LiDAR point cloud

(b) Triangulated mesh

(c) Point cloud projected using RGB and depth images

Figure 19: Example of point cloud triangulation and RGB image projection

3.1.2.3 Heightmap / Road Label Frames

The global BEV heightmap and road label label frames are incorporated with metadata regarding their resolution and coordinate origin. This enables cropping of the target (heightmap and road label) frames to be sampled from the global BEV images directly with an affine transformation. The affine transformation $\boldsymbol{U_i}$ for frame i

can be constructed from the frame vehicle transformation $\boldsymbol{Q_i}$ (details presented in Section 3.1.1.2), a global BEV image resolution based scaling factor s and global BEV image origin location (o_x, o_y). Derivation of $\boldsymbol{U_i}$ is presented in Eq. 57 to Eq. 61. The final frame is produced by cropping a region of desired size around origin from transformed global BEV image.

$$\boldsymbol{f_i} = \begin{bmatrix} \boldsymbol{Q_{i,1,1}} \\ \boldsymbol{Q_{i,2,1}} \end{bmatrix} \tag{57}$$

$$\hat{\boldsymbol{f}}_i = \frac{\boldsymbol{f_i}}{\|\boldsymbol{f_i}\|} \tag{58}$$

$$\boldsymbol{T}_{U,i} = \begin{bmatrix} 1 & 0 & -s\boldsymbol{Q_{1,4}} - o_x \\ 0 & 1 & s\boldsymbol{Q_{2,4}} - o_y \\ 0 & 0 & 1 \end{bmatrix} \tag{59}$$

$$\boldsymbol{R}_{U,i} = \begin{bmatrix} \boldsymbol{f_1} & -\boldsymbol{f_2} & 0 \\ \boldsymbol{f_2} & \boldsymbol{f_1} & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{60}$$

$$\boldsymbol{U_i} = (\boldsymbol{R}_{U,i}\boldsymbol{T}_{U,i})^{-1} \tag{61}$$

3.2 Neural Network Prediction Models

In this work two different neural network meta-architectures for prediction of the heightmap and road label frames are considered: a fully convolutional encoder-decoder meta-architecture and a corresponding U-Net equivalent (a similar structure with skip connections). U-Net architecture has been shown suitable for road detection from summertime satellite pictures [49], and thus is considered the primary candidate due to its relative simplicity and computational efficiency. Encoder-decoder architecture is considered as a baseline to evaluate the impact of the skip connections.

3.2.1 Modules

The network architectures presented are comprised of recurring structures of several layers referred to as *modules*. Each of the modules perform a single semantical operation and they form the basis for algorithmic construction of well-defined network architectures. This is crucial for automatic hyperparameter search, discussed in Section 4.1.

There are three modules: a convolution module, a downscale module and an upscale module. The convolution module conserves the input resolution and consists of minimum of three layers: a 2D convolution layer, ReLU nonlinearity and batch normalization [63]. The module can be defined with a dropout rate of greater than 0, in which case a dropout layer is also included. The convolution layer uses c $k \times k$ convolution kernels, with both parameters configurable. Batch normalization uses momentum term of 0.99 for moving mean and average, and ϵ of 0.001. The convolution module is presented in Fig. 20.

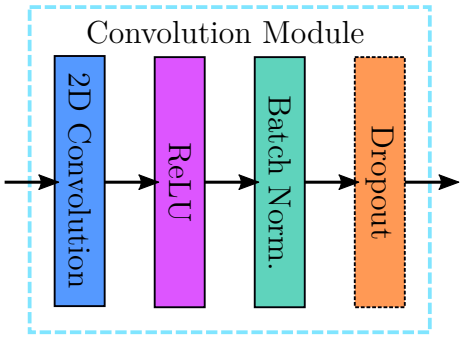
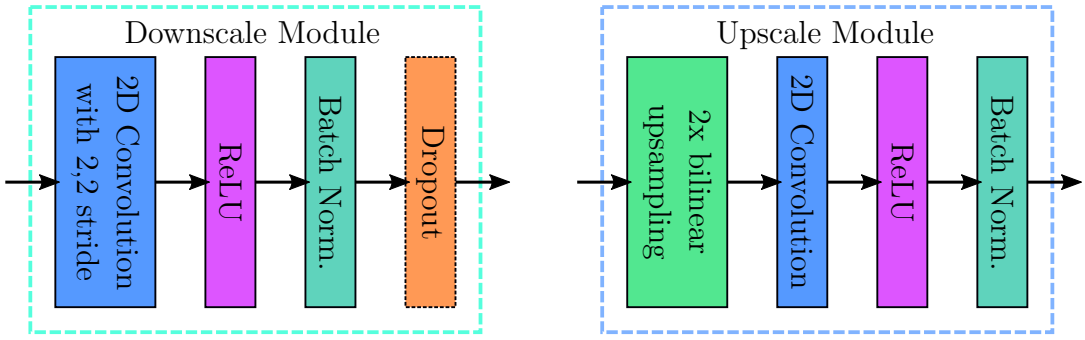


Figure 20: Convolution module

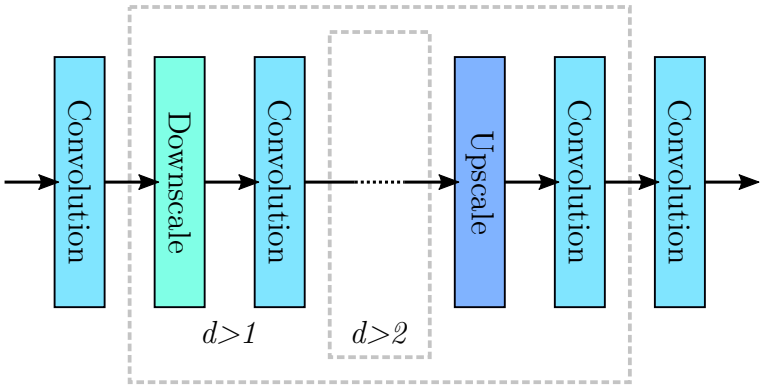
Down- and upscale modules perform the encoding and decoding tasks, respectively. This is achieved by reducing and increasing the spatial resolution of the network while altering the number of channels in a manner prominent in most convolutional architectures. Instead of pooling however, the downscaling is performed with strided convolutions. This enables the network to learn the most useful downscaled representation of the data (in contrast to the fixed, non-parametric functionality of pooling layers typically used for the task). Thus the 2×2 striding is the only differing factor between convolution and downscale modules. Upscaling is done by adding a bilinear upsampling layer with magnification ratio of 2×2 in front of a 1×1 convolution. Such approach was adopted due to the training instability of deconvolutional (convolution transpose) layers, which can cause grid-like artifacts in the resulting image [48]. Upscale modules do not use dropout layers, as they are not proved to be beneficial in the decoding stage [64, 65]. Instead, upscale modules are always followed by a convolution module, enabling information flow over a larger area than is enabled only by the upsampling and 1×1 convolutional layer. Down- and upscale modules are presented in Fig. 21.



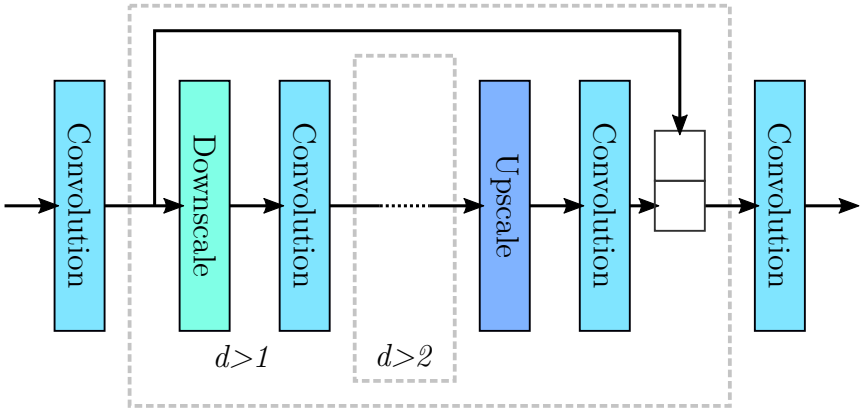
(a) Downscale module (b) Upscale module

Figure 21: Down- and upscale modules

3.2.2 Network Architectures



(a) Encoder-decoder network architecture



(b) U-Net network architecture, with skip connection and concatenation visualized

Figure 22: Network architectures, comprised of modules discussed in 3.2.1

Both network meta-architectures are defined as a recursive, nested structures with respect to depth hyperparameter d, as depicted in 22. Maximum depth is constrained by the fact that tensor with dimension subject to scaling of size 1 cannot be further downscaled. With the 2×2 down/upscaling ratio used in all of the networks this can be formulated as $d_{\max} = \lfloor \log_2(\min(n, m)) \rfloor + 1$ for an input tensor $\boldsymbol{I} \in \mathbb{R}^{n \times m \times 6f}$.

