UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Computer Science Curriculum

Jan Aare van Gent

Using LiDAR as Camera for End-to-End Driving

Master's Thesis (30 ECTS)

Supervisor(s): Tambet Matiisen, MSc

Tartu 2021

Using LiDAR as Camera for End-to-End Driving

Abstract:
Research on autonomous driving has seen a growing surge in popularity in the last decade. One of the more interesting avenues of autonomous driving, known as end-to-end (E2E) driving, involves training a neural network to predict control signals directly from input sensors. Usually, the main sensor used for E2E driving is a regular front-facing camera. Cameras are the preferred sensor since they can perceive the road and traffic the way humans see it. Additionally, to make self-driving affordable, the sensor set should be relatively simple and cost-effective, which simple front-facing dashcams excel at. However, Light Detection And Ranging (LiDAR) instruments give accurate distance estimations and can be more robust to weather and lighting conditions than regular cameras. In this thesis, the feasibility of using LiDAR as a camera for E2E driving is evaluated. Specifically, the sensor examined is the Ouster OS1-128 LiDAR instrument, which can output measurements as a 360-degree raster image with range, intensity and ambience channels. A convolutional neural network (CNN) was trained to predict steering angles from LiDAR images. In addition, multiple experiments, including varying the data and the network architecture, were performed. The trained models were evaluated with both offline with open-loop metrics and online with closed-loop metrics. The evaluation results confirm that using LiDAR measurements as a raster image (instead of point cloud) allows to make use of the well-tested CNN networks for E2E driving. This means that Ouster OS1-128 lidar can be used as a drop-in replacement for the camera in E2E driving solutions, with potential improvements due to range sensing, less sensitivity to weather and lighting conditions and novel data augmentation opportunities.

Keywords:
Computer Vision; Machine Learning; Deep Learning

CERCS:
P176 Artificial intelligence
P170 Computer science, numerical analysis, systems, control

LiDARi kasutamine kaamerana närvivõrkudel põhinevas isejuhtivas autos

Lühikokkuvõte: Isejuhtivate sõidukite valdkond on viimasel aastakümnel näinud märgatavat arengut. Üks huvitavamaid suundi selles valdkonnas keskendub närvivõrgu treenimisele, mis suudab ennustada sensori andmete põhjal auto madala taseme käsklusi (nt. rooli keeramisnurk). Selleks eesmärgiks kasutatakse sensorina kõige levinumalt tavalist pardakaamerat. Seda valikut põhjendatakse sellega, et kaamera näeb ees olevat teed ja liiklust nii nagu inimene. Lisaks, kui eesmärk on laiendada isejuhtivate autode tehnoloogiat massidele, peaks see tehnoloogia olema võimalikult lihtne ja odav, ning just kaamera vastab nendele nõuetele paremini kui mitmed muud sensorid. Küll aga pakub LiDAR väga täpseid kagugusmõõtmisi ning on erinevate ilmastiku- ja valgusolude suhtes tavalisest kaamerast robustsem. Antud magistritöös uuritakse LiDAR kasutamist kaamerana närvivõrkudel põhinevas isejuhtivas autos. Uuritava seadmena kasutati Ouster OS1-128 tüüpi LiDARit, mis väljastab mõõtmistulemusi 360-kraadise rasterpildina, mille kanaliteks on kaugushinnang, tagasipeegelduva signaali intensiivsus ning taustvalgus. Saadud piltide põhjal treeniti konvolutsiooniline närvivõrk , mis ennustab auto rooli keeramisnurka. Lisaks katsetati erinevaid võrgu sisendeid ja väljundeid ning võrreldi saadud tulemusi. Treenitud mudeleid hinnati erinevate meetrikatega, seal hulgas päriselu sõidu tulemustega. Tulemused kinnitavad, et LiDARist saadud rasterpilte saab efektiivselt kasutada levinud konvolutsioonilisi närvivõrgu arhitektuuridega isejuhtivate sõidukite kontekstis ja see võib asendada kaamerat.

Võtmesõnad:
Masinnägemine; Masinõpe; Sügavõpe

CERCS:
P176 Tehisintellekt
P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine

Contents

1 Introduction

Autonomous driving has received much attention from researchers in recent years. While the task of self-driving is interesting, it is also complex and challenging. The main difficulty is caused by a variety of external factors such as road conditions or illumination changes. Recent developments in convolutional neural networks (CNNs) encouraged the development of more sophisticated systems for an autonomous vehicle.

Currently, there are two approaches in autonomous driving research. The modular approach divides autonomous driving into inter-connected modules such as perception, planning and control. For a vehicle to function correctly, input and output of each module should be tuned carefully. The main advantage of the modular approach is testability and interpretability: it is easy to track the system's behaviour and, if needed, to identify the initial source of the errors. While this approach proved to be reliable and efficient in some scenarios, its main drawback is high complexity and high maintenance, as each module must be designed and written by human experts. In addition, it is not obvious if these modules extract enough useful information which other modules could make use of.

Another approach is to let a deep neural network (DNN) learn all the necessary logic needed to output vehicle control commands just from raw sensor inputs, such as camera. This approach, known as end-to-end (E2E) driving, was first introduced as early as 1989 by Dean Pomerleau in a vehicle called Autonomous Land Vehicle In a Neural Network (ALVINN). This vehicle made use of a small multi-layer perceptron (MLP) to transform camera images and the results of laser scans into steering commands [28]. A modern-day follow up to ALVINN, making use of modern deep learning networks, was created by a research team from Nvidia in 2016 and dubbed DAVE-2 [6]. Many similar E2E approaches have appeared since then [39].

One of the sensors that is ubiquitous in the field of autonomous driving (AD) is the Light Detection And Ranging (LiDAR) instrument, which can provide range estimation with millimeter accuracy and is most often used for 3D object detection. If combined, LiDAR and camera data could provide a much needed boost to E2E driving. There are a few issues when trying to fuse LiDAR data with camera data. The first is that the transform between the sensors must be found with careful calibration. The second is that the sensors must be synchronized time-wise, which is complicated in the case of a 360-degree rotating LiDAR. Additionally selecting the best method of fusion might not be obvious.

The Ouster OS-1 family of LiDARs provides a simple solution to both, such that the LiDAR range data aligns with the incoming ambient light data by virtue of using the same infrared sensors for both types of data. Moreover, in comparison to a regular camera, Ouster LiDAR is more robust to different weather and lighting conditions. It can also be used as a drop-in replacement for camera input in E2E driving networks. The raster image data format provided by Ouster LiDARs can already be processed by

conventional and heavily researched CNN models.

As of the time of writing this thesis, no publicly available research has been done with using the Ouster LiDAR as a camera for E2E driving. In this work, the feasibility of Ouster LiDAR as a camera for E2E driving and data collection is tested. Additionally, the model was trained to predict steering angle. Experiments relating to input and output modalities of the network, generalization to new roads, overfitting, and the importance of intervention data were conducted and the results analyzed.

The rest of the thesis is structured as follows. Section 2 covers the theoretical and technical details needed to understand the thesis. Section 3 is the core of this thesis and covers the actual implementation of the network training, system integration, evaluation metrics and experiments. Section 4 details the results of the conducted experiments. This is followed by Section 5, which analyzes and discusses the results. Finally, Section 6 gives a brief summary of the results and suggests directions for future work.

2 Background

This section gives the necessary background information needed to understand the thesis.

2.1 Imitation learning

Imitation learning (IL) is a form of supervised learning where the model is trained to mimic an expert [39]. It is usually contrasted to the alternative approach, reinforcement learning (RL), where the model attempts to learn from a reward function for the environment and find a policy function that maximizes the reward. The choice between IL and RL comes down to whether it is easier to find a reward function for the environment or demonstrate the desired behavior for different states of the environment [19]. The simplest form of IL is behaviour cloning, which includes recording the expert's actions in the environment the model is expected to perform. In the context of E2E driving, this is simply letting the model learn offline from an expert driver via recordings, with sensory information (e.g. camera images) as input and control signals (e.g. steering angles) as output. While behavior cloning can provide good results in certain applications, it tends to be problematic for the majority of use cases, including self-driving. The reason is that the model learns to react to situations the expert has caused. When letting the model take control in the environment, it must react to situations it has caused, which can be different from situations caused by the expert. This effect of learning only from expert induced situations but encountering new ones is known as the distribution shift problem [39]. An example would be where the expert driver is always centered on the lane. The model trained with behavior cloning will encounter a distribution shift as the model deviates more and more from the lane center due to factors, such as wind or bad road conditions.

2.1.1 Data augmentation

A common method used to counter the distribution shift problem includes the use of data augmentation. Data augmentation for images is a heavily researched topic and has been used to increase performance on tasks such as image classification [35]. Some examples of image data augmentation techniques used in AD range from the addition of noise and blurring to varying brightness, contrast, and hues [37]. Other forms of image augmentation used include performing a shift or rotation in perspective to increase the number of unobserved situations along with a recovery signal [4]. For example, a car could be fitted with more than one camera, looking at different directions or being on the right or on the left side of the car (see Figure 1). A different recovery signal could then be generated for each misaligned camera, providing more useful data to recover from suboptimal situations. Another way to get changes in perspective could be done by transforming the image via a viewport transformation. For example, a viewport transform can be applied to an image of a front-facing camera of a car that is centered in its lane

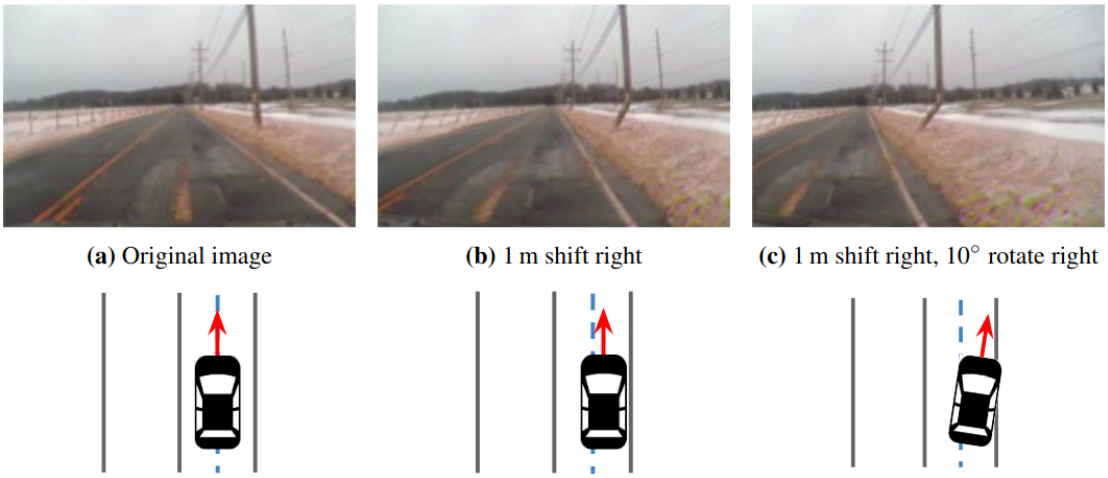
perfectly. The result could be an image observed from the very rightmost edge of the lane and rotated to too much to the right. The accompanying recovery signal could then be a left turn to steer back the the center of the lane and correct its rotation. An example of this type of viewport transformation can be seen in Figure 2.

Figure 1. Data collection setup from early stages of Nvidia's DAVE-2 project. Three GoPro cameras were mounted with suction cups on the front of the vehicle, left, center, and right [4].

There are a few caveats with these target-label changing perspective transforms, as they introduce new problems such as how the recovery signal should be calculated based on the rotation. The simplest solution would be to find the function to produce a recovery signal empirically. However, this involves finding the function through trial and error. A more sophisticated solutions may involve taking into account the current speed and offset from the target trajectory and constrain the recovery time to some fixed amount [6]. Classical control methods, such as the model predictive controller (MPC) can then be used to calculate the recovery signal. However, by letting the model output a trajectory instead of a steering angle, this problem disappears completely.

