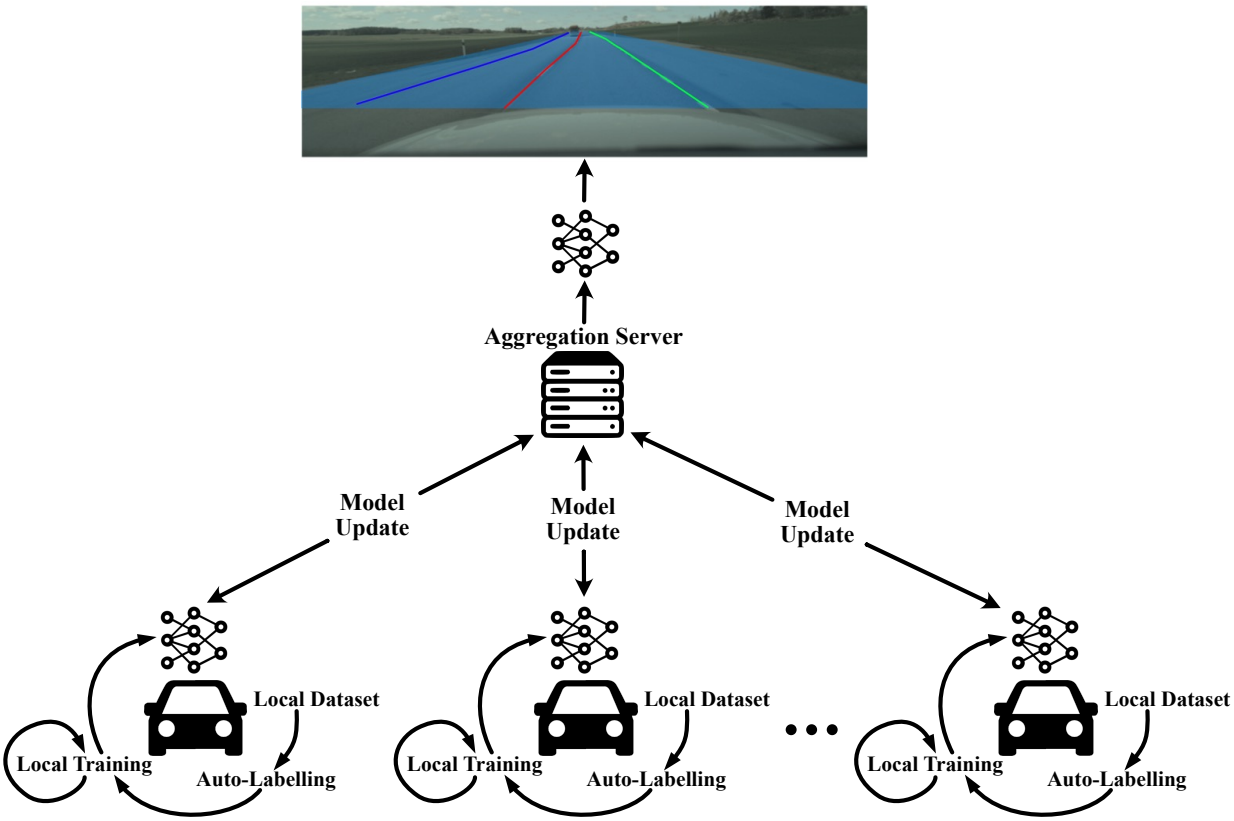
CHALMERS
UNIVERSITY OF TECHNOLOGY



Decentralized Training of 3D Lane Detection with Automatic Labeling Using HD Maps

Master's thesis in Data Science and AI

Yadong Mao & Zhuqi Xiao

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2022
www.chalmers.se

Master's thesis 2022

Decentralized Training of 3D Lane Detection with Automatic Labeling Using HD Maps

YADONG MAO & ZHUQI XIAO

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Signal Processing and Biomedical Engineering
Computer Vision and Medical Image Analysis Group
Chalmers University of Technology
Gothenburg, Sweden 2022

YADONG MAO
ZHUQI XIAO

Supervisor: Che-Tsung Lin, Electrical Engineering; Mina Alibeigi, Zenseact
Examiner: Christopher Zach, Electrical Engineering

Master's Thesis 2022
Department of Electrical Engineering
Division of Signal Processing and Biomedical Engineering
Computer Vision and Medical Image Analysis Group
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: An illustration of a fleet of cars learning a 3D lane detection model in a decentralized manner. Each car labels its unique and heterogeneous dataset and locally trains individual models that are later sent to a central server for aggregation to create a joint, global model.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2022

Decentralized Training of 3D Lane Detection with Automatic Labeling Using HD Maps
YADONG MAO & ZHUQI XIAO
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Image-based 3D lane detection is one of the critical foundations of many Advanced Driver Assistance Systems (ADAS) features. These have recently been extended to end-to-end deep learning models, which allow casting the 3D lane detection to a one-stage object detection problem. Training such a model requires vast amount of labeled 3D lanes data collected by cars driving worldwide to deliver the target performance. However, data collection and transfer are not straightforward due to the emerging data-sharing policies. Federated Learning (FL) enables the training of AI models across multiple decentralized edge devices while ensuring data privacy, but without access to the skilled human annotators at the edge makes it challenging to access superior annotations.