Each of the convolution and downscale modules use convolutional kernel of size $k \times k$, where k is another architecture-defining hyperparameter. After each downscale layer number of channels is doubled. For all encoder phase modules, dropout rate of $\sigma_l = \sigma - \frac{l-1}{10}$ is used, where σ is the base dropout hyperparameter and l denotes depth of the recursion level.

Inputs

The BEV LiDAR and RGB frames are concatenated into a 6-channel composite frame to form the network input tensor $\boldsymbol{I}$. As a single frame might not contain all the necessary information for the network to be able to produce accurate predictions, it is concatenated with a number of frames from previous time steps. This operation

forms an input tensor with dimensions of $n \times m \times 6f$, where f is a hyperparameter denoting the number of total frames included. The input concatenation process is illustrated in Fig. 23

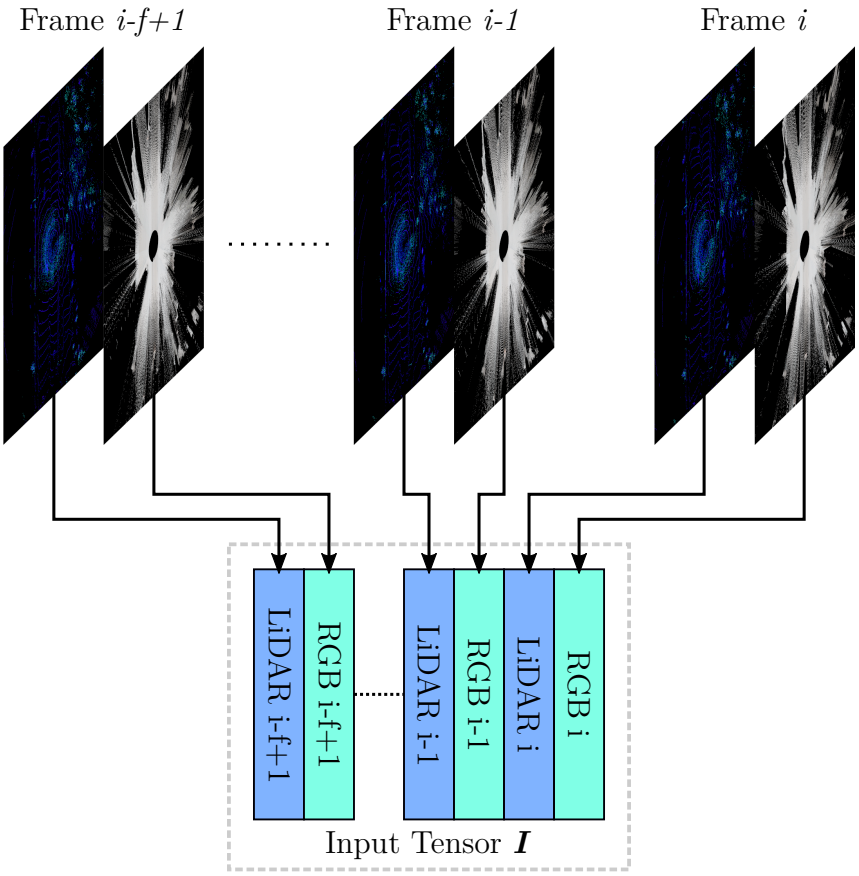


Figure 23: Input tensor concatenation

3.2.3 Output Branching

Each model produces two single-channel output frames with first two dimensions equal to input ($n \times m \times 1$): a heightmap and a road label. The road label frame is produced by passing it through a sigmoid activation function to compress the output onto $(0, 1)$ range. 1 signifies full confidence for the cell being road and 0 signifies full confidence that it is not road. No activation is used for the heightmap, since it is required to be allowed to reach values on the entire domain of real numbers.

However, if both outputs are produced directly from the same penultimate layer, the heightmap and road label modalities might become prohibitively tightly correlated. In such case, the last layer will not be capable of changing a value in one domain without affecting the other. This is illustrated in Fig. 24. (Note that low heightmap values correlate with high road label values.)

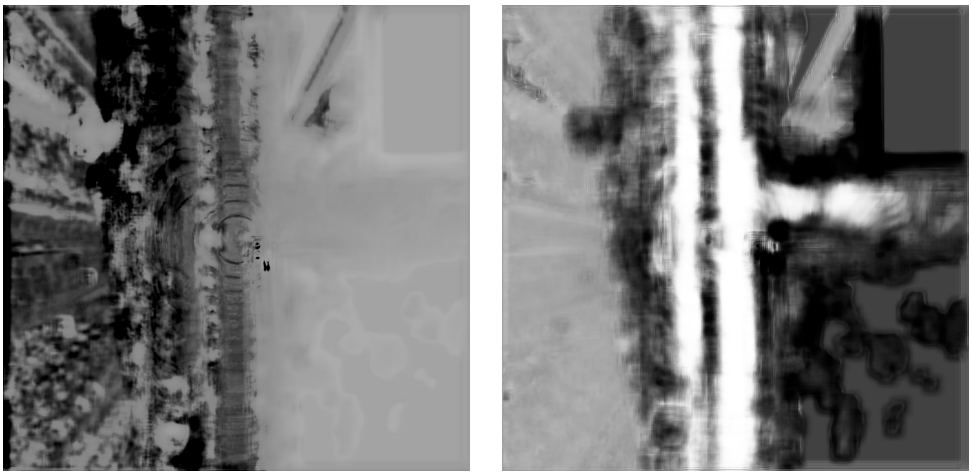


Figure 24: Strong correlation between the predicted heightmap (left) and road label (right)

To prevent this from happening, the encoding phase of the network is divided into two branches from depth b onwards; such approach enables the network to learn distinctive features relevant to each modality already in the more information-dense lower layers. To find the optimal branching depth, it is presented as the final architecture hyperparameter and thus incorporated into the hyperparameter search.

4 Experiments

The objective for the dataset conversion experiment was to evaluate the reliability and quality of the dataset tool. The objective for the hyperparameter search was to find the most suitable specific architecture for the road detection task.

Hardware and Environment Setup

All the experiments were run on a workstation PC equipped with a 24-core AMD Ryzen Threadripper 3960X, 64GB of system memory and a NVIDIA Titan RTX GPU with 24GB of memory. The system uses Ubuntu Linux 18.04 LTS operating system.

For the prediction model implementation, Tensorflow 2.4.0 library was used, as it enables GPU acceleration via CUDA API. For CUDA, version 11.0 was used. Python 3.6.9 was used for the neural network implementation.

Dataset Conversion

In this work, Canadian Adverse Driving Conditions (CADC) dataset [66] is used as input for the dataset tool. It provides *Global Navigation Satellite System + Inertial Navigation System* (GNSS+INS) measurements, 32-line LiDAR scans and 360°, 8-camera RGB imaging with 10Hz sample rate. The dataset is gathered in urban and suburban environments in Waterloo, Canada. It features difficult environmental conditions such as precipitation in form of water and snow and obstructed street markings and signs due to snow. The dataset consists of driving sequences with running time of approximately 1 minute.

All 75 raw data driving sequences in the CADC dataset were processed with the dataset tool first phase to produce the global BEV heightmap and RGB images. 18 of the sequences were labeled, of which 2 were excluded to form the evaluation set. The labeled sequences were selected with focus on quality (minimal distortions and discontinuities) and variety to provide the model the best possible opportunity for generalization. Labeling was done on *GNU image manipulation program* [2] (GIMP) with assistance from satellite images in areas of ambiguity and limited information.

Labeled sequences were processed by the dataset tool second phase to form BEV frames of size 1024×1024 corresponding to real world area of $102.4\,\mathrm{m} \times 102.4\,\mathrm{m}$. The vehicle origin was centered in each frame. In addition to forming the input-output frame pairs, power-of-two resolution reductions were produced down to resolution of 64×64. Reductions were computed as simple averages over the 2×2 pixel windows in the respective higher resolution frame. Pixels with no information (0 points after point cloud projection) were discarded.

[2] https://www.gimp.org/

4.1 Hyperparameter Search

The hyperparameter search was performed as two consecutive grid searches over the hyperparameter domain. Due to the combinatorical nature of the grid search, the search domain grows exponentially for each additional hyperpameter and linearly for each additional hyperparameter value. Therefore, to enable the search to be finished in reasonable timeframe, the first search was performed on smaller frame resolution of 256×256. The parameter domain reduction on the second, full resolution search was designed according to results obtained from the first search.