Another issue is that such viewport transforms introduce visual distortions and artifacts as can be seen in Figure 2. The model may be able to learn to predict the recovery signal based on these artifacts alone, leading to poor real world performance. The artifact problem proved to be a real issue for the DAVE-2 team in later stages of their work, when the network learned to predict the output trajectory by almost completely relying on measuring the distortions and calculating its offset from the expert path [4]. The team's solution was to instead let the network predict manually labeled lane markings and calculate the trajectory between them afterwards. The same problem was encountered by Comma.ai researchers, who were using viewport shift transformations

and predicting trajectories [10]. Their solution was to train the network first without viewport transforms, freeze the weights in the convolutional layers and continue training the model by only updating weights in the fully connected layers. In addition, they applied the KL-divergence loss on the activations of the final convolutional layer to reduce the chance of encoding distortions as learned features [10].



(a) Original image **(b)** 1 m shift right **(c)** 1 m shift right, 10° rotate right

Figure 2. Nvidia's DAVE-2 (PilotNet) project uses shifts and rotations in perspective for data augmentation [4]. Viewport shifts are created by skewing the part of the image below the horizon, introducing distortion artifacts, such as the slanted lower part of the pole.

2.1.2 On-policy learning

An alternative to data augmentation is to increase the number of different situations in the training set by letting the expert correct bad situations and let the model learn from these corrections. The simplest form of this would be to let the model take control until it drifts off the desired path, in which case, the expert intervenes and corrects this offset until a desired state is reached. The expert then gives back control to the model. Only the expert's actions get recorded and added to the training set, as there is no point in letting the model learn to drive into dangerous situations. This type of learning from interventions is useful because it directly addresses the situations where the model fails and ignores situations the model is already good at. The issue with this method is that data collection is extremely slow due to having a human in the loop and as the model improves and as it makes less mistakes, data collection becomes even slower as it can only collect data from the few places the expert intervenes. In addition to being slow, it is also be dangerous to evaluate the model on roads which may have other vehicles

or pedestrians nearby. A simple way to mitigate this would be to make the expert be more strict and already make corrections to slight deviations from the path. However, this might result is less useful intervention data as the model will not learn to recover from larger deviations.

Ideally, the training and interventions could be done in a safe graphical simulation environment. The problem is that such simulation environments and real-world environments are just too different and any model trained on data from a graphical simulation will encounter what is known as domain shift [15]. Researches from Nvidia's DAVE-2 project and Comma.ai have resorted to using natural simulators from recordings based on viewport transformations, sometimes also known as feedback synthesizer simulators [10, 4]. In essence, the model would predict a steering angle and a physics engine would calculate the resulting offset from the expert's driven trajectory. This offset would then be used to synthesize a new image just like the data augmentation method mentioned before. The new image then gets fed to the model again and the model predicts a new output, which is then fed to the physics engine again, etc. This now provides a safe environment to test how well the model can imitate the expert driver and where it would fail. This idea could be extended to training by letting the model drive until it is off the desired path by some threshold distance. A classic control algorithm, like the MPC, could then be used to correct the state and the intervention actions are added to the training set.

An even better approach would be to let the model drive and record the incoming images but matching them not to the model outputs, but to the outputs produced by the expert [32]. This method, known as on-policy training, allows for far faster data collection, as there is no need to explicitly wait for an intervention to record training data.

On-policy methods could also be extended to E2E driving models that are predicting trajectories instead of steering angles. The simplest approach would be to let the model drive and as it deviates from the desired path, annotate the input images with the ground truth trajectory in coordinates relative to the car. With the inclusion of localization data and maps, an expert could be a navigational planner, that could produce different types of trajectories [8, 43]. However, on-policy learning is not widely used for E2E driving tasks and tends to be overshadowed by data augmentation methods.

2.2 LiDAR

LiDAR is a technique for remote sensing using laser light [22]. A LiDAR instrument would actively generate pulses of laser light, usually in the infra-red spectrum, to measure the round-trip time the reflected light beam takes to return to the receiver, which can then be converted into a range estimate. The LiDARs used in autonomous driving usually have a 360-degree horizontal field of view (FOV) and multiple (usually 16, 32, or 64) concurrently firing laser beams aimed at different pitch angles, producing a set of relative range estimates. In addition to estimating distance, LiDAR sensors can estimate the

energy of the returning light known as the intensity value. After converting the angles and ranges from polar coordinates to relative 3D euclidean coordinates, the resulting set of points is referred to as a point cloud (see bottom part of Figure 3). One of the main use cases of LiDARs in autonomous driving include object or obstacle detection, which is usually achieved by clustering nearby points into clusters or encoding the point cloud into features which can be used by other machine learning (ML) methods, such as CNNs [29]. Another popular use case includes mapping the surrounding environment in 3D to later make use of the resulting maps for navigation.

2.2.1 Ouster OS1-128

The Ouster OS1-128 is a mid-range (maximum 120 meters) high-resolution (128 beams) imaging LiDAR, which can operate at 10Hz or 20Hz modes and record up to 2,621,440 points per second [24]. It operates at 850 nanometer wavelength infra-red light, which is less prone to absorption by water droplets in humid and foggy environments [25]. In addition to depth and intensity estimations, the LiDAR instrument can also record incoming ambient light of its operational frequency [26]. The firmware and software drivers provided by Ouster for their instruments allow the ambience, intensity and range to be represented as 2D images (see Figure 3). These 2D images could easily be used by different ML methods, such as CNNs, to extract useful visual features for a multitude of computer vision (CV) tasks.

2.3 Visualization of CNNs

While CNNs can often outperform other ML methods in computer vision tasks by learning the most useful or discriminative features, they lack intuitive interpretability. This is especially a problem when it comes to safety-critical systems, such as autonomous driving applications. Multiple solutions have been proposed to veer under the hood of CNNs and to visualize which parts of the input image the network focuses on [30]. The simplest solution involves visualizing the activation maps produced by the kernels of the last layer of the CNN for a given input image. The reason to only focus on the last layer comes from the fact that only the last layer provides the visual features used for any decisions the network has been trained to make. Although a valid approach, the activation maps in the last layer usually have a very low resolution, making direct pixel-to-pixel matching with the input image difficult. While a basic ad-hoc approach would be to upsample the activation maps to the size of the input image, more intricate solutions have been proposed, such as the deconvolution method [40] and sensitivity analysis [36] and their more recent extensions [38, 20], all of which are referred to as gradient-based methods as they attempt to reconstruct the input that maximizes a selected neuron's activation [41]. More recent methods such as CAM [42] and Grad-CAM [34]

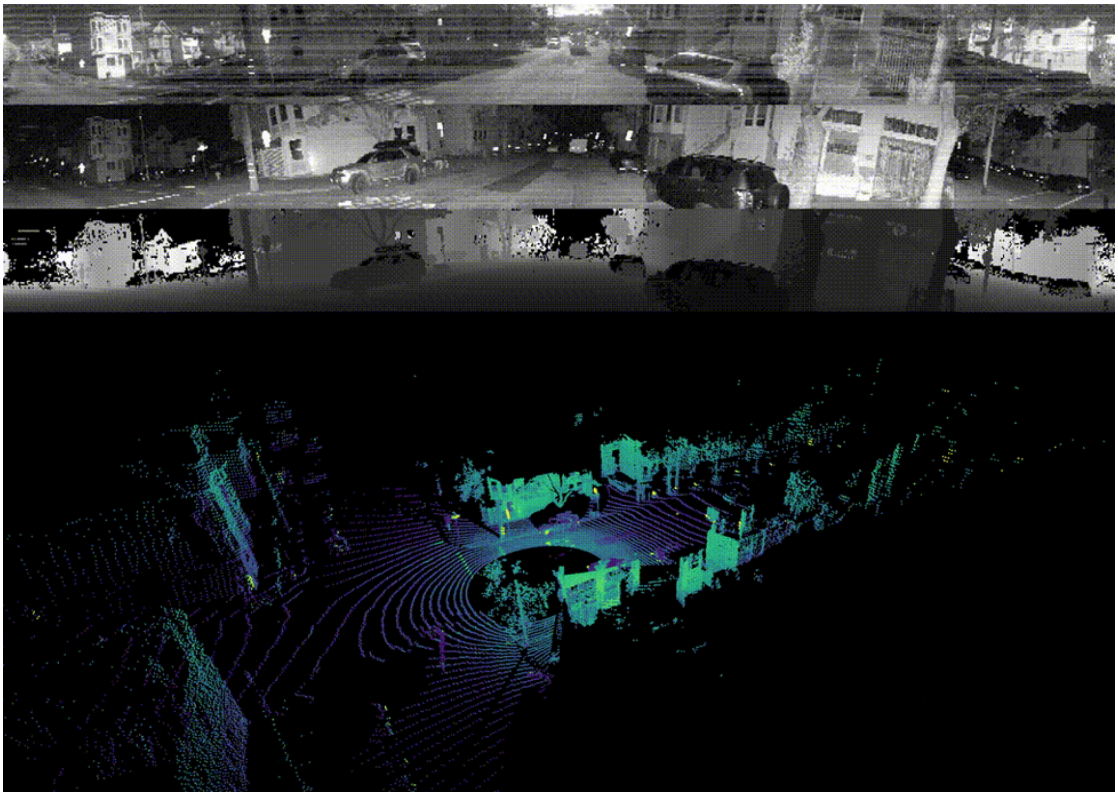


Figure 3. From top to bottom: ambient, intensity, range, and point cloud data. [26]

lack the ability to show accurate pixel-level fine-grained importance and are commonly fused with previously mentioned methods to get accurate visualizations [5].

2.3.1 VisualbackProp

After comparing different CNN visualization methods, the DAVE-2 team at Nvidia chose to develop their own visualization method, termed VisualBackProp, which they claim to be more accurate than previous methods [5]. It also has the added benefit of operating in real-time, which is useful for debugging and visualizing the network live while the car is driving. Example visualizations of their trained model can be seen in Figure 4.

The VisualBackProp method attempts to combine the high resolution of the activation maps of shallow layers with the relevance of the deeper layer activation maps. A block diagram of the method is presented in Figure 5. The activation maps of each layer are first averaged. The averaged activation map of the last layer gets backpropagated by one layer (this includes backpropagation through the activation function and previous convolutional layer). After the averaged map has been backpropagated though the previous layer, the

Figure 4. Images produced by the VisualBackProp method for the end-to-end trained DAVE-2 steering angle predictor model [7].

resulting activation map will have increased in resolution. This up-convolved activation map now gets multiplied element-wise by the averaged activation map of that layer, and the result is again backpropagated to the previous layer. The intermediate element-wise multiplications by other averaged activation maps ensure that the final resulting relevance map is accurate down to the pixel level. These steps are repeated until the first convolutional layer has been backpropagated and the resulting importance mask has the same resolution as the initial input image. This mask is the normalized to values between [0,1] and can be overlayed on the input image to visualize the pixels the network focused on the most.

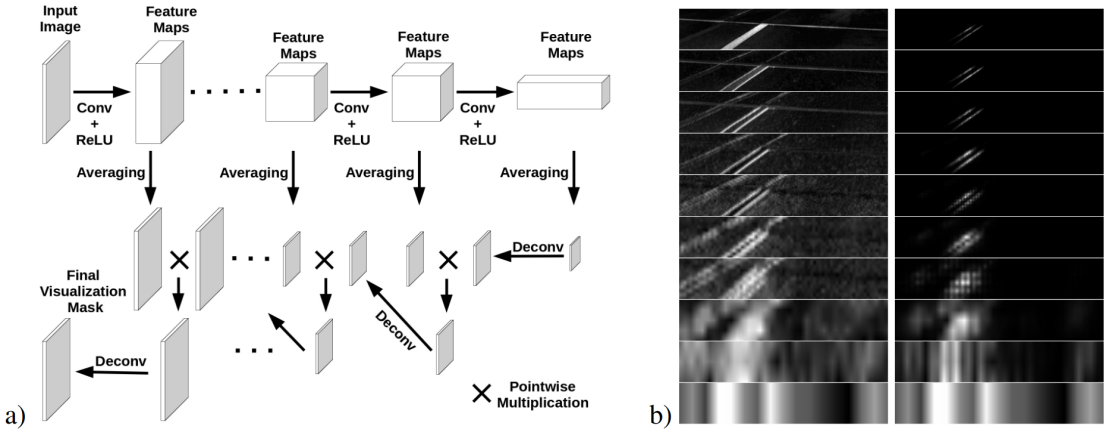


Figure 5. a) A block diagram for the VisualBackProp method. b) Left: the averaged activation maps for each layer. Right: backpropagated average activation maps after intermediate element-wise multiplication, where the bottom image is the initial averaged feature map in the last layer, and the top image is the resulting saliency map [5].