In this report, we introduce an automatic labeling pipeline using a pre-recorded HD map as the primary source to label the data collected in edge devices automatically. As references, a semi-automatic approach is applied for creating a 3D lanes dataset used as ground truth by combining manually annotated 2D lane images with depth maps from aggregated LiDAR point clouds. The results show that our auto-labeling pipeline can generate high-precision 3D lane marking annotations with parts being accurate up to more than 100 meters, which makes it possible to be employed to train a 3D lane detection model in a fully automatic way.

In addition to training the model using the semi-automatically and fully-automatically generated datasets in a centralized manner, we also simulate the decentralized training with two clients and one server. The decentralized training experiments investigate the impact on the performance of training a 3D lane detection model in decentralized way. In this report, we use the Flower framework to implement decentralized experiments. The clients' edge models are trained by datasets generated from semi-automatic and fully-automatic methods. The server's global model aggregates the edge model's parameters according to the FedAVG strategy. Finally, we found that for the same dataset, the average precision and F score (max) performance of the model obtained by federated learning is close to the model obtained by centralized learning. Although the model trained with the fully-automatic dataset performs slightly lower than the model trained with the semi-automatic data in the final result, we believe the model's performance will improve further as the dataset expands. This means we can train a model with excellent performance through federated learning while keeping the data locally on the edge devices.
Keywords: Federated Learning, Edge Learning, 3D Lane Detection, HD map, 3D Lane Annotations, Auto-Labeling, Deep Learning, Autonomous Vehicles, Object Detection, Autonomous Vehicles.

Acknowledgements

We would like to thank our supervisor Mina Alibeigi at Zenseact, who guided us throughout this project and contributed with her highly valuable expertise. We would also like to extend our gratitude to Che-Tsung Lin, our academic supervisor at Chalmers, for many rewarding discussions, not only about this project but also some life experiences.

Thanks to Zenseact for giving us this opportunity to work on such an interesting and challenging project. They provide us with the necessary data and tools for completing this thesis.

We would also like to thank our examiner at Chalmers, Christopher Zach.

Yadong & Zhuqi, Gothenburg, August 2022

Contents

List of Figures

List of Tables

1
Introduction

This chapter includes the background and aim of this thesis project, as well as the relevant delimitations.

1.1 Background

Over the past decade, autonomous driving has drawn considerable attention from both the industry and academia. There are many benefits that autonomous vehicles can bring for both safety and environmental concerns. Road accidents are currently a significant issue around the world. According to [1], during 2015, 35,092 people were killed in traffic accidents on U.S roadways, with the 7.2% increase being "the largest percentage increase in nearly 50 years". An analysis revealed that about 94% of those accidents were caused by human error, and the rest by the environment and mechanical failures [2]. The opportunity to reduce car accidents is making Advanced Driver Assistance Systems (ADAS) critical. Automatic emergency braking, pedestrian detection, surround view, parking assist, driver drowsiness detection and gaze detection are just some of the many ADAS applications that help drivers by providing safety-critical functions that leads to less crashes and consequently save lives.

To make some features in autonomous driving scenarios possible, such as lane-keeping assist, lane departure warning, and pilot assist, accurately perceiving and understanding the lane structure is one of the most critical steps. As described by [3], there are a number of perception modalities that have been extensively used in modern ADAS systems for road and lane understanding, including monocular vision (a single camera), stereo imaging (the use of two cameras enabling the perception of depth), radar, LiDAR (Light Detection and Ranging), vehicle dynamics information from the vehicle's odometer or inertial measurement unit (IMU), and global positioning information obtained using global positioning systems (GPS) and digital maps. Different modalities have their strengths and weaknesses. For instance, vision can perceive colours but is less able to do so in poor light conditions or perceive distance information; LiDAR can measure 3D structures around the vehicle independent of natural light conditions. Therefore, a fusion of data from multiple modalities can improve the overall system reliability.

Lane markings have been widely used as the primary identifiers of lane boundaries. Due to the fact that lane markings are designed for the human visual system, many frequently used lane detection modalities are based on visual perception. Therefore,

as one of the sensors closest to human eyes, cameras are the most frequently used for lane detection studies. In addition, following the same reasoning from an "evolutionary" point of view, it is also because the camera is currently the cheapest and most robust modality.