Batch sizes for both searches were set to maximum possible allowed by the hardware memory constraints. The second search was performed only on the U-Net architecture. Search domains for the first and the second grid searches are presented in Tab. 1 and Tab. 2, respectively.

Architectures		Encoder-Decoder, U-Net
Resolution	$n \times m$	256×256
Batch size	s	16
Number of input frames	f	2, 3, 4
Kernel size	k	1, 3, 5
Model depth	d	3, 4, 5
Dropout level	σ	0.6, 0.7
Output branching depth	b	0, 1, 2

Table 1: First hyperparameter grid search domain

Architectures		U-Net
Resolution	$n \times m$	1024×1024
Batch size	s	2
Number of input frames	f	2, 3, 4
Kernel size	k	5
Model depth	d	3, 4, 5
Dropout level	σ	0.6
Output branching depth	b	0, 1, 2

Table 2: Second hyperparameter grid search domain

4.2 Loss Functions and Accuracy Metric

Loss functions (sometimes called *error functions*) are used for model training - minimizing loss translates to minimizing difference between model output and the desired output in supervided learning training configuration. Accuracy metrics are functions used to evaluate model performance.

For road label frames, *binary cross-entropy* was used as the loss function and threshold classification as the accuracy metric. Binary cross-entropy is a widely used loss function in binary classification tasks, as it measures difference between two

probability distributions. As ultimately the decision on whether a pixel represents a patch of road or not is to be done, the threshold classification with threshold of 0.5 provides a reasonable accuracy metric.

The functions are presented in Eq. 62 and Eq. 63, respectively.

For heightmap frames, selective *L1 loss* (also known as *least absolute deviations*) was used as the loss function and modified masked negative L1 loss as the accuracy function. Masking in this context means omitting error for all pixels with point density 0 (density is denoted by $\rho(p)$ for pixel p) in desired output. The functions are presented in Eq. 66 and Eq. 67, respectively. $\boldsymbol{Y_p}$ and $\boldsymbol{Y_t}$ denote the predicted and ground truth frames.

$$L_{\text{BCE}}(\boldsymbol{Y_p}, \boldsymbol{Y_t}) = -\frac{1}{N}\sum_{i=1}^{N} y_{t,i}\log(y_{p,i}) + (1-y_{t,i})\log(1-y_{p,i}) \tag{62}$$

$$A_{\text{TC}}(\boldsymbol{Y_p}, \boldsymbol{Y_t}) = \frac{1}{N}\sum_{i=1}^{N} 1 - |\lfloor y_{p,i} \rceil - y_{t,i}| \tag{63}$$

$$a(y_t, y_p) = \begin{cases} 0 & \text{if } \rho(y_t) = 0 \\ |y_t - y_p| & \text{otherwise} \end{cases} \tag{64}$$

$$a^{-1}(y_t, y_p) = \begin{cases} 0 & \text{if } \rho(y_t) = 0 \\ 1 - |y_t - y_p| & \text{otherwise} \end{cases} \tag{65}$$

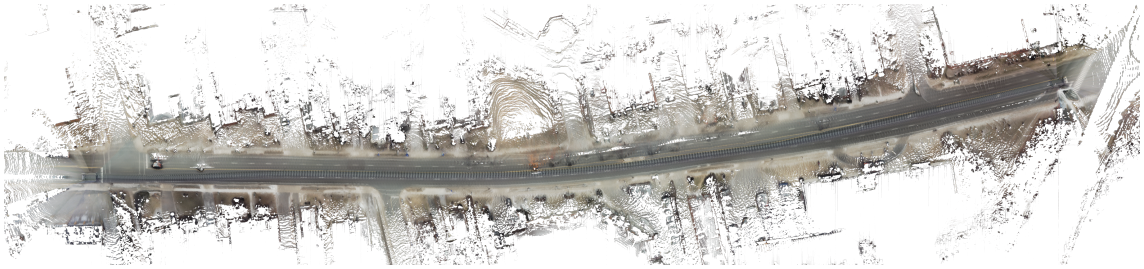
$$L_{\text{L1masked}}(\boldsymbol{Y_p}, \boldsymbol{Y_t}) = \frac{1}{N}\sum_{i=1}^{N} a(y_{t,i}, y_{p,i}) \tag{66}$$

$$A_{\text{L1masked}}(\boldsymbol{Y_p}, \boldsymbol{Y_t}) = \frac{1}{N}\sum_{i=1}^{N} a^{-1}(y_{t,i}, y_{p,i}) \tag{67}$$