2.4 Robot Operating System

Robot Operating System (ROS) is an open-source robotics software platform which, in addition to many other things, provides a framework for distributed processes, known as ROS nodes [31]. These nodes discover each other through a central process known as ROS Master which, along with a central logger node and the ROS parameter server, form the prerequisite for running any ROS nodes (see Figure 6). ROS uses a TCP-based inter-process communication (IPC) protocol to facilitate message passing between distributed nodes. Once a node receives a message from a topic it has subscribed to, a callback function is triggered in the subscribing node where it can process the incoming message.

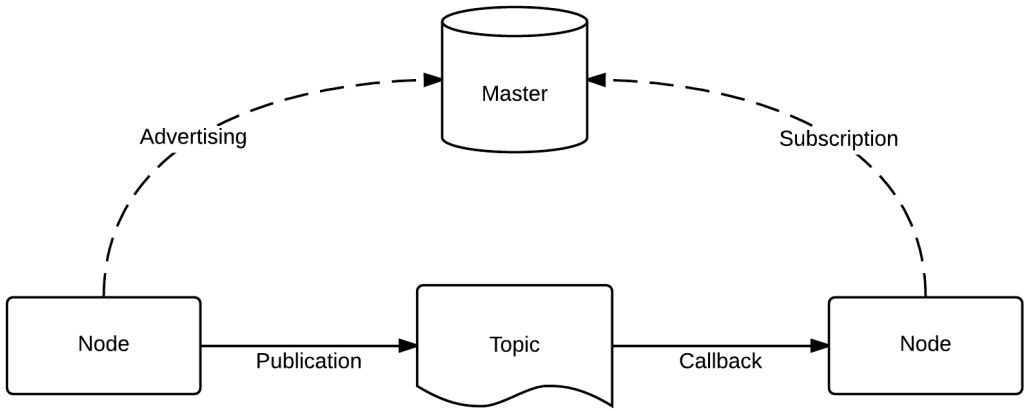


Figure 6. A diagram showing how ROS nodes and ROS Master interact with each other. The nodes can advertise or publish messages into topics and ROS Master can match nodes publishing a particular topic to nodes subscribing to this topic. After matching, the nodes can communicate directly with each other [21].

3 Methods

This section describes the methods and setup used to train and evaluate the network in different experiments. First, the Dataset subsection gives an overview of the type of data used and the process of gathering and storing the data. Then, the Network architecture section explains the network architecture used in this thesis along with its design decisions. The section on Training details outlines the training process from a theoretical perspective down to the training loop implementation details. The System integration section shows exactly what is needed to get a trained network model to control a car in real life.

3.1 Dataset

Since no public dataset exists for Ouster OS1-128 images coupled with steering angles, a new dataset was created as part of this thesis. The training data was collected from different road types, predominantly dirt roads and some low-traffic highways as well, over a period spanning from February to May 2021. Data was also gathered from very different lighting and weather conditions. LiDAR and steering angle measurements were recorded at rate of 10Hz. To make sure the steering angles were as accurate as possible and to get more varied data, the expert driver always drove as fast as possible given the present road conditions, nearby pedestrians and other vehicles, and traffic regulations without sacrificing safety or comfort.

Due to issues found in the Ouster LiDAR ROS driver in relation to generating a correct range image, modifications had to be made regarding the creation of the range image. For each pixel in the range image, equation 1 was used, where V_i is the final 8-bit value of the pixel, R_i is the range estimate for that pixel, and R_{max} is the maximum range, which is given as a parameter to the driver and a value of 50 meters was used for producing this dataset. This results in a range image where the range between 0-50 meters is linearly mapped to the range of integers 255-0. Pixels with range estimates larger than 50 meters would be set to 0. The choice of pixel values decreasing as the range increases order was made to remain consistent with far away objects outside of the range of the LiDAR, which would have a value of 0.

$$V_i = 255 - \left\lfloor 255 \cdot \min\left(\frac{R_i}{R_{max}}, 1\right)\right\rfloor \tag{1}$$

The recorded steering wheel angles were divided by the vehicle specific steering ratio (14.7 for the Lexus RX 450h), resulting in the yaw angle of the frontal wheels (in radians). One of the advantages of recording the angle of the frontal wheels is that the data and the resulting angle predicting model are agnostic to the vehicle specific steering ratio and can be extended to be used on other vehicles of similar length. Another advantage is that the software stack used in our test vehicle already expects the yaw angle of the

frontal wheels as input and hence, no additional conversion is required after producing a prediction. Most works on E2E lateral controllers, including the DAVE-2 project, predict the curvature as output, which is the inverse of the turning radius ($\frac{1}{r}$) [6]. The advantage of predicting curvature is that, theoretically, it is independent of the vehicle length and can be converted to the frontal wheel angle if the vehicle length is known. However, to stay in the bounds of the road, longer vehicles must start turning later than shorter ones, meaning that an E2E image-to-curvature model would simply overfit on the vehicle length and any other vehicle lengths would produce a distribution shift for the trained model. To extend an E2E lateral control model to different types of vehicles, an optimal solution would be to predict the future desired trajectory as 2D or 3D points and let a classical control algorithm tuned to the specific vehicle handle the actual steering. In fact, the DAVE-2 team also switched from predicting curvature to predicting future waypoints [4]. However, to maintain simplicity, the model trained in the context of this thesis predicts the frontal wheel yaw angle.

The distribution of the steering angles in the collected dataset is heavily biased toward straight driving (see Figures 8 and 9) with the standard deviation having a value of 0.04 radians (2.24 degrees) and the largest offset being 0.57 radians (33.03 degrees).

To counter the distribution shift resulting from behaviour cloning, some drives included letting the model drive until a potentially dangerous situation occurred in which case the expert took over, corrected the state, and let the model resume control. Only the interventions by the expert were recorded and added to the final dataset.

The total size of the collected dataset is 43826 images or about 1.2 hours of driving data made up of 30 different recordings (see sample images in Figure 7). A more detailed overview of the dataset is provided in Table 1. The dataset is stored as a collection of folders, where each folder corresponds to a single recording (30 folders in total), where each folder contains PNG format images with the RGB channels being used to combine the ambience, intensity and depth channels. In addition to image files, each folder contains a CSV file, which maps each image to its target steering angle in radians. The total size of the dataset is 12GB.

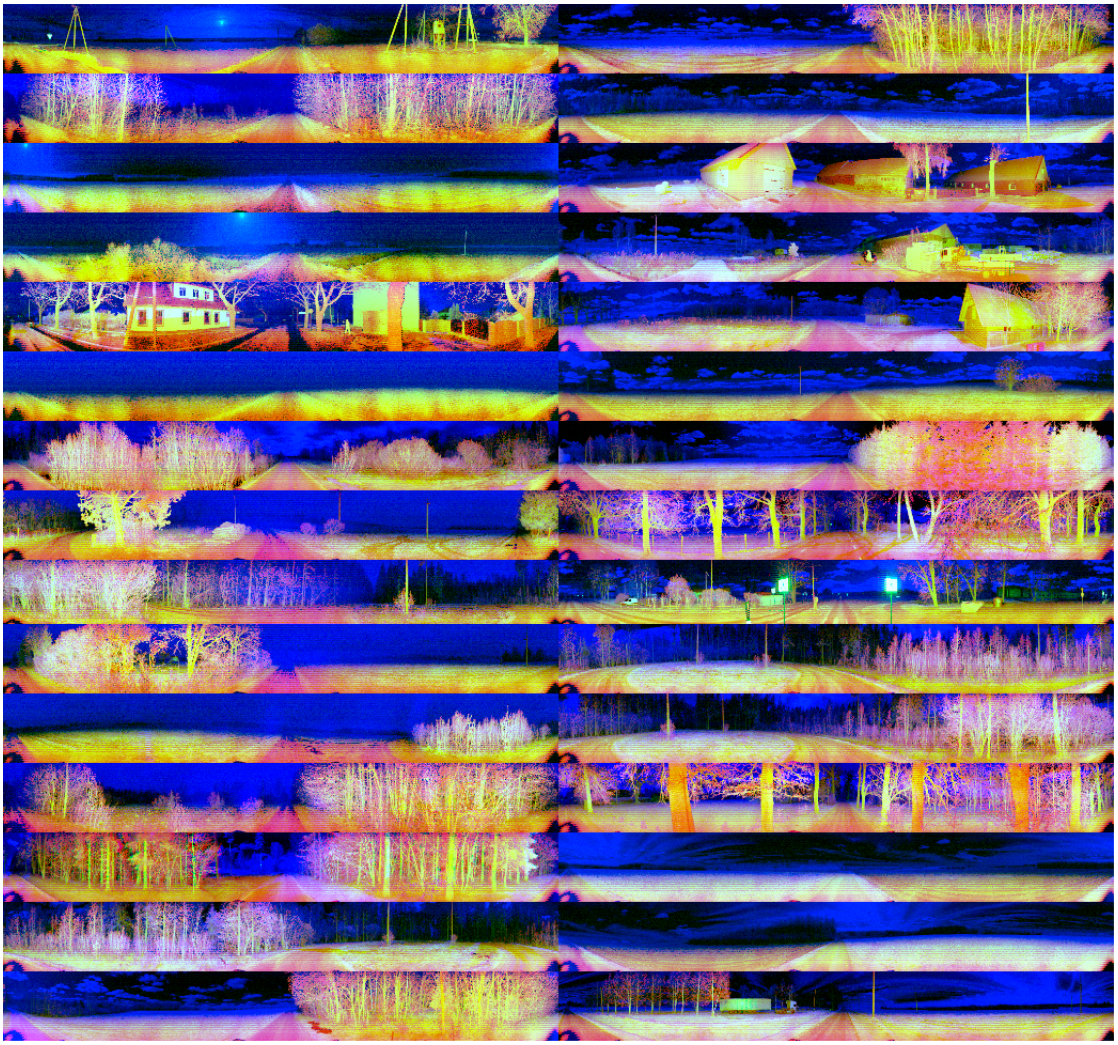


Figure 7. Sample ambience, intensity, depth images presented in RGB format from each recording ordered by date of recording. Left column: recordings 1-15. Right column: recordings 15-30.

Table 1. Overview of collected dataset. The "Intervention data" field shows whether only interventions by expert were recorded or not. Data from on-policy driving was discarded and is not represented in the "Size" field.

#	Date	Time	Road type	Road conditions	Size	Intervention data
1	2021-02-19	15-45-46	dirt road	snow	2053	no
2	2021-02-19	15-49-36	dirt road	snow	2005	no
3	2021-03-02	14-16-46	dirt road	snow	3034	no
4	2021-03-02	14-28-10	dirt road	snow	364	no
5	2021-03-02	14-29-02	highway	wet	2350	no
6	2021-03-03	19-47-40	dirt road	snow	3277	no
7	2021-03-29	15-26-56	dirt road	snow	3243	no
8	2021-03-30	15-39-55	dirt road	wet	76	yes
9	2021-03-30	15-46-04	dirt road	wet	212	yes
10	2021-03-30	15-47-04	dirt road	wet	1945	yes
11	2021-03-30	15-58-26	dirt road	wet	3112	no
12	2021-03-30	16-13-04	dirt road	wet	2713	no
13	2021-04-01	14-22-34	dirt road	dry	122	yes
14	2021-04-01	14-23-49	dirt road	dry	104	yes
15	2021-04-01	14-25-09	dirt road	dry	204	yes
16	2021-04-01	14-26-57	dirt road	dry	265	yes
17	2021-04-01	14-34-07	dirt road	dry	121	yes
18	2021-04-01	14-36-43	dirt road	dry	448	yes
19	2021-04-01	14-41-22	dirt road	dry	939	yes
20	2021-04-01	14-55-07	dirt road	dry	1391	yes
21	2021-04-01	15-05-37	dirt road	dry	2906	no
22	2021-04-01	15-11-12	dirt road	dry	3014	no
23	2021-04-01	15-18-11	highway	dry	802	no
24	2021-04-01	15-21-24	highway	dry	2110	no
25	2021-05-01	17-24-23	dirt road	dry	99	yes
26	2021-05-01	17-25-33	dirt road	dry	101	yes
27	2021-05-01	17-37-31	dirt road	dry	3054	no
28	2021-05-02	14-44-44	dirt road	dry	588	yes
29	2021-05-02	14-47-59	dirt road	dry	1178	yes
30	2021-05-02	14-57-39	dirt road	dry	1996	yes