The majority of image-based lane detection methods treats lane detection as a 2D task. From these images, lane lines can be extracted using a feature-based approach, where low-level edge features are typically used for lane boundary detection [4]. One needs to increase the complexity of these designed features to make lane marking detection more robust, which may result in failure testing when the scenery changes and then may require additional sensors like LiDARs [5]. Therefore, this method cannot account for the vast variability of road scenarios. Ideally, it is best to learn the inherent patterns directly from the data. Deep learning (DL) is well known for this and can digest multiple patterns in data. Good evidence of this is its success in the ImageNet image classification challenge [6], where convolutional neural networks (CNNs) with deep architectures have vastly outperformed previous state-of-the-art solutions. Therefore, deep learning has been explored extensively for lane detection in recent years.

A typical DL-based 2D lane detection method first uses a semantic segmentation component to assign each pixel in an image with a class label indicating if it is a part of a lane, followed by a component to do lane model fitting so as to form lane instances [7], [8]. These detected 2D lane representations can be further transformed into a 3D approximation by the assumption of the flat earth. However, this assumption could lead to unexpected problems. For example, these detectors may give inaccurate perception results if a driving vehicle encounters a hilly road. Finding a way to circumvent these assumptions effectively has become the focus of many researchers on such tasks.

3D lane detection is a challenging task that has recently attracted increasing attention. Compared with 2D lane detection, image-based 3D lane detection facilitates the perception of a real 3D driving environment and is more flexible in handling complex road undulations. The latest trend [9]–[11] has started to perceive complex 3D lane structures using an end-to-end system, which overcomes the shortcomings associated with the flat earth assumption. Most of these methods rely on CNNs to extract features of the input images and further make predictions of the 3D lanes. Specifically, the latest state-of-the-art 3D-LaneNet [9] predicts 3D lanes from a single image, which shows promising results.

The success of these machine learning (ML) technologies, in particular deep learning (DL), has been fueled by the availability of vast amounts of data. Such systems usually require access to considerable human-labeled data to achieve a satisfactory level of performance. For instance, 3D-LaneNet [9] ended up performing very well thanks to the training on a synthetic dataset containing 300K images. However, this is just one case of a research project, a larger volume of data is usually required for a system with a commercially acceptable level of performance. A good example

is Facebook's object detection system which was reported to be trained using 3.5 billion images from Instagram [12]. An ideal scenario for improving the robustness of a lane detection system would be to have a constant stream of data from fleet customers. A traditional and naive approach is to collect and transfer the data to one central location where powerful computers can train and build ML models. Today, this methodology is no longer valid because our society realises the importance of user data privacy, and regulators are tightening the laws governing private data sharing. The European Union's General Data Protection Regulation (GDPR) is a prime example. Therefore, a solution that keeps the data where it resides and instead brings AI models to the data is needed. Federated Learning (FL) enables edge devices to collaboratively learn a shared prediction model while keeping all the training data on the device, decoupling the ability to do machine learning from the need to store the data in the central server by bringing model training to the device. FL allows for smarter models, lower latency, and less power consumption, all while ensuring privacy.

It could be very expensive to generate the ground truth of a real-world 3D lane dataset as it normally requires a complex multi-sensor setup. That is why both [9] and [10] developed their model using synthetic data. Even though it has been suggested in some literature like [13] that synthetic data can improve the performance of 3D lane detection models on real-world data, the improvement is necessarily limited. Therefore, finding ways to generate sufficient ground truth of real-world 3D lane data is still essential, which is even more important in the case of FL tasks since the data does not come into contact with human and therefore cannot be manually annotated. In [14], they adopt a way to generate lane markings by creating a high-definition map. It is not scalable since it requires a multi-sensor setup including LiDAR, radar, camera, odometry, and GPS to work synchronously to create the map. However, their approach is very instructive as they demonstrate that the HD map can be utilized to generate large labeled datasets of static objects.

1.2 Goal

This thesis project aims to exploit HD-map as our main source to annotate 3D lanes in monocular images with high enough quality to enable training 3D lane detection networks. Automatic annotation methods will apply to a real-world dataset, which will be applied to train a baseline model similar to 3D-LaneNet [9]. The goal is to let the baseline model get a similar performance in centralized and decentralized training on the dataset with automatically generated labels.

1.3 Delimitations

One of the critical requirements for a fully autonomous vehicle is that it can handle all the situations the car might encounter, which means that the vehicle must be able to handle driving in various road situations, e.g. highway, multiple lanes, splits, merges, crossings roundabouts, etc. In addition, different road conditions also need

to be considered by a real-world driver. However, in this thesis project, we will try to develop and evaluate all methods in only one or a few datasets that do not cover all possible scenarios. Hence, we do not expect to present a solution that could be deployed in the real world.

In this project, we run FL setting only in the simulation mode, meaning that we will not try the developed solution in the real federation model (i.e. on the hardware edge devices). However, we expect that the developed solution works similarly in the real federated mode as in the simulation mode since we will use an FL framework that supports both modes.