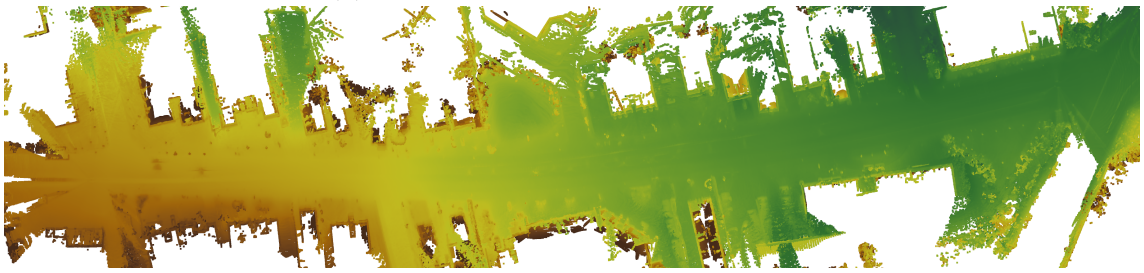
5 Results

5.1 Dataset Tool

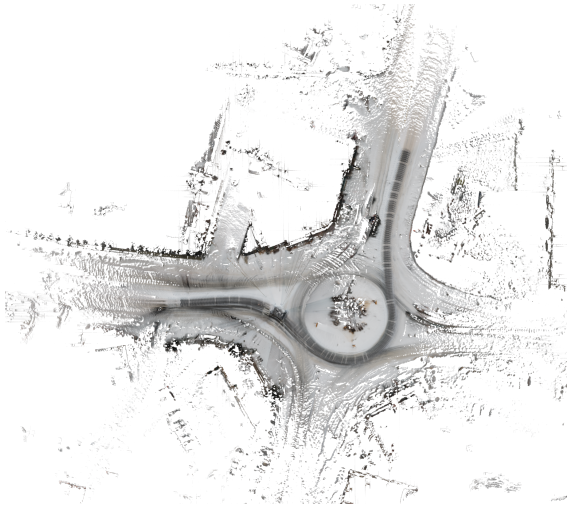
5.1.1 First Phase



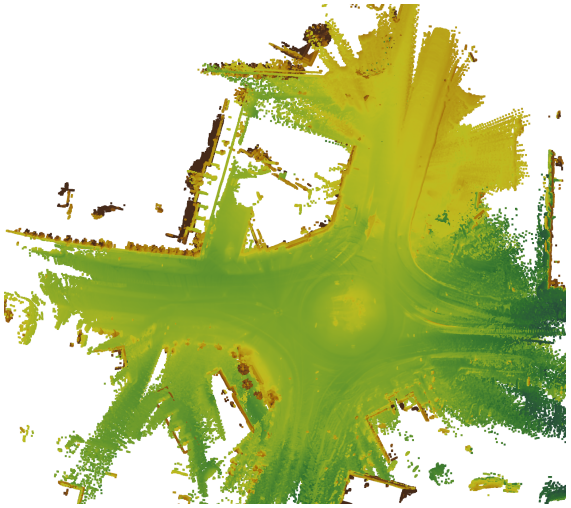
(a) Road markings visible, BEV RGB



(b) Road markings visible, BEV heightmap



(c) Road markings not visible, BEV RGB



(d) Road markings not visible, BEV heightmap

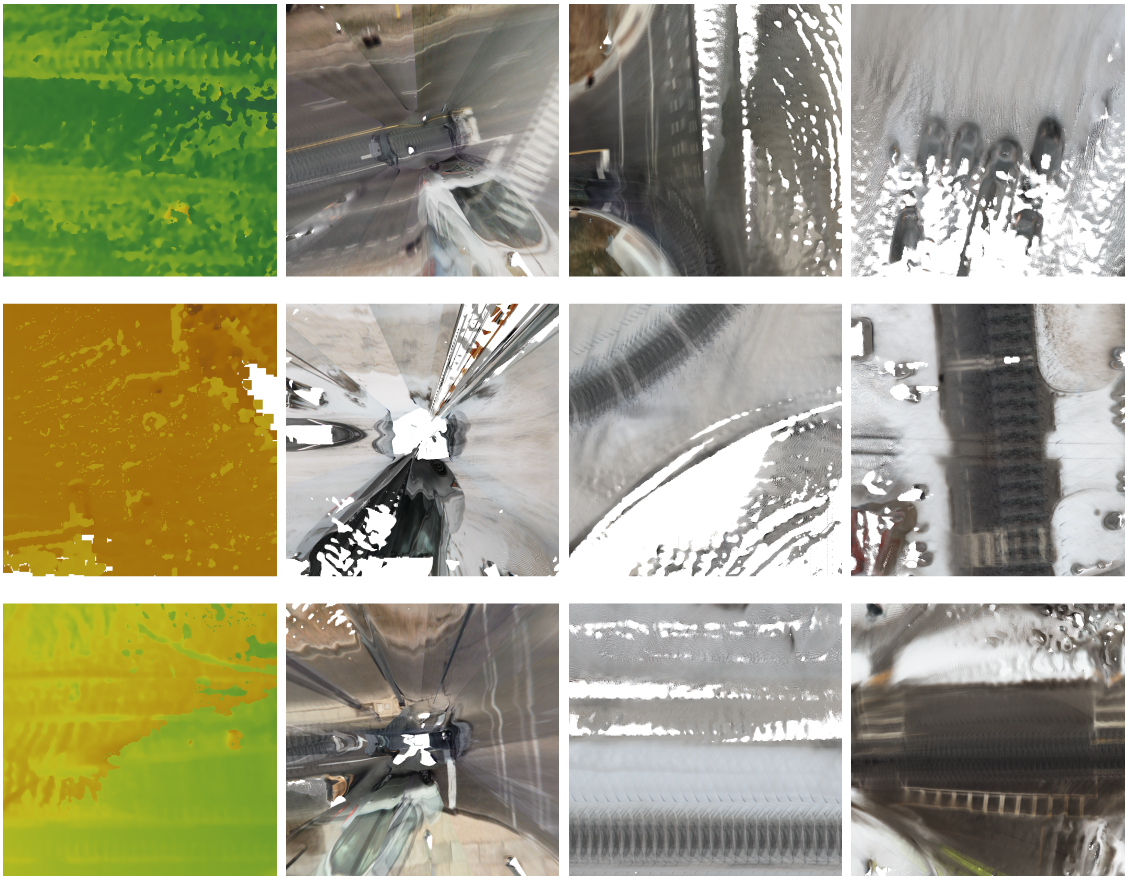
Figure 25: Global BEV RGB and heightmap images produced by the first phase

The first phase of the dataset tool produced mainly good results: all sequences were successfully converted into global BEV RGB and heightmap images without severe distortions or inconsistencies. The global BEV images were manually evaluated by a human inspector. Each sequence was processed in approximately 30 minutes,

depending on the length of the sequence (in frames). Examples of produced global BEV images with and without the road markings visible are presented in the figure 25.

However, closer inspection revealed (relatively minor) issues caused mostly by the chosen subalgorithms. Partly due to these issues it was challenging or impossible to label some of the driving sequences using the global BEV RGB image alone. Instead, a satellite image of the area around each driving sequence was aligned with the global BEV RGB image and used as an aid for differentiating the drivable area. Even if the issues were to be addressed, it is unclear whether it would be possible to produce labels using only global BEV RGB images featuring winter conditions with a snow cover.

Issues



(a) Heightmap inconsistency (b) Stationary vehicle (c) Other vehicles (d) Blurriness, misalignment

Figure 26: Issues encountered in the dataset tool first phase processing

Issues encountered in the dataset tool first phase are illustrated in the Fig. 26. Most notable defects are noise/tearing in the heightmap, artifacts caused by

prolonged immobility (such as in an intersection), artifacts caused by other vehicles and blurred/misaligned features.

Heightmap inconsistencies are caused by failed point cloud alignment, causing formation of two or more separate ground planes. This leads to confusion in the KDE ground level estimation and subsequent generation spots and hard edges in the heightmap. Distortions from the vehicle not moving are most likely caused by accumulation of radial artifacts formed in the point cloud spherical augmentation process. Other vehicles have significant amount of vertical surfaces on which plenty of LiDAR points land upon. This leads to those points dominating the distribution of a column, raising the ground level estimate and causing formation of wall-like structures in the final heightmap. In the RGB image, this can be noticed as occluded areas behind passing vehicles. Finally, misaligned or blurred features in the RGB image can be results of two factors: camera calibration errors and/or point cloud misalignment — in most cases the former.

5.1.2 Second Phase

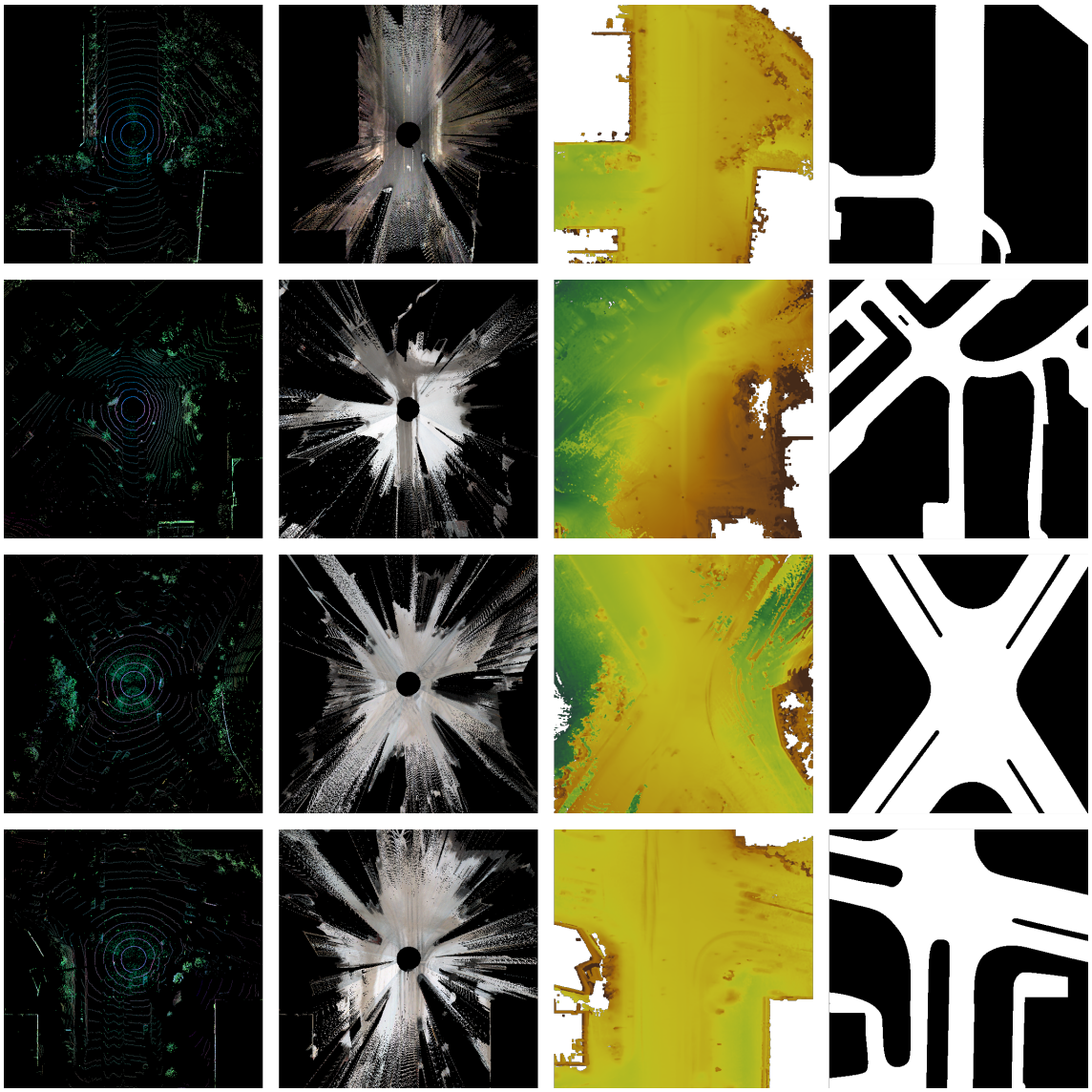


Figure 27: Dataset tool second phase output (BEV frames). From left to right: LiDAR, RGB, heightmap, road label

LiDAR and RGB frame formation was performed in total average of ~20 ms. Excluding minor discontinuities caused primarily by calibration errors, the RGB frame projection produces reliable results with immediate surroundings clearly visible. After a distance of ~10 m, "holes" start to emerge in the RGB frame due to insufficient resolution of the camera images. Dataset entries after second phase processing are presented in Fig. 27.