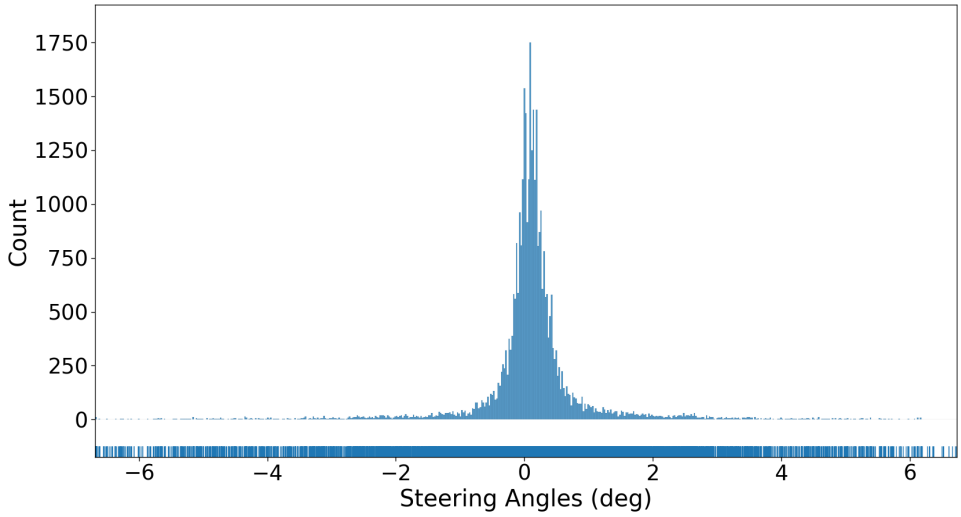


Figure 8. Distribution of the steering angles in the collected dataset. The plot is centered on the mean of the steering angles and is limited on both sides by 3 standard deviations.

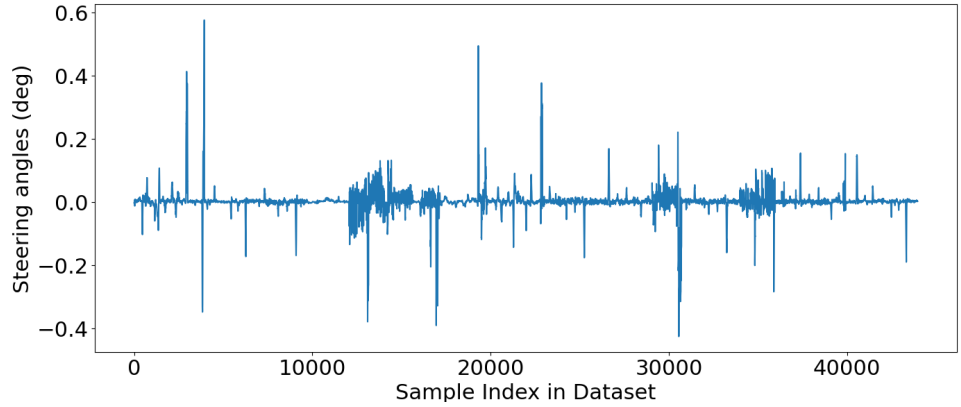


Figure 9. Line plot of steering angles over time for the entire dataset, made up of concatenated recordings. The breaks between recordings are not shown.

3.2 Network architecture

The architecture of the network is based on the DAVE-2 PilotNet network (Figure 10) with minor differences [6]. This choice was made due to the proven track record of PilotNet and the relative simplicity of the network, making training time faster and the model less prone to overfitting. One of the improvements made to the PilotNet in this thesis is the addition of batch normalization layers, which stabilizes the training process by making the optimization landscape significantly smoother [33]. Another addition to the model is the cropping operation done on the input layers, which is used to extract a 90-degree horizontal FOV front-facing image, with the intention of making the model smaller and to not process uninformative parts of the image. The same cropping operation also extracts vertically only the rows of pixels which encompass the road and not the sky or lowest rows of pixels where the hood of the car is visible. The cropping is visualized in Figure 11. A detailed overview of the final network used is shown in Table 2.

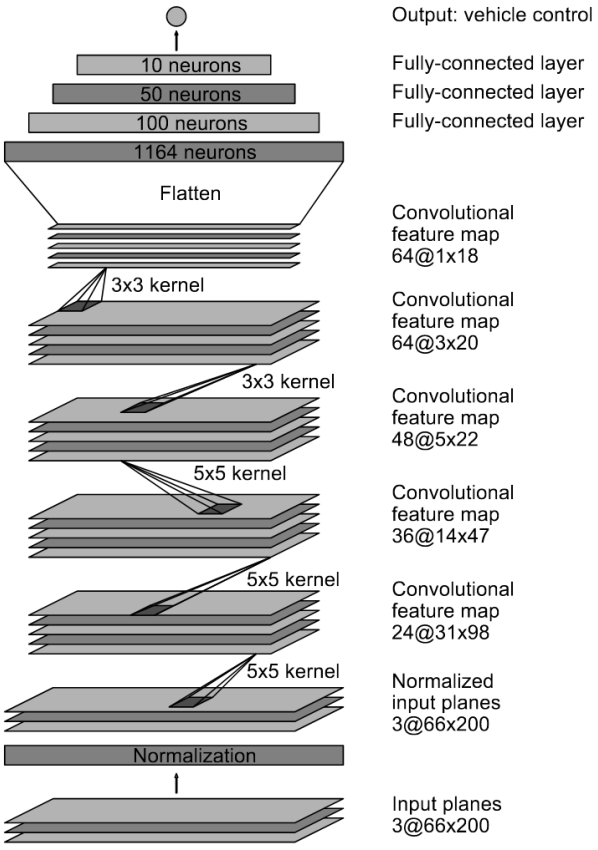


Figure 10. The PilotNet architecture from the DAVE-2 project [6].

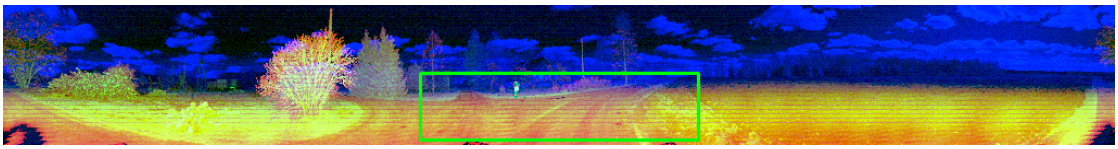


Figure 11. A sample image showing the cropping area as the green box. The crop excludes pixels above the horizon and outside of a 90-degree horizontal FOV.

Table 2. Network architecture used in this thesis. Convolution and linear layer parameters were chosen based on the PilotNet architecture. The initial size of the full 360-degree image is $3 \times 128 \times 1024$. The total number of trainable parameters is around 298 000.

Layer name	**Output size**	**# of params**
Crop	$3 \times 61 \times 256$	0
Convolution2D 1	$24 \times 29 \times 126$	1824
BatchNorm2D 1	$24 \times 29 \times 126$	48
ReLU 1	$24 \times 29 \times 126$	0
Convolution2D 2	$36 \times 13 \times 61$	21636
BatchNorm2D 2	$36 \times 13 \times 61$	72
ReLU 2	$36 \times 13 \times 61$	0
Convolution2D 3	$48 \times 5 \times 29$	43248
BatchNorm2D 3	$48 \times 5 \times 29$	96
ReLU 3	$48 \times 5 \times 29$	0
Convolution2D 4	$64 \times 3 \times 27$	27712
BatchNorm2D 4	$64 \times 3 \times 27$	128
ReLU 4	$64 \times 3 \times 27$	0
Convolution2D 5	$64 \times 1 \times 25$	36928
BatchNorm2D 5	$64 \times 1 \times 25$	128
ReLU 5	$64 \times 1 \times 25$	0
Flatten	1600	0
Linear 1	100	160100
BatchNorm1D 1	100	200
ReLU 6	100	0
Linear 2	50	5050
BatchNorm1D 2	50	100
ReLU 7	50	0
Linear 3	10	510
ReLU 8	10	0
Linear 4	1	11

3.3 Training details

The regression loss function used for training was the mean absolute error (MAE) function. A low MAE for steering angle prediction has been shown to correlate better with real-world driving performance than a low mean squared error (MSE) (equations for both losses in eq. 2), which is commonly used for regression problems [9]. This could be because MSE places heavy emphasis on correcting large errors and outliers, and focuses less on smaller errors, which could be just as important. There is also evidence that for image-based DNN regression tasks, the loss function choice is simply a hyper parameter and that training with one loss function might give better results for an other loss, even better than training with the other loss in the first place [17].

$$\begin{aligned} \text{MSE} &= \frac{1}{n}\sum (y_i - \hat{y}_i)^2 \\ \text{MAE} &= \frac{1}{n}\sum |y_i - \hat{y}_i| \end{aligned} \tag{2}$$

The Adam optimizer was used to optimize the loss during training, which is a common optimizer for DNNs and often outperforms more conventional optimizers such as stochastic gradient descent (SGD) [16]. The parameters for Adam were the default parameters provided by the PyTorch Adam implementation ($\text{lr} = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1e-8$) as early experimentation showed these give the best performance. In addition, the learning rate was multiplied by a factor of 0.6 if the maximum absolute error in the validation set did not improve for 4 epochs straight. For the batch size, 128 proved to be the best. The training lasted a total of 50 epochs, where a single epoch was the size of the entire training set. For all epochs, only the model resulting in the smallest maximum absolute error on the validation set was saved.

The validation set consists of recordings 21, 22, and 23 (see Table 1 for more details on each recording). Therefore, the train and validation sets make up roughly 84% and 16% of the dataset respectively.

To mitigate the distribution shift caused by pure behavioral cloning, rotational data augmentation was used. Since the images have 360-degree FOV, translating the 90-degree FOV crop left or right corresponds to a viewport rotation. A recovery signal equal to the produced rotation in radians was added to the existing target steering angle. The magnitude of the rotations was taken from a normal distribution with a standard deviation double that of the standard deviation of the training set as was done for the DAVE-2 project [6]. During early experimentation, it was found that while rotational data augmentation does help, using larger magnitude rotations made the steering significantly more aggressive. If very little rotational data augmentation was used, the network never saw large steering angles and hence, did not learn to produce them.

To alleviate the problem of imbalanced data due to most of the driving being straight driving with very few curves, the proportion of curves sampled for each batch was

increased, resulting in the oversampling of curves. This was done by using a random weighted sampler and giving higher weights to samples where the absolute target steering angle was above some threshold. This angle threshold (0.005 radians) was found empirically and can be thought of as a dividing line between curves and non-curves. While other data balancing techniques were also tried, such as setting the weights to be equal to the softmaxed MAE losses of all training samples after each batch when training, or binning the steering angles and setting the weights as the inverse number of samples in the bin, the simple thresholding method proved to be more effective. The weights were calculated such that there is a 50/50 (found empirically to be the best split) chance to sample a curve or a non-curve. The exact formula for calculating the weight of a curve sample is given in equation 3, where q_1 is the proportion of instances of a particular class (such as non-curves), q_2 the desired proportion (0.5 or 50%) with random weighted sampling, and w the final weight given to all instances of the specified class. The weights for non-curves are assumed to be 1.

$$w = \frac{q_2(1 - q_1)}{q_1(1 - q_2)} \tag{3}$$

The training and evaluation was done on 4 Nvidia GeForce RTX 2080 devices in a multi-processing fashion, where each graphics processing unit (GPU) would have a full replica of the model. This method is also known as data parallelism. During training, each GPU process would get different data from the training set and after separately completing intermediate stages of backpropagation, would sync and average the gradients across all GPUs so that the optimizer would make exactly the same updates on all devices and the models would always be in sync [18].

During evaluation, each sample in the validation set is sampled only once and thus, each process uses a data loader that can access only its portion of the dataset. Each process sums the losses of each sample in its current validation batch and uses multi-processing reduction to sync and sum the results. The final sum is then divided by the number of samples in the validation set and the final average MAE for the evaluation step is produced. The maximum absolute error is also calculated in a similar way, where each process finds the maximum error in the validation set, syncs with the other processes, and finds the maximum of the maximum errors from each process. Training a single model for 50 epochs lasted around 3h.