2
Theory

Some relevant background theory is needed to generate lane marking annotations using a High Definition (HD) map and train a 3D lane detection model using deep learning technology. The following sections will present theory regarding coordinate systems and transformations, the representation of an HD map, object detection using deep learning, federated learning, and lane detection.

2.1 Coordinate Systems

This thesis uses different coordinate systems, each corresponding to a different purpose, which will be explained in this section.

2.1.1 Global Geographical System

The Global Geographic System is a widely used coordinate system for describing the location of objects relative to the entire globe. In this coordinate frame, one position contains the values specifying the longitude, latitude, and altitude. Longitude is the degree describing a position on Earth that is east or west of a north-south alignment known as the Prime Meridian. The longitude of the Prime Meridian is 0°, and the longitude of other locations on Earth is either east to 180° or west to -180°. Latitude refers to the degrees from north to south on the Earth's surface, 90° from the equator to the north pole and -90° to the south pole. The altitude is an ellipsoid's height approximating the Earth's surface, and the WGS84 standard is used in this case. GPS is usually employed to track the vehicle in the global geographical system, which makes the records from GPS can be used to initialize the vehicle's positioning on the map.

2.1.2 East-North-Up System

The global geographical system can be converted into a local Cartesian system, in our case it is an East North Up (ENU) system, shown in Figure 2.1. This frame is attached to the vehicle, but dose not rotate with it, which can simplify the calculations. The East axis (E) is perpendicular to gravity, perpendicular to the north axis and is in the East direction. The North axis (N) is perpendicular to the gravity vector and in the direction of the north pole along the earth's surface. The Up axis (U) is co-axial with the gravity vector, and positive in the up direction.

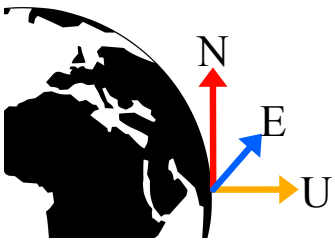


Figure 2.1: Schematic diagram of the east-north-up (ENU) coordinate system.

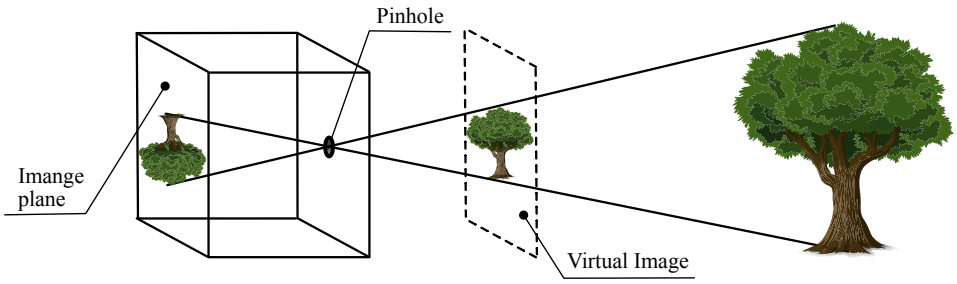
2.1.3 Ego-vehicle Frame

The data acquired by inertial sensors mounted on the vehicle is relative to the so-called ego-vehicle coordinate system. According to the standard ISO 8855 [15], the X-axis is perpendicular to gravity and is in the vehicle's forward direction, projected onto the horizontal plane. The Y-axis is perpendicular to gravity, perpendicular to the X-axis and is in the lateral direction of the vehicle, pointing to the left, projected onto the horizontal plane. The Z-axis is co-axial with the gravity vector and positive in the up direction. Thus, this frame is attached to the vehicle and rotates by the vehicle's heading but not with the roll and pitch of the vehicle.