5.2 Prediction Models

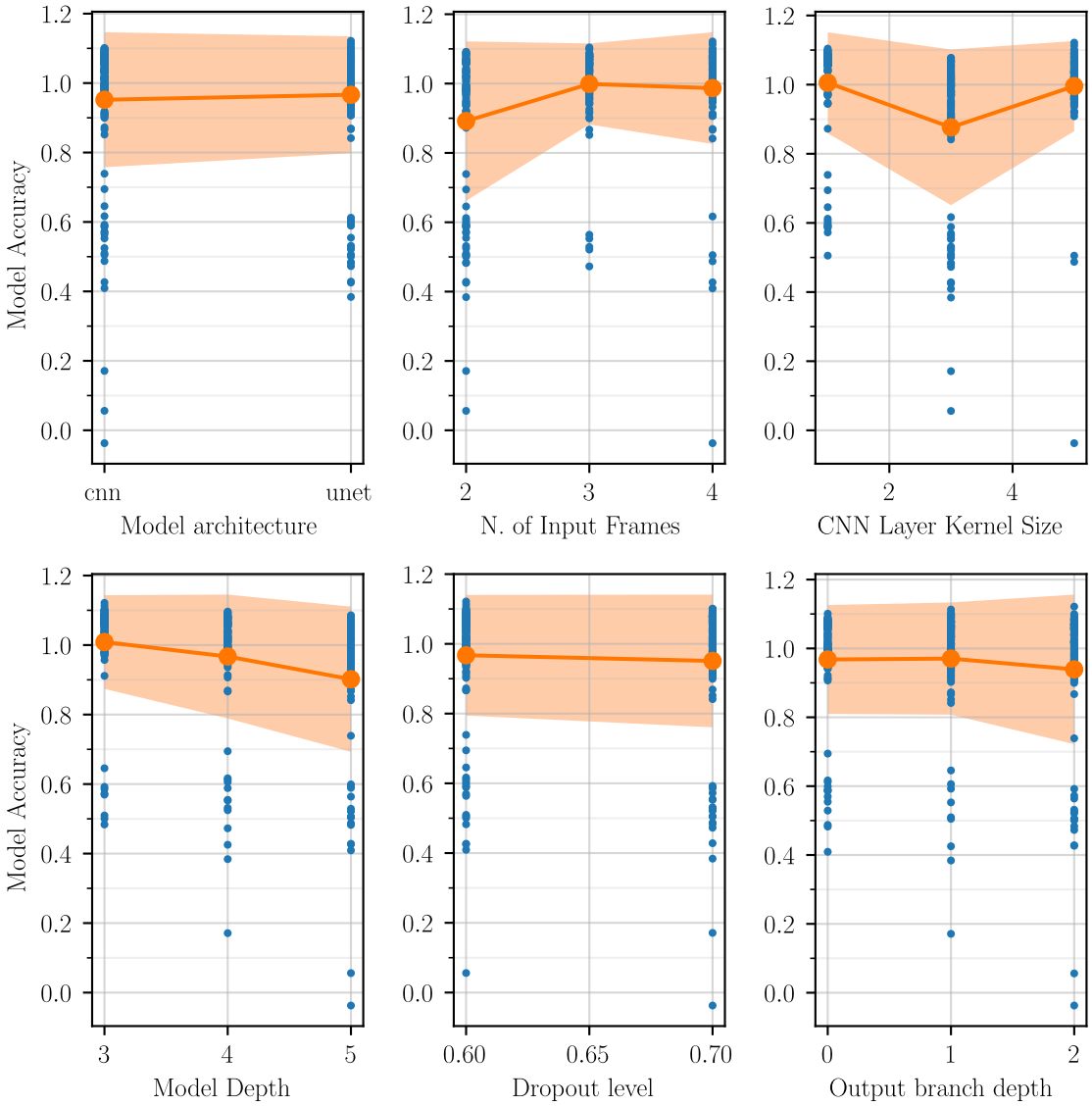


Figure 28: Results of the first hyperparameter search – each blue dot represents an unique model

Results of the first hyperparameter search, performed on resolution of 256×256, are presented in the Fig. 28. Model distribution is presented with respect to each hyperparameter. Noteworthy observations include nonlinear average performance impact of number of input frames f and convolution kernel size k, and negative performance impact of increased model depth d.

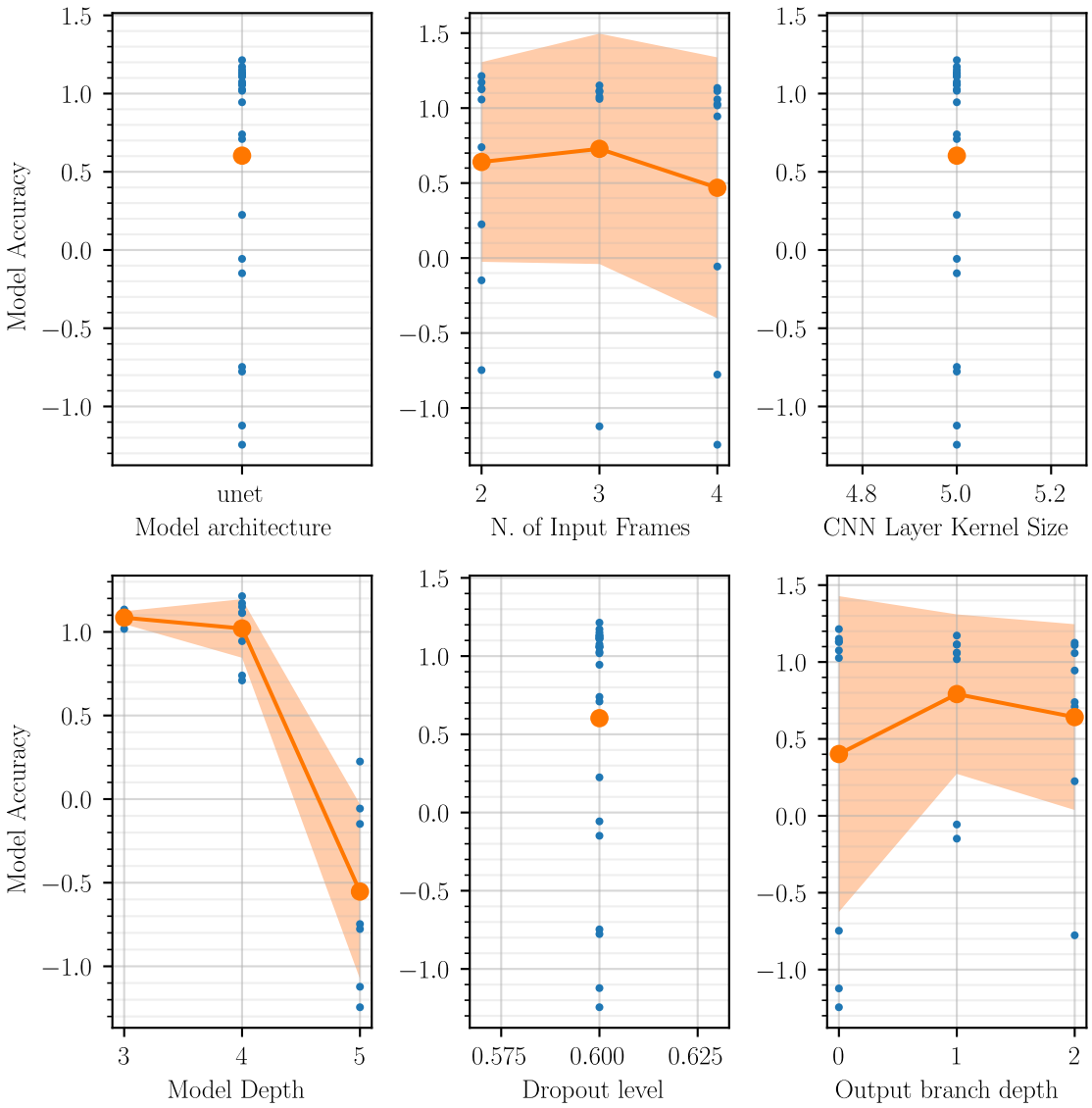
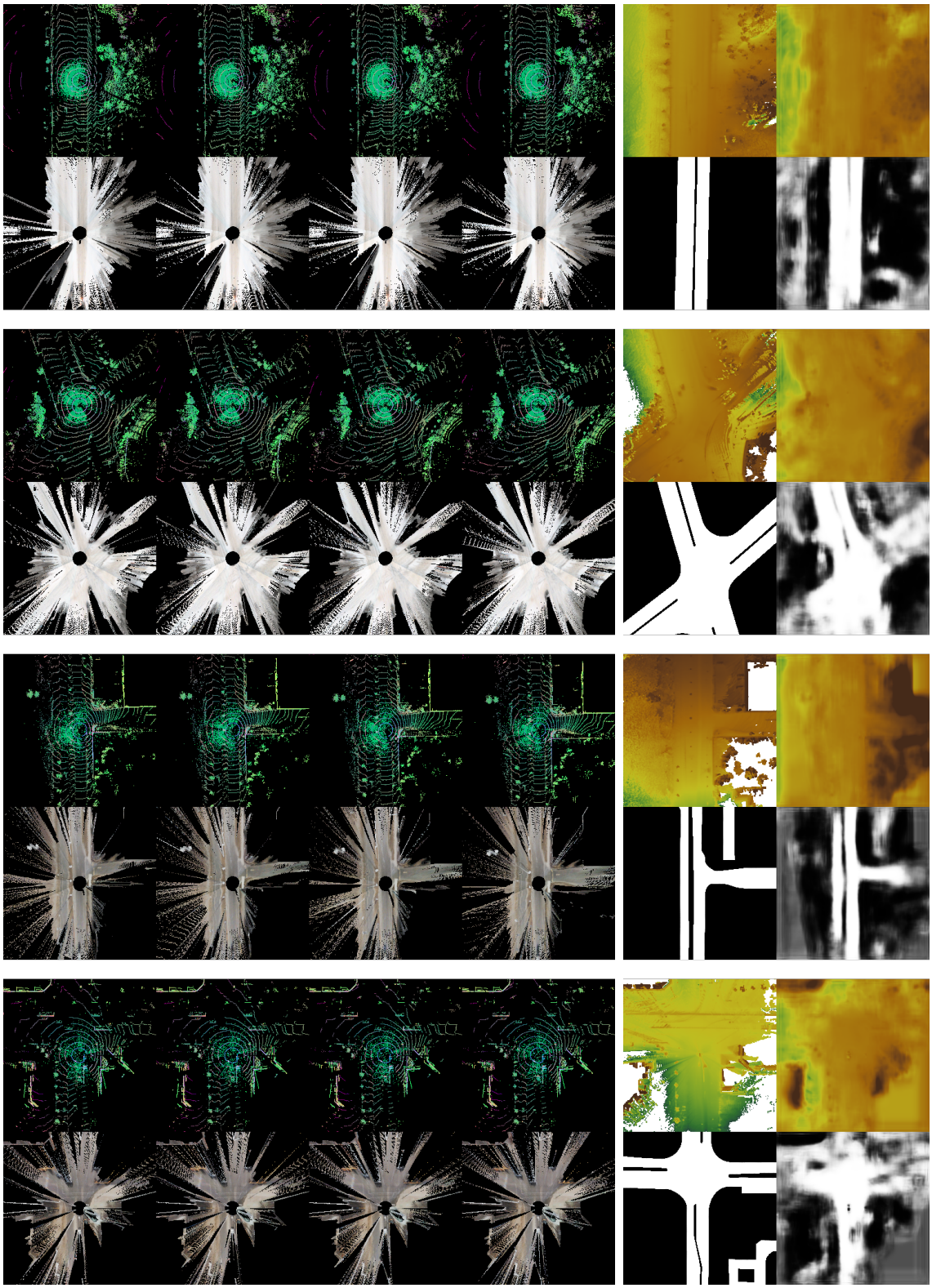