3.4 System integration

The test vehicle used for this thesis is a customized Lexus RX 450h (Figure 12) fitted with a PACMod v3 drive-by-wire system provided by AutonomouStuff [3], which can be used to control the car accurately from an external source (such as a PC) over a Controller Area Network (CAN) bus. The PACMod system has publicly available ROS drivers

Figure 12. The customized Lexus RX 450h from the University of Tartu's Autonomous Driving Lab (ADL)[1].

that can be used to receive vehicle control commands and publish PACMod specific messages to a ROS-based CAN driver, which forwards the messages through a physical CAN interface onto a CAN bus (see Figure 13). The CAN bus acts as a bridge between the PACMod system and the PC. AutonomouStuff also provides an additional closed source ROS package, the Speed and Steering Control (SSC) module, that implements low level steering and speed controllers tuned for human like driving, and includes a configuration mode specifically tuned for the Lexus RX450h vehicle [2]. The SSC acts as an interface between the user code and the PACMod system by listening to user published high-level control commands, translates them to comfortable human-like control signals that are vehicle specific, and forwards the outputs to the PACMod ROS node. Therefore, a self-driving software stack will only need to send the desired speed and steering angles to the SSC, which in turn sends acceleration, brake and steering wheel torque commands to the PACMod.

To run the trained network model in the car, Nvidia's TensorRT was chosen as the inference runtime [23]. A simple Python based ROS node was written to

[1]https://www.cs.ut.ee/et/isejuhtivate-soidukite-labor

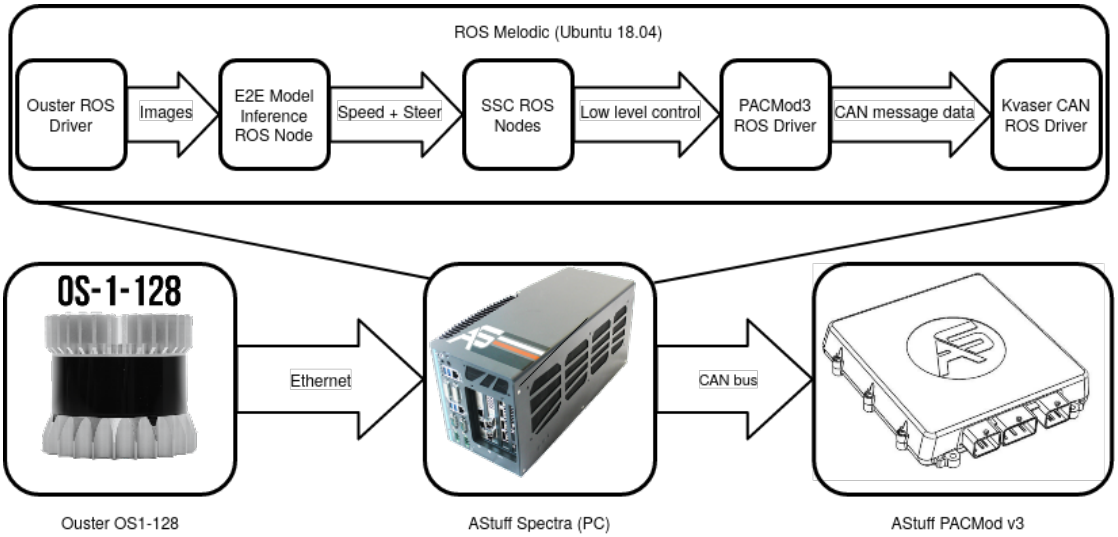


Figure 13. An overview of the entire system used in the test vehicle to control the car with Ouster LiDAR sensor information [1, 27].

1. load a trained model in the Open Neural Network Exchange (ONNX) format and convert it to a TensorRT engine;
2. read in synchronized ambience, intensity and range images produced by the Ouster ROS driver;
3. run inference on the combined image with the TensorRT engine;
4. publish the output as a steer command (coupled with a constant speed command) to the SSC ROS node.

3.5 Evaluation

Generally, evaluation metrics for self-driving models can be divided into two categories: open-loop and closed-loop metrics [39]. Open-loop metrics, where we compare the model's steering angle prediction to that of human generated steering angles for the same images, are the easiest way to evaluate the model, as it can be done entirely offline without requiring the car to drive. The main open-loop metric used in this thesis is MAE, as the models were optimized for this loss and it has shown to correlate better with real-world driving performance [39]. In addition, root mean squared error (RMSE) is also reported as many works on E2E lateral control use RMSE to measure their performance [13] [12]. To measure the smoothness of the steering angles over time, we use the whiteness metric (W) defined in Equation 4, which has the unit of degrees per

second [13]. The variables P and D denote the steering angle predicted by the model and the size of the dataset respectively. The partial differential was approximated by finding the differences between consecutive steering angles and dividing them by the sampling period (0.1 seconds).

$$W = \sqrt{\frac{1}{D}\sum_{i=1}^{D}\frac{\partial P}{\partial t}} \tag{4}$$

For comparison, these metrics are also provided for a zero-predictor model and a training mean predictor model which, since the training mean is very close to 0, should be quite similar. Finally, maximum absolute error is used to measure the worst case performance.

Closed-loop metrics require online testing with a real car or a simulated one in a realistic simulator with accurate physics calculations. Since there is no accurate simulation that can simulate Ouster LiDAR images at this time, all closed-loop metrics are evaluated in real-life on the test vehicle. Since this type of evaluation is relatively time-consuming, each model is evaluated only on a single track once in both directions. The track chosen for evaluation is the training track which was used to gather at least half of the training data. This track is approximately 5 km long in each direction, meaning an evaluation on the track includes a 10 km drive. When other vehicles or pedestrians were close, the safety driver took over. Since the model can not control its speed nor has it been trained to deal with other agents on the road, interventions caused by cars or pedestrians did not contribute to the final metrics. In addition, due to speed limits in certain sections and other road agents, the evaluation was done at a fixed speed of 30 km/h.

There are many types of closed-loop metrics reported in literature [39]. However, many of these are feasible only in a simulation environment or require high amounts of data. The most important closed-loop metric is the number of interventions (also called infractions in literature), which is also used in this thesis. In addition, lateral deviation from the expert trajectory is calculated based on the absolute coordinates obtained by the NovAtel PwrPak7 receiver, which combines the inertial navigation system (INS) data with real-time kinematic positioning (RTK) to get centimeter-level accuracy. The lateral errors were found for each point in the model-driven trajectory. For each model-driven trajectory point, the two closest expert trajectory points were used to construct a line, from which the smallest distance between the point and the line was found. The lateral errors (measured in meters) were then reduced to MAE and RMSE, which are presented as additional closed-loop metrics. While MAE and RMSE give an overview of the lateral errors, they do not specifically measure how often the model deviates dangerously far from the expert trajectory. To measure the proportion of time the model spends too far from the expert trajectory, a new metric dubbed failure rate is introduced. The failure rate is dependant on a threshold used to decide if the model is too far from the expert trajectory is based on the maximum lateral error allowed by Nvidia's DAVE-2 team

in their simulation, which was 1 meter [6]. To see how well the open-loop whiteness generalizes to a closed-loop performance, closed-loop whiteness was also calculated based on the actual steering done by the car. Since the car cannot make sudden turns (the torque of the steering wheel is determined by SSC and PACMod), closed-loop whiteness will be much smaller.

4 Results

This section outlines the experiments performed and the results of each of the conducted experiments.

4.1 Base model

Before diving into the different experiments, the results of the model trained as described in Section 3 are presented. This model is referred to as **BASE**. Table 3 contains the open-loop metrics for the **BASE** model and provides a comparison to baseline methods. The first baseline method, **ZERO** is a model that simply outputs 0 as the steering angle for any input image provided. This is useful to compare and see how informative the metrics MAE and RMSE are and how the behave. In reality, they also provide information on the expert steering angles themselves, with RMSE representing the standard deviation of the expert steering angles. If the network struggles to find meaningful features from the images in the dataset, it is most likely going to start predicting the mean values of the target label, which may yield better results than **ZERO**. Therefore, a baseline predicting the mean of the steering data (referred to as **MEAN**) from the training set is also used for comparison. Finally, the human expert's steering angle whiteness is calculated and compared to the whiteness of **BASE**.

Table 3. Open-loop evaluation metrics on test data collected from the training track.

Model	RMSE [Degrees]	MAE [Degrees]	Whiteness [Degrees / Second]	Maximum error [Degrees]
BASE	7.44	6.29	64.25	28.80
Human driver	N/A	N/A	0.67	N/A
ZERO	0.96	0.32	0	9.76
MEAN	0.95	0.31	0	9.83

Based on the open-loop metrics, the performance of **BASE** seems to be relatively bad, as its predicted steering angle constantly fluctuates and bad results for all metrics when comparing to the baselines. Since most of the driving includes keeping the steering angles very close to 0, **ZERO** and **MEAN** baselines show relatively good performance on open-loop metrics. However, neither of the baselines could be used to actually drive on the road, while **BASE** was actually able to keep on the road for a majority of time, even being able to handle multiple sharp curves.

Table 4. Closed-loop evaluation metrics on test data collected from the training track.

Model	Interventions	MAE [Meters]	RMSE [Meters]	Whiteness [Degrees / Second]	Failure Rate %
BASE	1	0.25	0.33	0.77	0.28
Human driver	N/A	N/A	N/A	0.67	N/A

When looking at the closed-loop metrics of **BASE** (see Table 4) on the training track, it becomes clear that the model is able to drive the track relatively well. Not only does it seem to remain relatively close to the expert trajectory, with only 0.28% of the time being further than 1 meter from the expert trajectory, it also has a whiteness score comparable to a human driver and only had a single intervention in the course of two 5 km drives. The likeliest cause of the intervention were visible tracks going off-road. Interestingly enough, most models evaluated in successive methods did not fail in the same point on the training track as **BASE** did. The closed-loop whiteness is orders of magnitude lower than open-loop whiteness and confirms the fact that the steering control pipeline from SSC to PACMod and the actuation of the actual vehicle filter the noisy steering angle predictions heavily. These initial results suggest that open-loop metrics should be taken with a grain of salt as they can be significantly worse than the closed-loop metrics.

4.2 Generalization to new roads

If a model is trained and tested with driving data acquired from the same road, it is expected to perform well on that road and worse on previously unseen roads. However, when getting more data from more different roads, the question arises whether the model is able to generalize better on unseen roads. For this experiment, the **BASE** model evaluated on the training track and a previously unseen track (referred to as the test track) for comparison. The test track was chosen based on relatively little traffic and similar road conditions to the training track.

Table 5. Open-loop evaluation metrics of model **BASE** on test data collected from the training track and a previously unseen track.

Track	RMSE [Degrees]	MAE [Degrees]	Whiteness [Degrees / Second]	Maximum error [Degrees]
Training	7.44	6.29	64.25	28.80
Test	7.85	6.48	61.44	24.55

The open-loop metrics from Table 5 show very similar yet slightly worse performance on the test track. While the model has better whiteness and maximum error scores on

the test track, this is most likely due to the training track being much more varied and difficult as it includes sharper turns.

Table 6. Closed-loop evaluation metrics of model **BASE** on test data collected from the training track and a previously unseen track.

Track	Interventions	MAE [Meters]	RMSE [Meters]	Whiteness [Degrees / Second]	Failure Rate %
Training	1	0.25	0.33	0.77	0.28
Test	0	0.27	0.37	1.08	1.91

Closed-loop metrics for the **BASE** model on both tracks show also slightly worse performance on the previously unseen test track, with the exception of having no interventions, while an intervention occurred on the training track (see Table 6). Counter to the open-loop evaluation, the closed-loop whiteness of steering angles was slightly worse on the test track than on the training track. Additionally, the model deviated more on the test track than on the training track, echoed by their respective failure rates.

4.3 Overfitted model

Overfitted models usually perform well on distributions similar to the training data and have trouble generalizing to new data from different distributions. To mitigate this, the training data is diversified as much as possible to learn only the most general and robust features. However, these general features might be difficult to learn and may not be as useful. To test if overfitting is a reasonable way to get better results on a particular track than with varied training data, a model trained only on data from the training track (**SINGLE-TRACK**) is compared to the **BASE** model trained on same track and some other additional tracks, hence, with more training data. Since around 67% of the training data was collected from the training track, **SINGLE-TRACK** had around 1.4 times less training data. Both models are then evaluated on the same training track.