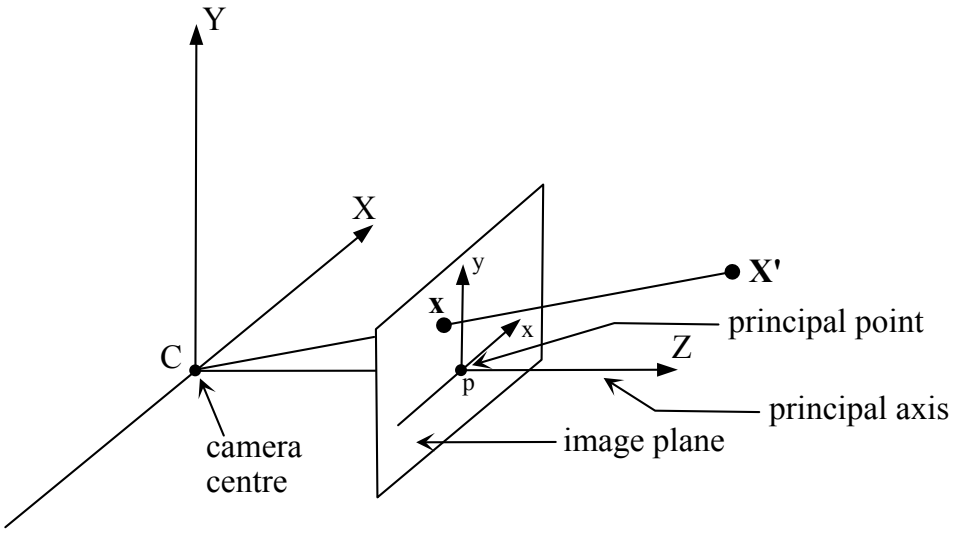
2.1.4 Camera Coordinate System

The most commonly used mathematical camera model is the so called pinhole camera. The camera has a box shape with the light from an object enters though a small hole (the pinhole) in the front and produces an image on the image plane. When representing the camera model like this, the image appears on the image plane as upside down, as can be seen in Figure 2.2a. To overcome this problem, we create a mathematical model called camera coordinate system, shown in Figure 2.2b.

The origin of the Euclidean coordinate system C represents the camera center (pinhole). p is the center of the image, named the principal point. To generate an image projection x on image plane of a scene point X', a line is formed between X' and C and intersect it with the image plane at x.



(a) The Pinhole camera model



(b) A mathematical model of camera coordinate system.

Figure 2.2: The Pinhole camera model and its mathematical representation.

2.1.5 Homogeneous Coordinates

In Figure 2.2b, X' is a vector representing a point in a three-dimensional coordinate system $\mathbb{R}^3$ and x is its projection in the projective space $\mathbb{P}^2$. In fact, x can represent all the points in $\mathbb{R}^3$ that lie on the line CX' except point C. Homogeneous coordinates are a system of coordinates used in projective geometry, which could be used when transforming between a three-dimensional coordinate system and a two-dimensional projective space.

To convert a vector from a Cartesian coordinate system to homogeneous coordinates, the Cartesian coordinates are expanded with a scaling factor 1 in the fourth dimension. In general, the homogeneous coordinates for representing $\mathbb{P}^n$ are the vectors of $\mathbb{R}^{n+1}$, and if coordinate $n+1$ is not zero, then we can interpret them as points of $\mathbb{R}^n$ by dividing with this coordinate. Given a point in $\mathbb{R}^3$, it can be

converted to homogeneous coordinates as:

$$v = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \Rightarrow v_h = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{2.1}$$

where v_h is the point v in $\mathbb{R}^3$ expressed in homogeneous coordinates. Then all points $\lambda v_h (\lambda \neq 0)$ represent the same point in $\mathbb{P}^2$.

The extrinsic camera parameters, which defines the position and rotation of the camera relative to a known world reference, is used to transform points in 3D world coordinates to points in 3D camera coordinates. A general camera extrinsic is defined as

$$\begin{bmatrix} R_{3\times3} & T_{3\times1} \\ 0_{1\times3} & 1 \end{bmatrix}_{4\times4} \tag{2.2}$$

where R is the rotation matrix describing how to align the axes of the world coordinate and the camera coordinate, and T is the translation vector describing the relative positions of two origins in two coordinates.

The intrinsic matrix defines the internal properties of the camera, which is a transformation matrix that converts points from the camera coordinate system to the image coordinate system. The camera intrinsic matrix, K, is defined as:

$$\begin{bmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

where (f_x, f_y) is the focal length (in pixels), (c_x, c_y) is the principal point (in pixels), and γ defines the skew coefficient. The intrinsic of a camera does not depend on the scene and only needs to be estimated once. The distortion coefficient is another internal property of the camera that is not described by the intrinsic matrix but instead defined by its own vector. Typically, cameras mounted on cars use fisheye lenses with some degree of optical distortion to get a more comprehensive view.

2.1.6 Top-view Projection

Inverse Perspective Mapping (IPM) is a homographic transformation that warps a perspective image to a birds-eye-view (BEV) image. In autonomous driving, IPM aids in several downstream tasks such as lane marking detection, path planning and intersection prediction solely from using a monocular camera as this orthographic view is scale-invariant. IPM first assumes the world to be flat on a plane. Then it maps all pixels from a given viewpoint onto this flat plane through homography projection.

See Figure 2.3 for an illustration of the following definitions. Two coordinate systems are shown, C_{cam} and C_{road}. C_{cam} is the camera coordinate system with y' as its

viewing direction. P_{road} is the plane tangent to the local road surface. In C_{road}, x and y is the projection of x' and y' onto P_{road}, and z is the normal to P_{road} represents the elevation of this specific position on road. A six degrees of freedom (DOF) of homographic matrix $\mathbf{T}$ is needed to project points between these two frames. If we assume zero camera roll, camera height h_{cam} and pitch θ can uniquely determine the homography, which can be obtained from the camera intrinsic and extrinsic parameters.