Figure 29: Results of the second hyperparameter search – each blue dot represents an unique model

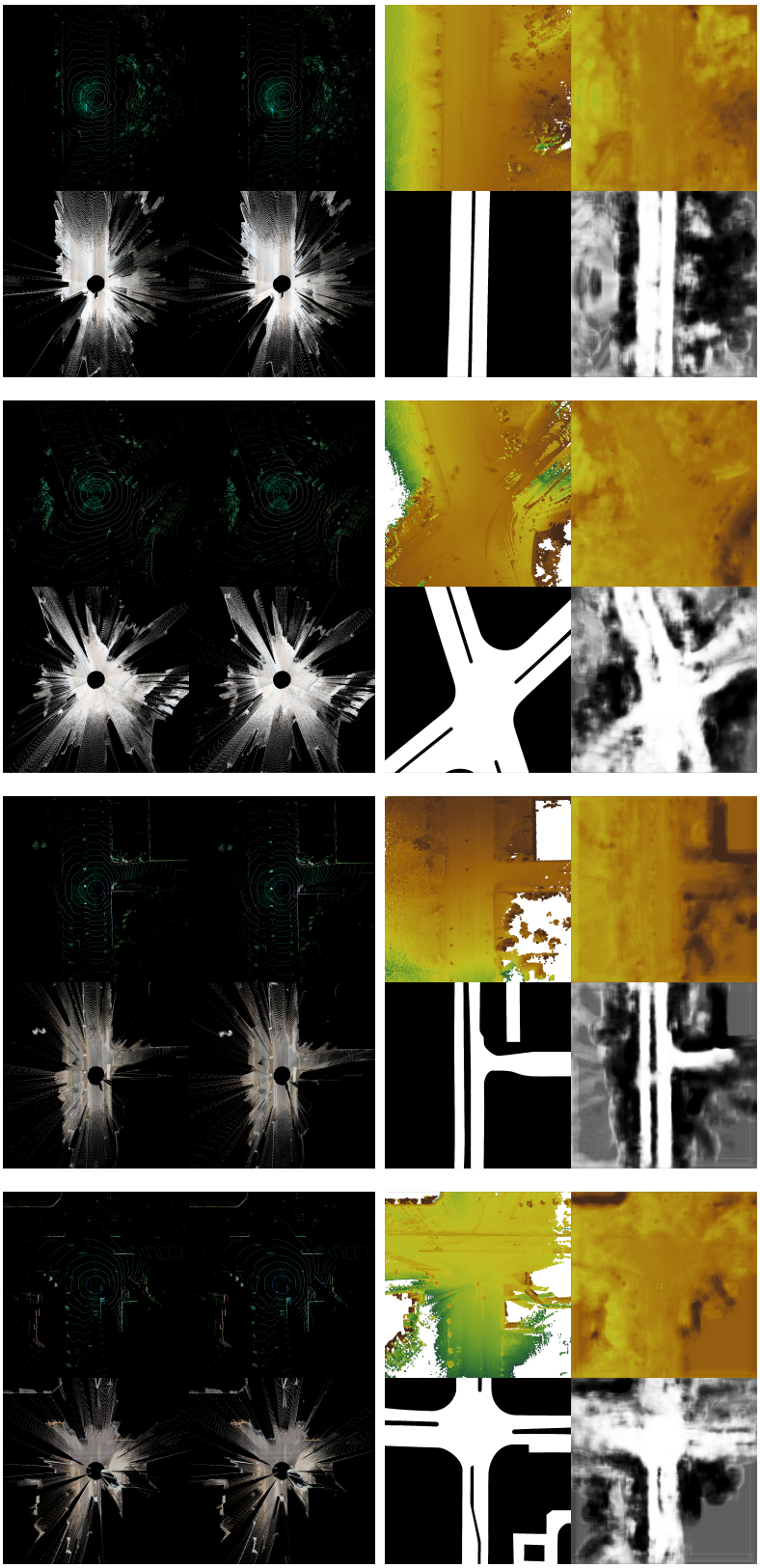
The second, full 1024×1024 resolution search shows results very similar to the first search, only more pronounced. Especially interesting is the steep decline in performance on model depth $d > 4$. It is also noteworthy that the second search results most likely exhibit large variance due to small batch size of 2. Results of the second hyperparameter search are presented in Fig. 29.



(a) Inputs

(b) Outputs: target (left), prediction (right)

Figure 30: Best model of the first hyperparameter search



(a) Inputs

(b) Outputs: target (left), prediction (right)

Figure 31: Best model of the second hyperparameter search

Results for best model of the first hyperparameter search are presented in Fig. 30. Gray areas in road label predictions signify uncertainty about the proper label. Overall the model predicts generic shape for the drivable area relatively reliably. Lack of snow cover appears to also impact the performance positively. Most prominent errors are shown in the areas with little or no data; on these areas the model appears to either predict no label (gray) or noisy labels. Heightmap predictions appear to mostly correlate with road detection and global features such as hills are not evident.

The best model of the second hyperparameter search exhibits very similar issues to the best model of the first parameter search. Overall, the predictions can be described to be sharper and noisier. Results for the best model of the second hyperparameter search are presented in Fig. 31.

Overall the results showed positive outcomes for most of the experiments; the dataset tool was able to produce an useful dataset for ML model training and evaluation. In the hyperparameter search, interesting trends especially regarding the kernel size and model depth were observed. These can partly be caused by the convergence test, which stopped the training if the lowpass filtered loss decrease rate dropped under a predefined threshold. Furthermore, some of the trends present in the first search were profoundly pronounced and thus introduce the issue of generalization and stability when the resolution is increased. The most accurate models demonstrate the capability of detecting general shape of the surrounding drivable area but inability to detect exact shape and boundary of it. Plenty of other architectures and module designs that may perform better have been developed over the last years. These experiments can not address whether such approaches would provide improved perfomance; such comparations would be subject to further research.