Table 7. Open-loop evaluation metrics on test data collected from the training track.

Model	RMSE [Degrees]	MAE [Degrees]	Whiteness [Degrees / Second]	Maximum error [Degrees]
BASE	7.44	6.29	64.25	28.80
SINGLE-TRACK	9.12	8.19	54.37	24.75

Table 8. Open-loop evaluation metrics on test data collected from the test track.

Model	RMSE [Degrees]	MAE [Degrees]	Whiteness [Degrees / Second]	Maximum error [Degrees]
BASE	7.85	6.48	61.44	24.55
SINGLE-TRACK	8.64	7.53	56.63	18.56

Based on the open-loop metrics (see Tables 7 and 8), the performance of **SINGLE-TRACK** is slightly worse on the training set than that of **BASE** in terms of MAE and RMSE, even though **SINGLE-TRACK** has better whiteness and maximum error results.

Table 9. Closed-loop evaluation metrics on test data collected from the training track.

Model	Interventions	MAE [Meters]	RMSE [Meters]	Whiteness [Degrees / Second]	Failure Rate %
BASE	1	0.25	0.33	0.77	0.28
SINGLE-TRACK	6	0.48	0.58	1.08	8.03

The closed-loop metrics (Table 9) seem to corroborate with the open-loop metrics, suggesting a much worse performance of **SINGLE-TRACK** compared to **BASE**. It is likely that the addition of training data from other roads provided the data diversity needed to learn more robust features. It is also plausible to conclude that these robust features are not much more difficult to learn than simpler features.

4.4 Intervention data

This experiment focuses on seeing the effects of using intervention data. A model (**NO-INT**) trained without intervention data is evaluated and compared to **BASE**. Since learning from intervention data helps mitigate the problem of distribution shift caused by pure behavior cloning, it is expected that the model trained with intervention data performs better on closed-loop metrics than a model trained without intervention data. This experiment aims to quantify the effects of training with and without intervention data. It should also be noted that intervention data accounts for roughly 37% of the total training data, which means **NO-INT** is trained on a noticeably smaller dataset.

Table 10. Open-loop evaluation metrics on test data collected from the training track.

Model	RMSE [Degrees]	MAE [Degrees]	Whiteness [Degrees / Second]	Maximum error [Degrees]
NO-INT	2.38	1.81	13.51	14.27
BASE	7.44	6.29	64.25	28.80

Table 11. Open-loop evaluation metrics on test data collected from the test track.

Model	RMSE [Degrees]	MAE [Degrees]	Whiteness [Degrees / Second]	Maximum error [Degrees]
NO-INT	2.11	1.67	12.30	7.09
BASE	7.85	6.48	61.44	24.55

NO-INT seems to perform significantly better on all open-loop metrics compared to **BASE** (see Tables 10, 11). This is somewhat expected as intervention recovery signals tend to be large in magnitude and the network trained on intervention data learns to imitate these large outputs more often. Thus the **NO-INT** model produces angles closer to 0, which corresponds to large sections of the ground truth angles.

Table 12. Closed-loop evaluation metrics on test data collected from the training track.

Model	Interventions	MAE [Meters]	RMSE [Meters]	Whiteness [Degrees / Second]	Failure Rate %
NO-INT	2	0.23	0.30	0.54	0.42
BASE	1	0.25	0.33	0.77	0.28

In contrast to the open-loop metrics, the closed-loop metrics show very similar performances by both models, with the main tie-breaker being the number of interventions (see Table 12). **NO-INT** failed in the sharpest turn of the track both ways. This suggests that the model encounters a distribution shift when taking sharp curves.

4.5 Input and output modalities

Many works related to E2E driving do something more than just predict the steering angle based on the current input image. Often they incorporate temporal data into their models or predict a future trajectory. In this experiment, different models with different input and output modalities are trained and evaluated, all sharing the same training and

testing data.

The first model keeps the default network architecture but is trained on difference images, where the previous frame is subtracted from the current frame. Since the input image includes a depth channel, a difference image would include a speed channel, which might give the network additional information, such as how far to look given the current speed. We refer to this model as **DIFF**.

The second model has ordinary input images but instead of just predicting the current steering angle, also predicts two additional steering angles, 1 second and 2 seconds into the future respectively. Since the data is recorded at 10 Hz, the future steering angles can be found from sequential data by taking the steering angles 10 and 20 future samples ahead respectively. The thinking is that forcing it to predict the future steering angles in addition to the current steering angle will force the network to learn more robust road curvature features. This model will be referred to as **FUT**.

The third model, which will combine both by taking difference images as input and outputting the 3 future steering angles, will be referred to as **DIFF+FUT**.

There is also the question as to how important each of the three input channels (ambience, intensity, and range) are for E2E driving. To test this, a new model was trained with each possible combination of LiDAR image channels as inputs. First, a new model is trained using for each individual channel (ambience, intensity, range) using only that channel image as input. These models are respectively referred to as **AMB**, **INT**, and **RNG**. Additional 2-channel models were also trained to show which combinations of 2 image channels are the most useful. These models include **RNG+AMB**, **RNG+INT**, and **INT+AMB**.

Table 13. Open-loop evaluation metrics on test data collected from the training track.

Model	RMSE [Degrees]	MAE [Degrees]	Whiteness [Degrees / Second]	Maximum error [Degrees]
INT+AMB	3.66	2.83	29.54	20.31
FUT	4.83	3.68	46.78	19.00
RNG+INT	4.95	4.01	32.99	17.25
FUT+DIFF	6.26	5.23	54.74	28.35
INT	6.24	5.28	40.19	22.73
DIFF	6.36	5.05	48.67	30.52
AMB	6.92	5.77	65.76	22.34
RNG	6.92	6.05	70.51	24.27
BASE	7.44	6.29	64.25	28.80
RNG+AMB	7.48	6.27	69.39	22.39

Table 14. Open-loop evaluation metrics on test data collected from the test track.

Model	RMSE [Degrees]	MAE [Degrees]	Whiteness [Degrees / Second]	Maximum error [Degrees]
INT+AMB	3.77	2.78	30.27	15.65
FUT	4.38	3.29	42.34	18.51
RNG+INT	4.69	3.72	31.73	16.66
DIFF	5.46	4.34	45.38	19.79
INT	6.01	5.06	38.97	18.08
FUT+DIFF	6.15	5.10	49.39	19.49
RNG+AMB	6.62	5.50	61.86	21.33
RNG	6.95	5.97	67.66	19.74
AMB	7.09	5.92	62.69	20.93
BASE	7.85	6.48	61.44	24.55

The open-loop metrics (Tables 13, 14) seem to suggest that **FUT** performs better than both **DIFF** and **FUT+DIFF**. When comparing models using subsets of the 3 LiDAR channels, the open-loop metrics seem to favor models making use if the intensity image with **INT+AMB** showing the best performance while **RNG+AMB** showing the worst performance. It is interesting to note that having additional channels to just intensity increases open-loop performance while combining range and ambience provides no real added benefit compared to using the two channels separately. Surprisingly, almost all models show an improvement over the **BASE** model in open-loop metrics.

Table 15. Closed-loop evaluation metrics on test data collected from the training track. Both **DIFF** and **FUT+DIFF** were excluded from closed-loop evaluation due to extremely poor real-life performance.

Model	Interventions	MAE [Meters]	RMSE [Meters]	Whiteness [Degrees / Second]	Failure Rate %
RNG+AMB	3	0.24	0.32	1.91	0.67
BASE	1	0.25	0.33	0.77	0.28
INT+AMB	1	0.28	0.36	0.80	0.78
AMB	4	0.28	0.38	3.13	1.04
INT	1	0.29	0.37	1.07	1.21
RNG+INT	0	0.31	0.42	0.87	3.44
FUT	0	0.31	0.41	0.71	2.72
RNG	0	0.33	0.42	2.29	2.41

The closed-loop metrics (Table 15) however paint a very different picture of the

results. On one hand, there seems to be a relatively high correlation between MAE, RMSE and failure rate, while on the other, a correlation between whiteness and number of interventions can be seen. However, there seems to be no clear correlation between these two groups. The correlation between whiteness and the number of interventions makes sense, as a larger whiteness value means the steering wheel turns more aggressively and more often, leading to more opportunities to deviate from the path. None of the open-loop metrics seem to correlate to the closed-loop results. It seems that overall, the best model in this experiment is **BASE**. While **RNG+INT**, **FUT**, and **RNG** had no interventions, they all seem to have their issues. **RNG** has a relatively high whiteness value (similarly to **AMB**), which translated to uncomfortable swerving in real life.

It is likely that **RNG** has difficulty finding important road features from the range image, such as track marks in different parts of the road and can only really distinguish road boundaries, which usually vary in height compared to the road. Seeing only road boundaries means that **RNG** was able to stay on the road but was not able to drive smoothly without oscillating between the left and right road boundaries.

AMB seems to have had the noisiest predictions from these models and also consequently had the most interventions. Since the ambience image is essentially a grayscale image from an infrared camera, the **AMB** model is the closest model imitating a regular camera-based E2E driving model. In fact, many camera-based E2E lane-following solutions first convert their input image into grayscale to lessen the number of parameters of the network and to avoid spurious correlation from different color channels [14]. Even though road boundaries and tyre tracks are clearly visible in the ambience image, the worse performance of **AMB** compared to other models utilizing different channels seems to suggest that features learned from just ambience images are not robust enough to drive with different lighting or weather conditions.

INT seemed to have no obvious problems aside from a single intervention, which happened in the same place where **BASE** experienced its intervention due to tracks leading off the road. This suggests that road tracks from other vehicles are more distinguishable from the rest of the road in the intensity channel and that **INT** and **BASE** models place high importance on these tracks. It also gives credence to the hypothesis that the intensity image gives the most useful features to **BASE**.

RNG+INT, while better than **RNG** and similar to **INT** in terms of driving smoothness, has surprisingly the highest failure rate. It is unclear what caused the high failure rate for **RNG+INT**, as **INT** using just the intensity image had a lower failure rate.

INT+AMB seems to be very close to the performance of **BASE** in terms of all metrics. This is somewhat interesting as it suggests that the range image gets overshadowed by the intensity and ambience images in terms of importance to the network. This could be simply because the most clearly visible features occur in only ambience and intensity channels.

While **RNG+AMB** kept closer to the expert trajectory most of the time, even more

than **BASE**, the relatively large whiteness score shows that the steering wheel was actively twitching left and right by slight amounts. This might make sense as models trained on range and ambience channels individually also had a high steering whiteness, suggesting that it is difficult to discern a comfortable path following steering angle from either of these images.

FUT also has a relatively high failure rate, but might be the best from the three models without any interventions. In fact, it is the only model that has lower steering whiteness than **BASE**. However, the difference is small and could be argued to be negligible and prone measurement errors. Thus, it does not seem to be better than **BASE** in any obvious way.

It is also worth noting that neither **DIFF** nor **FUT+DIFF** were evaluated with closed-loop metrics, as they had extremely poor performance in real-life and kept driving off the road constantly. Both **DIFF** and **FUT+DIFF** seemed slightly unstable for a few seconds after engaging the car, but as time moved on, the networks started to turn more and more off-road. It seemed as though there was some kind of steering feedback loop where large steering angles produced even larger steering angles in that direction. It is possible that the network learned to predict the steering angle directly from the difference images by looking at how the direction of the car changed over time. This is known as causal confusion and has been documented in literature before [11].

4.6 Visualizing what the network focuses on

To evaluate visually what the network learned, the VisualBackProp method was used to find the relative importance of pixels. This method can also provide insight as to what kind of visual features the convolutional layers are tuned for.

From Figure 14, it can be seen that the network pays close attention to the right side of the road. This is probably due to the fact that the safety driver always kept to the right side of the road and at a fixed distance from the right edge, even as the width of the road varied greatly.