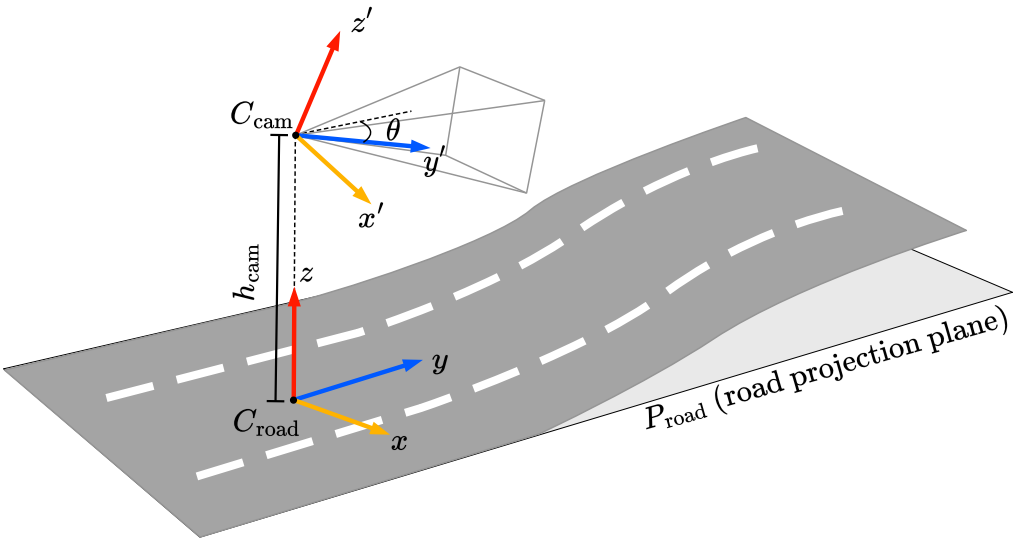


Figure 2.3: Camera position and road projection plane. The image is inspired by [9] and [10].

2.2 High Definition Map Representation

As one of the critical enabling technologies for autonomous driving, High Definition (HD) maps has become a major research focus in recent years [16]. An HD map is a highly accurate map containing details such as road shape, lane marking, traffic signs, barriers, etc. [17], which can be precise at a centimetre level [18]. HD maps are often captured using an array of sensors, such as LiDARs, radars, cameras, and GPS. Within the functional system architecture of an autonomous driving system, HD maps are tightly associated with localization functionality, interacting with the perception module and ultimately supporting the planning and control module [19], [20].

A typical HD map is composed of three layers, *road model*, *lane model*, and *localisation model*, and each layer is designed for different functionalities [16]. Road model in digital maps uses an ordered sequence of shape points describing the geometry of roads, which is used for strategic planning (navigation). Each road section is represented as a polyline and has its start and end nodes. The lane model is for

perception and tactical planning (guidance) that considers the current road and traffic conditions. Figure 2.4 demonstrates a road model on top of a lane model. The localisation model is used to localise the ego vehicle on the map.

Each road is usually divided into multiple road segments, each of which consists of one or multiple lanes and each lane has an imaginary centerline represented as a polyline. A polyline is described as a series of nodes connected by short line segments. Each node is defined by a tuple of longitude, latitude, and altitude in the global geographical system as described in section 2.1.1, thus making the polyline three-dimensional. Each centerline has its unique ID as an identifier, as do the lane markings on each side of the centerline. A typical centerline usually contains metadata such as lane type, speed limit, lane width, lane length, and the ID of the lane markers on each side. It should be noted that the lane length here is the length of the current lane in this specific road segment. Additional metadata in a centerline is its immediate topology, showing its connected lanes' ID (both its predecessor and successor) and neighbours' lanes if they exist. Figure 2.5 demonstrates an example of two road segments consist of multiple parallel two-way driving lanes.