6 Conclusions

In this thesis the feasibility of drivable area (road) detection using machine learning methods and BEV-projected data is demonstrated and evaluated. Furthermore, an approach for semi-automatic dataset production is described and implemented, improving the labeling labour efficiency by a factor of several thousands.

For the dataset tool, multiple novel algorithms for manipulation of point cloud, heightmap and kernel density data were developed and implemented. Additonally, established methods were utilized to implement a tool for dataset production that

1. uses LiDAR, GNSS+INS and RGB camera data to produce global BEV heightmap and RGB images,

2. demonstates and provides a reference implementation for the real-time production of BEV LiDAR and RGB input frames and

3. produces BEV heightmap and road label frames with the vehicle centered and forward-oriented by cropping and rotating the global images.

Relatively minor defects were observed in the global BEV images, including heightmap inconsistency, blurriness, misalignment and artifacts caused by immobility or other vehicles. Most of these defects can likely be rectified by fine-tuning or replacing some of the subalgorithms.

For the prediction models, two similar, well established meta-architectures (Encoder-Decoder and U-Net) based on convolutional neural networks were evaluated. Optimal specific architecture was found by performing two subsequent hyperparameter searches on different resolutions. Best models were able to detect generic shape of drivable area on previously unseen environments. Predictions feature high levels of noise and uncertainty, particularly on areas with little input information. However, applying modifications and techiques including those presented in Section 2 will most likely yield positive impact on the predictions.

6.1 Future Work

Since the BEV heightmap is used in the model training process, it is imperative that issues related to it are addressed. Inconsistency caused by point cloud misaligment can most likely be rectified with a more sophisticated alignment method, such as the *Normal Distributions Transform* (NDT) proposed by a Saarinen et al. [67]. Artifacts caused by vehicle immobility or other vehicles can most likely be addressed with more careful point cloud preprocessing; instead of using spherical augmentation and stochastic density adjustment introduced in Section 3.1.1.1, the triangle mesh reconstruction (section 3.1.2.2) could be sampled to produce point clouds with desired charasteristics. Such charasteristics would include uniform density and reduction of vertical point clusters inside a column. RGB image projection misalignment is caused by calibration errors and therefore is difficult to address algorithmically. However,

since the global RGB image is used only as a visual aid for the labeler, correcting these issues are less critical for the performance of the ML models.

There are multiple potential routes for improving the performance of the machine learning models. A significant issue in full resolution training was the batch size of two, which yields only very poor estimates of data distributions. Due to the fully convolutional nature of the Encoder-Decoder and U-Net architectures, it would be trivial to use cropped frames for training, enabling reallocation of memory requirements from spatial resolution to batch size. Further potential improvements related to inputs and outputs include aligment of the previous frames with the most recent one, positional encoding for each pixel and decoupling the prediction uncertainty from the prediction itself.

Architecture of the ML models could be developed to numerous directions. For example, simultaneous usage of dropout and batch normalization layers is usually considered redundant, as batch normalization already provides regularization. Number of convolutional layers per module could be altered and ResNet-style [8] skip connections introduced. Furthermore, the detector efficiency could be improved with compression schemes such as the one introduced in PointPillars [16].

6.2 Autonomous Driving and Artificial Intelligence

Vehicles capable of fully autonomous driving have tremendous socioeconomic impact potential. In addition to the safety and driver expenditude considerations, a hypothetical future city with appropriate infrastructure could facilitate transportation based on autonomous vehicles with efficiency vastly exceeding that of cities of the present day. Such city would not require traffic signals, large parking lots and wide roads to facilitate large amount of vehicles driven by humans. With a centralized traffic management system, even spontaneusly forming traffic jams could be avoided [68].

Given such disruptive potential of transportation, cargo and city planning, advent of autonomous driving would seem inevitable. Replacing the human driver has so far proven to be practically impossible, but advancements in sensor and information technology are providing tools for bridging the gap at accelerating pace. Current sensors exceed human drivers in enviroment information gathering in fidelity, modality and quantity. With the datasets and computing power currently available, it is possible to develop machine learning models performing on superhuman level in narrow tasks [69].

However, models created using machine learning are extremely nonlinear systems with large number of parameters. As such, they remain difficult (practically impossible) to analyze and test thoroughly. These are highly undesirable traits for a system responsible of human lives. Therefore additional tools (and most likely theory) are required for such methods to be introduced in autonomous driving systems with confidence for safety. Nevertheless, given the industry-wide effort of bringing autonomous driving to market, development of such tooling is of a great interest and will most likely be accelerated in the near future.

References

[1] World Health Organization, "Global status report on road safety 2018," 2018, ISBN: 978-92-4-156568-4.

[2] P. Davidson and A. Spinoulas, "Autonomous vehicles: what could this mean for the future of transport," in *Australian Institute of Traffic Planning and Management (AITPM) National Conference, Brisbane, Queensland*, 2015.

[3] R. M. Kumar and K. Sreekumar, "A survey on image feature descriptors," *International Journal of Computer Science and Information Technologies (IJCSIT)*, vol. 5, pp. 7668–7673, 2014.

[4] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013. [Online]. Available: https://doi.org/10.1177/0278364913491297

[5] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3d object detection network for autonomous driving," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[6] T. Roddick, A. Kendall, and R. Cipolla, "Orthographic feature transform for monocular 3d object detection," *British Machine Vision Conference (BMVC)*, 2019.

[7] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun, "Deep learning for 3d point clouds: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, pp. 1–1, 2020.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[9] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention (MICCAI) 2015*, N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, Eds. Cham: Springer International Publishing, 2015, pp. 234–241.

[10] B. Yang, W. Luo, and R. Urtasun, "Pixor: Real-time 3d object detection from point clouds," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[11] B. Yang, M. Liang, and R. Urtasun, "Hdnet: Exploiting hd maps for 3d object detection," in *Proceedings of The 2nd Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, A. Billard, A. Dragan, J. Peters, and J. Morimoto, Eds., vol. 87. PMLR, 29–31 Oct 2018, pp. 146–155. [Online]. Available: http://proceedings.mlr.press/v87/yang18b.html

[12] J. Beltrán, C. Guindel, F. M. Moreno, D. Cruzado, F. García, and A. De La Escalera, "Birdnet: A 3d object detection framework from lidar information," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, Nov 2018, pp. 3517–3523.

[13] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28. Curran Associates, Inc., 2015. [Online]. Available: https://proceedings.neurips.cc/paper/2015/file/14bfa6bb14875e45bba028a21ed38046-Paper.pdf

[14] W. Luo, B. Yang, and R. Urtasun, "Fast and furious: Real time end-to-end 3d detection, tracking and motion forecasting with a single convolutional net," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[15] Y. Yan, Y. Mao, and B. Li, "Second: Sparsely embedded convolutional detection," *Sensors (Basel, Switzerland)*, vol. 18, no. 10, 2018. [Online]. Available: https://www.mdpi.com/1424-8220/18/10/3337

[16] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "Pointpillars: Fast encoders for object detection from point clouds," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[17] W. Shi and R. Rajkumar, "Point-gnn: Graph neural network for 3d object detection in a point cloud," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[18] S. Vora, A. H. Lang, B. Helou, and O. Beijbom, "Pointpainting: Sequential fusion for 3d object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[19] E. Khatab, A. Onsy, M. Varley, and A. Abouelfarag, "Vulnerable objects detection for autonomous driving: A review," *Integration*, vol. 78, pp. 36–48, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167926021000055

[20] W. Elmenreich, "An introduction to sensor fusion," *Vienna University of Technology, Austria*, vol. 502, pp. 1–28, 2002.

[21] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *Journal of Basic Engineering*, vol. 82, no. 1, pp. 35–45, 03 1960. [Online]. Available: https://doi.org/10.1115/1.3662552

[22] J. Humpherys, P. Redd, and J. West, "A fresh look at the kalman filter," *SIAM Review*, vol. 54, no. 4, pp. 801–823, 2012. [Online]. Available: https://doi.org/10.1137/100799666

[23] S. J. Julier and J. K. Uhlmann, "Unscented filtering and nonlinear estimation," *Proceedings of the IEEE*, vol. 92, no. 3, pp. 401–422, 2004.

[24] S. Särkkä, *Bayesian filtering and smoothing*. Cambridge University Press, 2013, no. 3, ISBN: 9781139344203.