Figures 15 and 16 show that highly contrasting lines and edges seem to activate the convolutional filters the most, indicating that the most useful features for predicting steering angle includes clearly visible lines. It should be noted that although the VisualBackProp images seem to suggest that the model gets confused by seeing visible lines outside of the road boundaries, the model does not seem to be affected by them in real-life. This could be explained by the fact that the deeper fully-connected layers learn to simply ignore strong activations coming from outside the usual road area. The fact that VisualBackProp does not take into account how the fully-connected layers behave limits its ability to be used as a debugging tool to see what the network mistook for a road feature. In fact, looking at images produced by VisualBackProp just before an intervention does not seem to give useful insights into what may have caused it, as images long before the intervention produce very similar outputs. Thus, while a useful

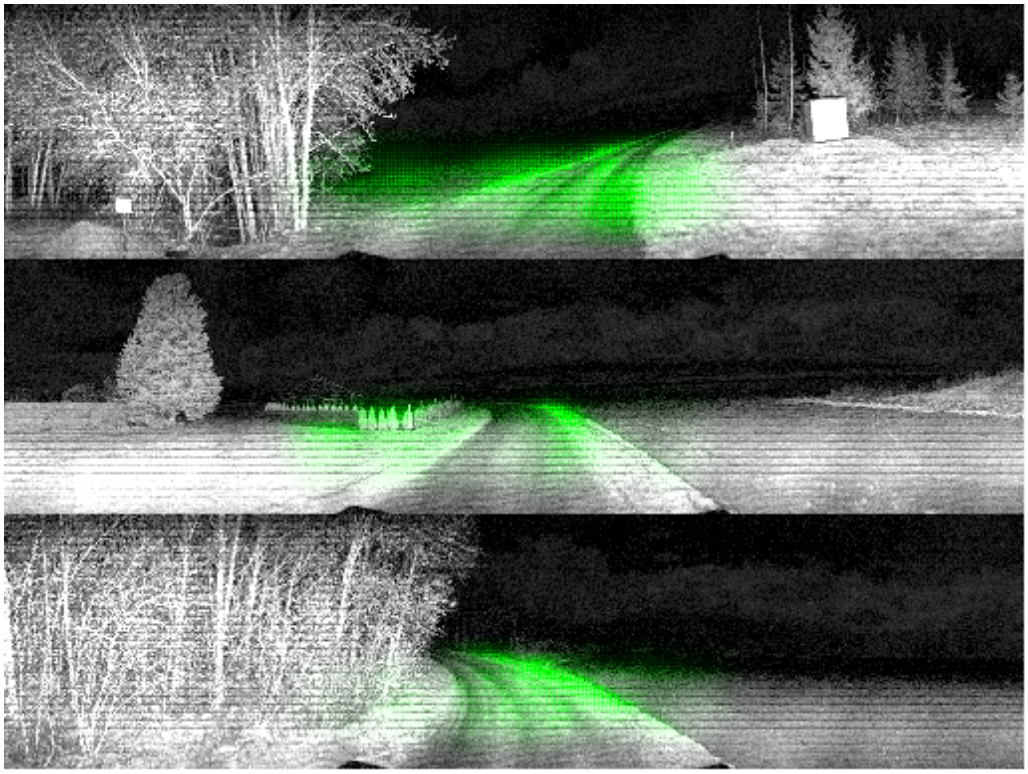


Figure 14. Images produced by the VisualBackProp method. The green saliency mask is added on the intensity channel of the LiDAR image. **BASE** model was the network examined.

tool, VisualBackrop is no silver bullet when it comes to figuring out why the network made the decision it did.

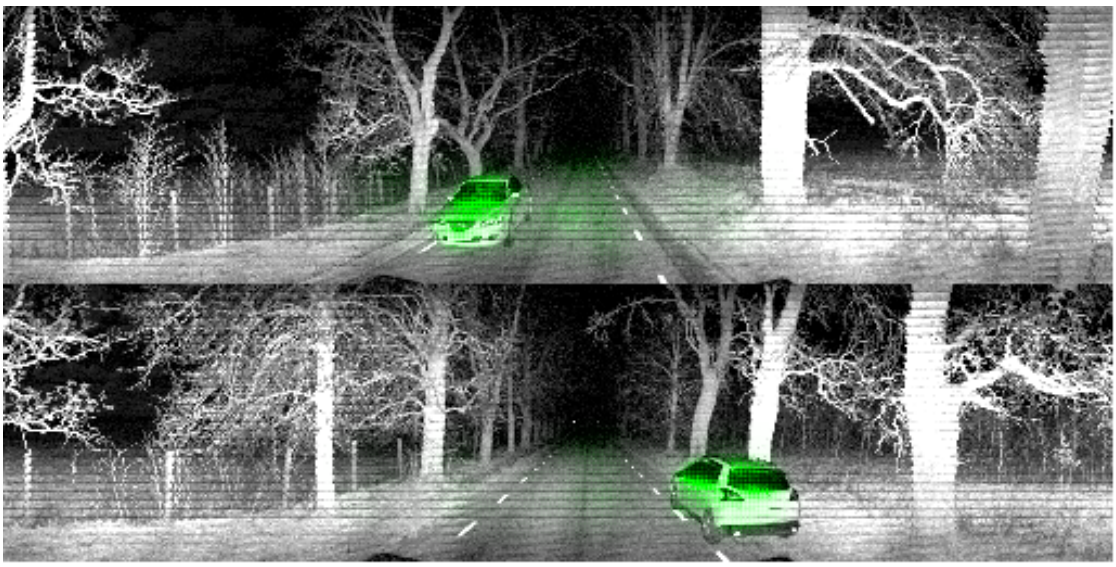


Figure 15. Images produced by the VisualBackProp method on images from the dataset containing cars. The green saliency mask is added on the intensity channel of the LiDAR image. **BASE** model was the network examined.

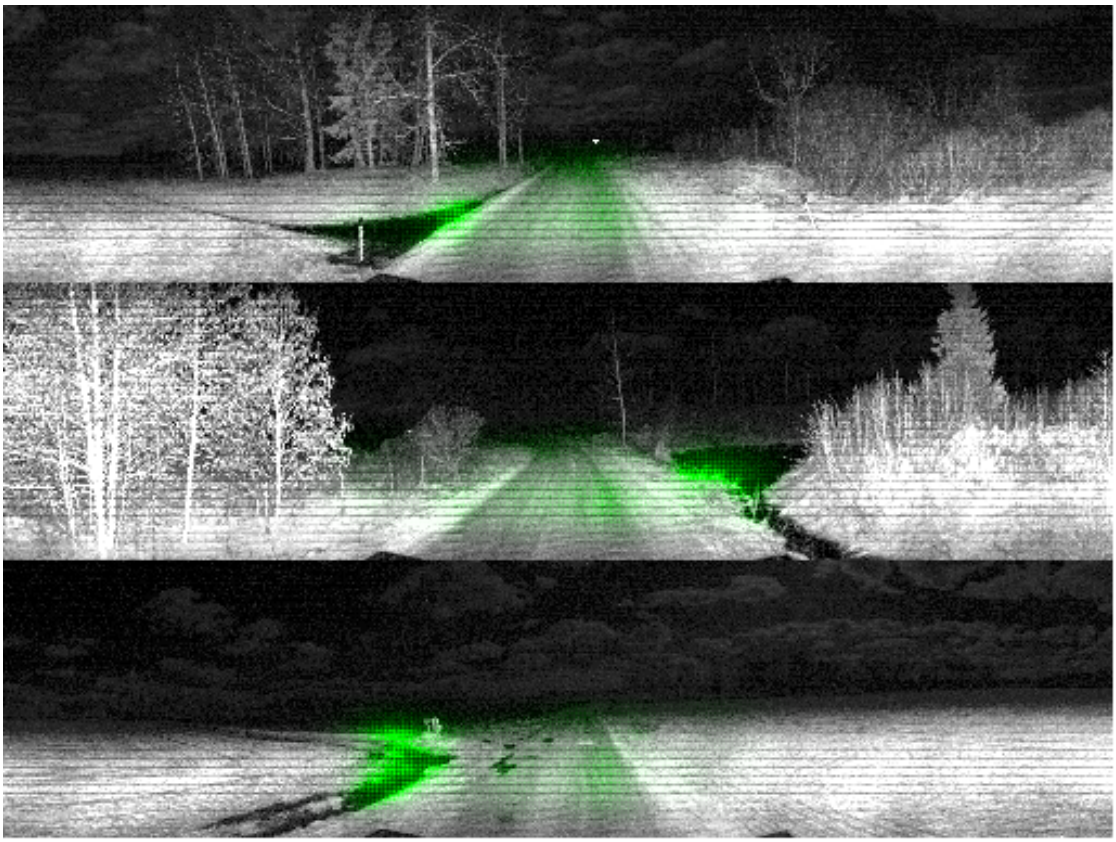


Figure 16. Images produced by the VisualBackProp method on images from the dataset containing bodies of water. The green saliency mask is added on the intensity channel of the LiDAR image. **BASE** model was the network examined.

5 Discussion

Initial results of the models evaluated on closed-loop metrics seem to suggest that using LiDAR as a camera, as done in this thesis, is a reasonable approach to E2E autonomous driving. In fact, based on experimenting with different channel configurations, the LiDAR specific intensity image may be more useful for E2E road following than grayscale camera images, or maybe even color camera images in general, as they are less prone to changing weather and lightning conditions.

The models were evaluated with both open-loop and closed-loop metrics to highlight how well these metrics capture the actual performance of the model during driving. The results obtained in this thesis suggest that open-loop metrics simply can not be used to estimate closed-loop performance. Even open-loop steering whiteness failed to predict closed-loop steering whiteness, suggesting there is barely any correlation between the two. This suggests that none of the open-loop metrics used in this thesis should be used to compare model performances and that closed-loop testing either in real life or in a simulator is a must. This also begs the question whether it makes sense to even train the network to optimize for the simple open-loop metrics, such as MAE or MSE and instead use alternative methods such as reinforcement learning to directly optimize for closed-loop metrics.

One of the experiments which measured the generalization ability of the network seems to suggest that the the model is able to drive on a previously unseen test road given, that the road conditions and surroundings look somewhat similar to what was seen in the training data. However, the results may also suggest that the test track was simply too easy without interesting confusing elements or extremely sharp turns.

Another experiment which involved examining how well a model trained only on a single track drives on that track compared to a model trained on more data from other tracks as well. The results show that the overfitted model had a much lower performance. These results are somewhat interesting as they suggest that an overfitted model underperforms compared to a model with more varied data in the context of E2E driving. However, this does not mean that a model trained with varied data is always going to be better, as fine-tuning on data from a specific track may yield better result than a model trained with just varied data. This type of transfer learning for E2E driving could be a topic of future research.

When comparing models trained with and without intervention data, the results show that the model trained without intervention data fails in sharp curves, while other closed-loop metrics seem relatively similar to a model trained with interventions data. It is possible that due to delays in the system from producing an inference by the network to actuating the steering wheel, the model will quickly diverge from the expert driving distribution. Another option is that small lateral shifts in the curve alter the distribution more heavily than lateral shifts while driving straight. It is also possible that the result is the interplay between these two phenomena. In any case, intervention data seems to

be useful. A possible future research topic involves comparing how much intervention data provides the best increase in performance and if there is a more principled way of collecting it, which could produce better data.

To allow for the model to make use of temporal data, a model trained only on difference images was compared to a model trained on the ordinary images. The open-loop metrics for the model only trained on difference images seem to suggest that it is somewhat good compared to many other models. However, closed-loop evaluation of this model was impossible as the network kept trying to drive off the road too often, possibly due to the previously mentioned causal confusion. This is not a problem with difference images specifically and applies to any kind of E2E driving model that has access to temporal data. It is noteworthy that many works on E2E driving improve their open-loop performance simply by introducing temporal data in the form of difference images [2], optical flow [13], or recurrent neural networks [12] without evaluating them in a simulation or on a real-world test vehicle.

A model using difference images and predicting future steering angles was also trained. However, it proved to have the same problem of predicting larger and larger steering angles based on the previous steering angle. While one might think that predicting into the future enforces learning the correct causality (future steering angles should be more dependent on the future road curvature and not on the current steering angle), it did not seem to help much. One possibility is that the network was simply unable to learn to predict into the future based on road curvature and instead predicted future steering angles based on the current steering angle. This might be avoided by predicting further into the future than 1 and 2 seconds, which was used in this thesis.