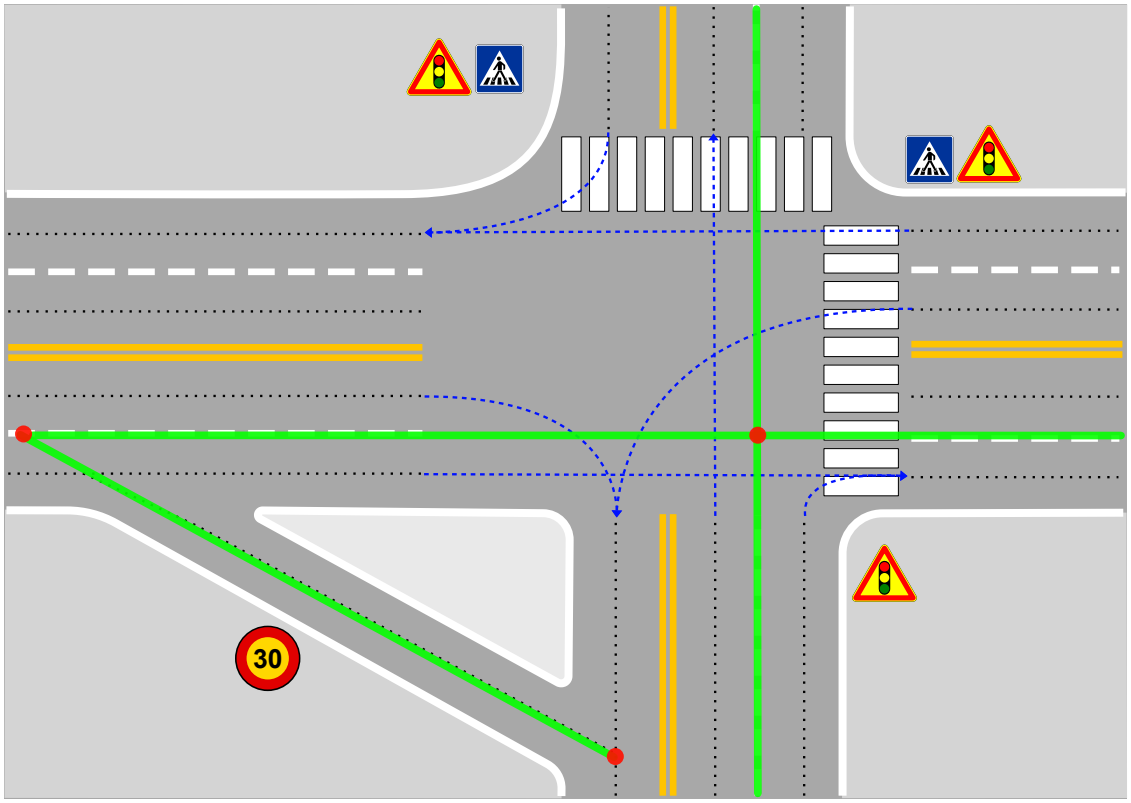


Figure 2.4: An example of a road model (green polylines and red nodes) on top of a lane model in HD map. The image is inspired by [16].

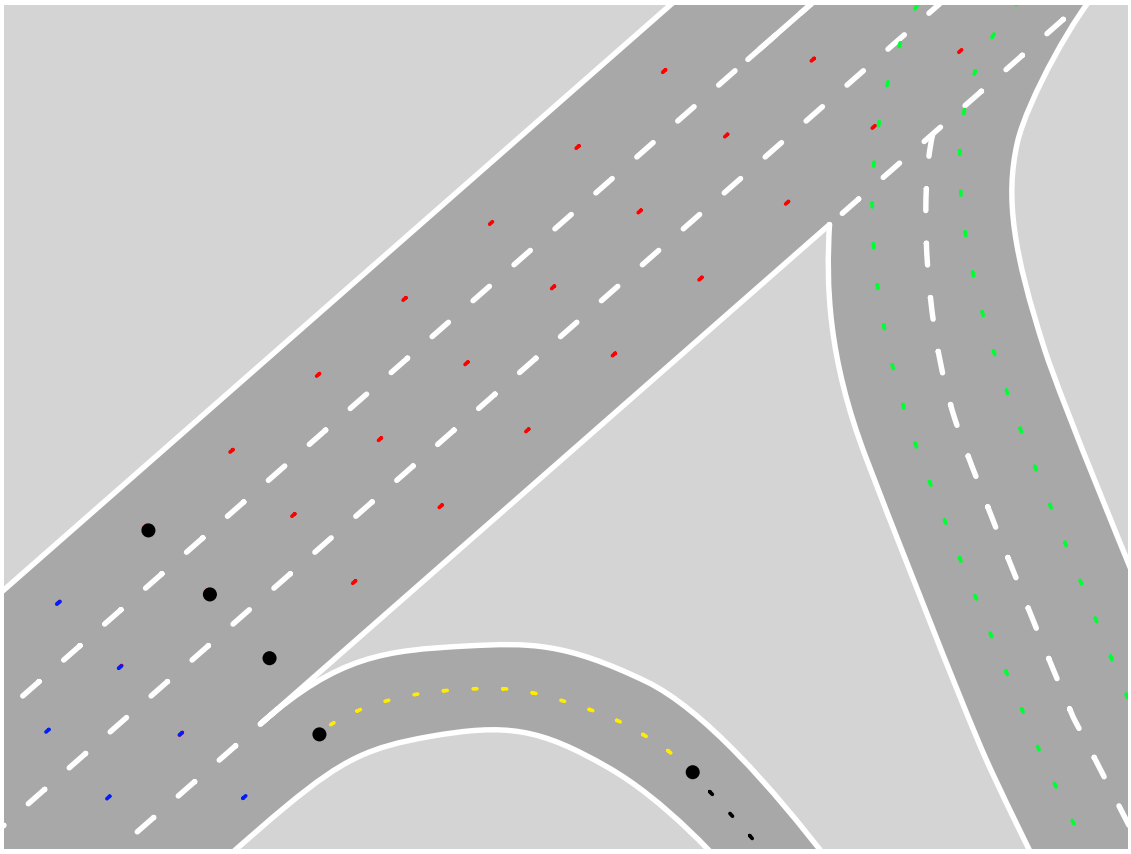
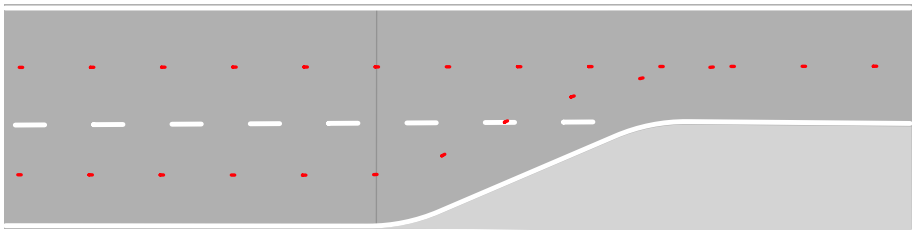
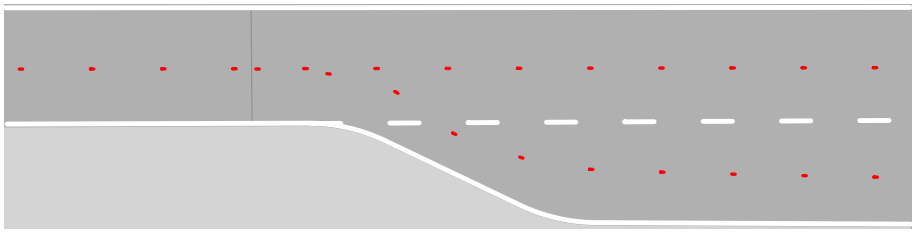