[25] M. L. Fung, M. Z. Q. Chen, and Y. H. Chen, "Sensor fusion: A review of methods and applications," in *2017 29th Chinese Control And Decision Conference (CCDC)*, 2017, pp. 3853–3860.

[26] D. Crisan and A. Doucet, "A survey of convergence results on particle filtering methods for practitioners," *IEEE Transactions on Signal Processing*, vol. 50, no. 3, pp. 736–746, 2002.

[27] F. Castanedo, "A review of data fusion techniques," *The Scientific World Journal*, vol. 2013, Oct 2013. [Online]. Available: https://doi.org/10.1155/2013/704504

[28] L. Caltagirone, M. Bellone, L. Svensson, and M. Wahde, "Lidar–camera fusion for road detection using fully convolutional neural networks," *Robotics and Autonomous Systems*, vol. 111, pp. 125 – 131, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0921889018300496

[29] J. Wang, Z. Wei, T. Zhang, and W. Zeng, "Deeply-fused nets," *Computing Research Repository (CoRR)*, vol. abs/1605.07716, 2016. [Online]. Available: http://arxiv.org/abs/1605.07716

[30] G. Larsson, M. Maire, and G. Shakhnarovich, "Fractalnet: Ultra-deep neural networks without residuals," *arXiv preprint arXiv:1605.07648*, vol. abs/1605.07648, 2016. [Online]. Available: http://arxiv.org/abs/1605.07648

[31] J. Ku, M. Mozifian, J. Lee, A. Harakeh, and S. L. Waslander, "Joint 3d proposal generation and object detection from view aggregation," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 1–8.

[32] M. Liang, B. Yang, S. Wang, and R. Urtasun, "Deep continuous fusion for multi-sensor 3d object detection," in *Computer Vision (ECCV) 2018*, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds. Cham: Springer International Publishing, 2018, pp. 663–678.

[33] Z. Wang, W. Zhan, and M. Tomizuka, "Fusing bird's eye view lidar point cloud and front view camera image for 3d object detection," in *2018 IEEE Intelligent Vehicles Symposium (IV)*, 2018, pp. 1–6.

[34] Y. Wang, W.-L. Chao, D. Garg, B. Hariharan, M. Campbell, and K. Q. Weinberger, "Pseudo-lidar from visual depth estimation: Bridging the gap in 3d object detection for autonomous driving," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[35] X. Weng and K. Kitani, "Monocular 3d object detection with pseudo-lidar point cloud," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV) Workshops*, Oct 2019.

[36] H. Fu, M. Gong, C. Wang, K. Batmanghelich, and D. Tao, "Deep ordinal regression network for monocular depth estimation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[37] Y. You, Y. Wang, W. Chao, D. Garg, G. Pleiss, B. Hariharan, M. Campbell, and K. Q. Weinberger, "Pseudo-lidar++: Accurate depth for 3d object detection in autonomous driving," *Computing Research Repository (CoRR)*, vol. abs/1906.06310, 2019. [Online]. Available: http://arxiv.org/abs/1906.06310

[38] J.-R. Chang and Y.-S. Chen, "Pyramid stereo matching network," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 5410–5418.

[39] F. Wulff, B. Schäufele, O. Sawade, D. Becker, B. Henke, and I. Radusch, "Early fusion of camera and lidar for robust road detection based on u-net fcn," in *2018 IEEE Intelligent Vehicles Symposium (IV)*, 2018, pp. 1426–1431.

[40] S. Minaee, Y. Y. Boykov, F. Porikli, A. J. Plaza, N. Kehtarnavaz, and D. Terzopoulos, "Image segmentation using deep learning: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, pp. 1–1, 2021.

[41] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[42] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[43] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations (ICLR) 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1409.1556

[44] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, "Dive into deep learning," *Computing Research Repository (CoRR)*, vol. abs/2106.11342, 2021. [Online]. Available: https://arxiv.org/abs/2106.11342

[45] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 3431–3440.

[46] W. Liu, A. Rabinovich, and A. C. Berg, "Parsenet: Looking wider to see better," *Computing Research Repository (CoRR)*, vol. abs/1506.04579, 2015. [Online]. Available: http://arxiv.org/abs/1506.04579

[47] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 40, no. 4, pp. 834–848, 2018.

[48] H. Noh, S. Hong, and B. Han, "Learning deconvolution network for semantic segmentation," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[49] Z. Zhang, Q. Liu, and Y. Wang, "Road extraction by deep residual u-net," *IEEE Geoscience and Remote Sensing Letters*, vol. 15, no. 5, pp. 749–753, 2018.

[50] J. Fu, J. Liu, Y. Wang, J. Zhou, C. Wang, and H. Lu, "Stacked deconvolutional network for semantic segmentation," *IEEE Transactions on Image Processing*, pp. 1–1, 2019.

[51] P. J. Besl and N. D. McKay, "A method for registration of 3-d shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 14, no. 2, pp. 239–256, 1992.

[52] K. S. Arun, T. S. Huang, and S. D. Blostein, "Least-squares fitting of two 3-d point sets," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. PAMI-9, no. 5, pp. 698–700, 1987.

[53] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, p. 509–517, Sep. 1975. [Online]. Available: https://doi.org/10.1145/361002.361007

[54] M. Rosenblatt, "Remarks on Some Nonparametric Estimates of a Density Function," *The Annals of Mathematical Statistics*, vol. 27, no. 3, pp. 832 – 837, 1956. [Online]. Available: https://doi.org/10.1214/aoms/1177728190

[55] E. Parzen, "On Estimation of a Probability Density Function and Mode," *The Annals of Mathematical Statistics*, vol. 33, no. 3, pp. 1065 – 1076, 1962. [Online]. Available: https://doi.org/10.1214/aoms/1177704472

[56] "OpenCV - OpenCV," https://opencv.org/, accessed: 2021-04-20.

[57] "OpenCV: Camera Calibration and 3D Reconstruction," https://docs.opencv.org/master/d9/d0c/group__calib3d.html, accessed: 2021-04-20.

[58] M. Pharr, W. Jakob, and G. Humphreys, *Physically Based Rendering: From Theory to Implementation*, 3rd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2016, ISBN: 0128006455.

[59] R. Finkel and J. Bentley, "Quad trees: A data structure for retrieval on composite keys." *Acta Inf.*, vol. 4, pp. 1–9, 03 1974.

[60] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Surface reconstruction from unorganized points," *SIGGRAPH Comput. Graph.*, vol. 26, no. 2, p. 71–78, Jul. 1992. [Online]. Available: https://doi.org/10.1145/142920.134011

[61] Y.-J. Liu and M. M.-F. Yuen, "Optimized triangle mesh reconstruction from unstructured points," *The Visual Computer*, vol. 19, no. 1, pp. 23–37, Mar 2003. [Online]. Available: https://doi.org/10.1007/s00371-002-0162-2

[62] T. Akenine-Mller, E. Haines, and N. Hoffman, *Real-Time Rendering, Fourth Edition*, 4th ed. USA: A. K. Peters, Ltd., 2018, ISBN: 0134997832.

[63] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, 07–09 Jul 2015, pp. 448–456. [Online]. Available: http://proceedings.mlr.press/v37/ioffe15.html

[64] S. Yeung, A. Kannan, Y. N. Dauphin, and L. Fei-Fei, "Tackling over-pruning in variational autoencoders," *Computing Research Repository (CoRR)*, vol. abs/1706.03643, 2017. [Online]. Available: http://arxiv.org/abs/1706.03643

[65] D. Berthelot, C. Raffel, A. Roy, and I. J. Goodfellow, "Understanding and improving interpolation in autoencoders via an adversarial regularizer," *Computing Research Repository (CoRR)*, vol. abs/1807.07543, 2018. [Online]. Available: http://arxiv.org/abs/1807.07543

[66] M. Pitropov, D. E. Garcia, J. Rebello, M. Smart, C. Wang, K. Czarnecki, and S. Waslander, "Canadian adverse driving conditions dataset," *The International Journal of Robotics Research*, vol. 40, no. 4-5, p. 681–690, Dec 2020. [Online]. Available: http://dx.doi.org/10.1177/0278364920979368

[67] J. Saarinen, H. Andreasson, T. Stoyanov, and A. J. Lilienthal, "Normal distributions transform monte-carlo localization (ndt-mcl)," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 382–389.

[68] Y. Sugiyama, M. Fukui, M. Kikuchi, K. Hasebe, A. Nakayama, K. Nishinari, S. ichi Tadaki, and S. Yukawa, "Traffic jams without bottlenecks—experimental evidence for the physical mechanism of the formation of a jam," *New Journal of Physics*, vol. 10, no. 3, p. 033001, mar 2008. [Online]. Available: https://doi.org/10.1088/1367-2630/10/3/033001

[69] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015.