Using ordinary images and predicting into the future did not improve the driving performance, which suggests that the model did not learn to predict future steering angles by looking further ahead, but by looking at the current steering angle. However, there is still doubt if predicting future steering angles from a single image is a good thing to do. One could argue that if the future steering angle labels are taken by looking at sequential dataset samples a number of steps ahead, it makes the network implicitly assume a fixed speed as seen in the dataset. In reality, the dataset was collected with varying speeds and how far the model should look at the road to predict future steering angles is unclear. Therefore any future steering angle predictions should either assume a fixed speed or have temporal data as input. By using temporal data and enforcing the model to learn the correct causality by predicting future steering angles, the model should be able to deduce its current velocity and decide how far it should look to calculate future steering angles. Neither future prediction nor temporal data would work well alone and should be combined.

Finally, experiments were also conducted to measure the relative importance or usefulness of each of the three LiDAR channels: intensity, ambience and range. The

[2] https://github.com/tanelp/self-driving-convnet

results seems to suggest that the intensity image offers the most important features as it can detect tyre tracks and other visible lines on the road that can be used to more easily follow the path. While ambience seems to be the second most important channel in terms of clearly visible features, the features learned from ambience images do not seem to be as robust as features learned from other channels, explaining the poor performance of the model trained only on ambient images. Finally, the range channel seems to be able to detect road boundaries but not the track lines on the road, which seem to be important for stable driving. A model trained only on range images produced uncomfortable oscillations as it swerved between the left and right edges of the road.

After analyzing images produced with the VisualBackProp methods, it is clear that it can give insight into what kind of features the convolutional layers learned and what the networks generally focus on. It turns out the network is looking for clearly visible edges, which are most often found in the intensity image. The VisualBackProp method also showed cases where the network seems to be focusing on completely unrelated things such as bodies of water, where it finds the edges around the body of water to be very important. However, the model never reacts to these bodies of water in any strange way. This is probably because the fully-connected layers after the convolutional layers have learned to ignore activations coming in from places outside of the usual road area. However, it suggests that a better method might be needed, one that could also account for the fully-connected layers. This is a problem because it can not be used to actually debug the network and see which of the important pixels actually force the network to veer of the path and cause an intervention. It is also possible that the fully connected layers weigh some regions of the road section more than others, making debugging with VisualBackProp even more difficult. In the end, it should probably not be treated as a debugging tool but as a way to see what kind of features the convolutional layers have learned in general.

6 Conclusion

In this thesis, the feasibility of using LiDAR as a camera for E2E driving was examined. This was done by training a CNN to predict steering angles from LiDAR images. For this, a new dataset of Ouster LiDAR images and their corresponding steering angles was collected. In addition, different experiments were performed relating to generalization ability, input and output modalities, overfitting, and the importance of intervention data. The resulting models were evaluated with both offline open-loop metrics and online closed-loop metrics. Closed-loop evaluation was performed by letting the models drive a track both forwards and backwards. The final results were then analyzed and conclusions presented.

The results from the generalization experiment suggest that the trained model is able to drive without issues on new roads given the road and lightning conditions are similar in appearance to the training data. The experiment examining the importance of intervention data suggests that intervention data is needed to successfully take sharp turns. Experiments with overfitting suggest that more varied data from different roads increases performance significantly both on open-loop and closed-loop metrics. A comparison of using different subsets of LiDAR images as input (instead of the default 3-channels) showed that the intensity image proved to be the most useful for the network for predicting smooth steering angles. In addition, models trained on difference images showed good open-loop performance while very bad closed-loop performance. Finally, a model trained to predict future steering angles in addition to the current steering angle showed no real improvements over the base model. From all of the evaluation metrics from different experiments, it can be concluded that open-loop metrics can not predict closed-loop performance.

An obvious direction for future work includes training with a bigger and more advanced CNN architecture. A larger dataset of more varied roads is also needed to increase model performance on all roads. To mitigate the issue of slow evaluation, a realistic data-driven simulator should be built to evaluate the true performance of the model as quickly as possible. In addition, predicting future trajectories instead of the final steering angle would improve generalization of this method to vehicles of different sizes.

References

[1] AutonomouStuff. Astuff autoware kit. https://autonomoustuff.com/products/astuff-autoware-kit. Accessed: 26.04.21.

[2] AutonomouStuff. Astuff speed and steering control software. https://autonomoustuff.com/products/astuff-speed-steering-control-software. Accessed: 25.04.21.

[3] AutonomouStuff. Pacmod by-wire kits. https://autonomoustuff.com/products/pacmod. Accessed: 25.04.21.

[4] Mariusz Bojarski, Chenyi Chen, Joyjit Daw, Alperen Değirmenci, Joya Deri, Bernhard Firner, Beat Flepp, Sachin Gogri, Jesse Hong, Lawrence Jackel, Zhenhua Jia, BJ Lee, Bo Liu, Fei Liu, Urs Muller, Samuel Payne, Nischal Kota Nagendra Prasad, Artem Provodin, John Roach, Timur Rvachov, Neha Tadimeti, Jesper van Engelen, Haiguang Wen, Eric Yang, and Zongyi Yang. The nvidia pilotnet experiments, 2020.

[5] Mariusz Bojarski, Anna Choromanska, Krzysztof Choromanski, Bernhard Firner, Larry D. Jackel, Urs Muller, and Karol Zieba. Visualbackprop: visualizing cnns for autonomous driving. *CoRR*, abs/1611.05418, 2016.

[6] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016.

[7] Mariusz Bojarski, Philip Yeres, Anna Choromanska, Krzysztof Choromanski, Bernhard Firner, Lawrence D. Jackel, and Urs Muller. Explaining how a deep neural network trained with end-to-end learning steers a car. *CoRR*, abs/1704.07911, 2017.

[8] Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krähenbühl. Learning by cheating. *CoRR*, abs/1912.12294, 2019.

[9] Felipe Codevilla, Antonio M. López, Vladlen Koltun, and Alexey Dosovitskiy. On offline evaluation of vision-based driving models. *CoRR*, abs/1809.04843, 2018.

[10] Comma.ai. End-to-end lateral planning. https://blog.comma.ai/end-to-end-lateral-planning/, 2021. Accessed: 19.04.21.

[11] Pim de Haan, Dinesh Jayaraman, and Sergey Levine. Causal confusion in imitation learning. *CoRR*, abs/1905.11979, 2019.

[12] Hesham M. Eraqi, Mohamed N. Moustafa, and Jens Honer. End-to-end deep learning for steering autonomous vehicles considering temporal dependencies. *CoRR*, abs/1710.03804, 2017.

[13] Nelson Fernández. Two-stream convolutional networks for end-to-end learning of self-driving cars. *CoRR*, abs/1811.05785, 2018.

[14] Jonas Heylen, Seppe Iven, Bert De Brabandere, Jose Oramas M., Luc Van Gool, and Tinne Tuytelaars. From pixels to actions: Learning to drive a car with deep neural networks. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 606–615, 2018.

[15] Judy Hoffman, Eric Tzeng, Taesung Park, Jun-Yan Zhu, Phillip Isola, Kate Saenko, Alexei A. Efros, and Trevor Darrell. Cycada: Cycle-consistent adversarial domain adaptation. *CoRR*, abs/1711.03213, 2017.

[16] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[17] Stéphane Lathuilière, Pablo Mesejo, Xavier Alameda-Pineda, and Radu Horaud. A comprehensive analysis of deep regression. *CoRR*, abs/1803.08450, 2018.

[18] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. Pytorch distributed: Experiences on accelerating data parallel training, 2020.

[19] Zoltán Lőrincz. A brief overview of imitation learning. `https://smartlabai.medium.com/a-brief-overview-of-imitation-learning-8a8a75c44a9c`, 2019. Accessed: 19.04.21.

[20] Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. *CoRR*, abs/1412.0035, 2014.

[21] Noel Martignoni. Ros-master-node-topic. `https://commons.wikimedia.org/wiki/File:ROS-master-node-topic.png`, 2016. Accessed: 25.04.21.

[22] NOAA. What is lidar? `https://oceanservice.noaa.gov/facts/lidar.html`, 2021. Accessed: 19.04.21.

[23] NVIDIA. Nvidia tensorrt. `https://developer.nvidia.com/tensorrt`. Accessed: 25.04.21.

[24] Ouster. High-resolution os1 lidar sensor: robotics, trucking, mapping. `https://ouster.com/products/os1-lidar-sensor/`. Accessed: 19.04.21.

[25] Angus Pacala. How multi-beam flash lidar works. https://ouster.com/blog/how-multi-beam-flash-lidar-works/, 2018. Accessed: 19.04.21.

[26] Angus Pacala. Lidar as a camera - digital lidar's implications for computer vision. https://ouster.com/blog/the-camera-is-in-the-lidar/, 2018. Accessed: 19.04.21.

[27] Angus Pacala. Introducing the os1-128 lidar sensor. https://ouster.com/blog/introducing-the-os-1-128-lidar-sensor/, 2019. Accessed: 26.04.21.

[28] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA ARTIFICIAL INTELLIGENCE AND PSYCHOLOGY . . . , 1989.

[29] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *CoRR*, abs/1612.00593, 2016.

[30] Zhuwei Qin, Fuxun Yu, Chenchen Liu, and Xiang Chen. How convolutional neural network see the world - A survey of convolutional neural network visualization methods. *CoRR*, abs/1804.11191, 2018.

[31] ROS. Ros/introduction. http://wiki.ros.org/ROS/Introduction. Accessed: 25.04.21.

[32] Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. No-regret reductions for imitation learning and structured prediction. *CoRR*, abs/1011.0686, 2010.

[33] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization?, 2019.

[34] Ramprasaath R. Selvaraju, Abhishek Das, Ramakrishna Vedantam, Michael Cogswell, Devi Parikh, and Dhruv Batra. Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization. *CoRR*, abs/1610.02391, 2016.

[35] Connor Shorten and Taghi M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1), jul 2019.

[36] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps, 2013.

[37] Ibrahim Sobh, Loay Amin, Sherif Abdelkarim, Khaled Elmadawy, Mahmoud Saeed, Omar Abdeltawab, Mostafa Gamal, and Ahmad El Sallab. End-to-end multi-modal sensors fusion system for urban automated driving. 2018.

[38] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net, 2015.

[39] Ardi Tampuu, Tambet Matiisen, Maksym Semikin, Dmytro Fishman, and Naveed Muhammad. A survey of end-to-end driving: Architectures and training methods. *IEEE Transactions on Neural Networks and Learning Systems*, page 1–21, 2020.

[40] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013.

[41] Quanshi Zhang and Song-Chun Zhu. Visual interpretability for deep learning: a survey. *CoRR*, abs/1802.00614, 2018.

[42] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[43] Jinyun Zhou, Rui Wang, Xu Liu, Yifei Jiang, Shu Jiang, Jiaming Tao, Jinghao Miao, and Shiyu Song. Exploring imitation learning for autonomous driving with feedback synthesizer and differentiable rasterization, 2021.

Appendix

I. Glossary

Acronyms

AD autonomous driving. 5, 7

ALVINN Autonomous Land Vehicle In a Neural Network. 5

CAN Controller Area Network. 24, 25

CNN convolutional neural network. 5, 6, 11, 12, 44

CV computer vision. 11

DNN deep neural network. 5, 23

E2E end-to-end. 5–7, 10, 17, 26, 33, 34, 36, 41, 42, 44

FOV field of view. 10, 21–23

GPU graphics processing unit. 24

IL imitation learning. 7

INS inertial navigation system. 27

IPC inter-process communication. 15

LiDAR Light Detection And Ranging. 5, 6, 10, 11, 16, 26, 27, 34, 35, 38–42, 44

MAE mean absolute error. 23, 24, 26, 27, 29, 32, 36, 41

ML machine learning. 11

MLP multi-layer perceptron. 5

MPC model predictive controller. 8, 10

MSE mean squared error. 23, 41

ONNX Open Neural Network Exchange. 26

RL reinforcement learning. 7

RMSE root mean squared error. 26, 27, 29, 32, 36

ROS Robot Operating System. 15, 16, 24–26

RTK real-time kinematic positioning. 27

SGD stochastic gradient descent. 23

SSC Speed and Steering Control. 25, 26, 28, 30

II. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Jan Aare van Gent**,
*(*author's name)

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   Using LiDAR as Camera for End-to-End Driving,
   *(*title of thesis)

   supervised by Tambet Matiisen.
   *(*supervisor's name)

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Jan Aare van Gent
14.05.2021