Figure 2.5: It shows a part of the HD map containing multiple road segments, with colored dash lines representing the centerline of each lane. Different colors represent they belong to different road segments, totalling five road segments shown in the figure. The distance between each node in a certain centerline may vary depending on the road topology. In addition, those big black dots represent some of the connecting points.

Based on different graph topologies, there are three different lane types. Typically, one centerline has one predecessor and one successor, in which its successor is a continuation of the current lane. However, in some cases, it instead has two predecessors or two successors, as in the case of two lanes merging or splitting. Therefore, these cases can be classified as merges and splits. These merges and splits need to be treated separately as the lane markers on each side of their centerlines should be divided into separate lane marker instances. The cases of merges and splits are shown in Figure 2.6.



(a) Two lanes of type continuation merge into one, with the triangular-shaped lane being of type merge.



(b) A lane of type continuation splits into two lanes of type split.

Figure 2.6: Visualizations of lane merge and split.

2.3 Object Detection With Deep Learning

Object detection is an essential task in computer vision and includes two tasks, object identifying and object location. The majority of object detection is based on images taken by mono-camera or multi-cameras. And it is the basis for many computer vision applications, such as autonomous driving and intelligent monitoring [21]. As mentioned in [21], it is widely accepted that object detection has gone through two periods, with 2014 as the dividing line, before 2014 as the traditional object detection period, and after 2014 as the deep learning-based object detection period. Traditional object detection methods are mostly based on sliding window framework or feature points matching, such as histogram of oriented gradients detector [22], deformable part-based model [23]. From AlexNet [6] that used convolutional neural networks to improve image classification accuracy significantly in the imageNet competition, deep learning is widely used in the field of computer vision. The main methods in object detection with deep learning include two-stage methods and one-stage methods. This section will introduce the object detection methods base on deep learning.

2.3.1 Two-stage Methods

R. Girshick et al. proposed R-CNN [6] in 2012 and it is the first two-stage object detection method. The main idea of R-CNN is first to extract the regions of interest (RoI) [6] containing the objects detected in the input image. Then the convolutional neural network extracts the feature information in the RoI and trains the model on the feature map. Finally, a multi-class SVM is used to identify the class of each RoI. Therefore, this idea can be simplified into two steps, localization and classification. Faster R-CNN is currently the most classic two-stages algorithm, its first step is

to use the region proposal network to obtain RoI from the input image to achieve localization, and the second step is to perform classification and recognition through RoI pooling and fully connected layers.

The popular two-stage methods also include Faster-RCNN [24] and feature pyramid networks [25]. Faster-RCNN extracts regions containing useful information through the region proposal network and then uses the convolutional neural network for feature extraction. Compared to the one-stage methods that directly apply convolutional operations on the input images, the two-stage methods generally have better localization and recognition. However, since two-stage methods divide object detection task into two steps, they often need more time for training and inference.

2.3.2 One-stage Methods

In one-stage methods, the predefined anchors are used to replace the region proposals step in two-stage methods. For each anchor, the network calculates the probability that the anchor covers the object, as well as the offset between the anchor and the object. The network output includes the object's class, confidence and offset relative to the anchor. The bounding box of the object can be obtained through the offset and anchor position.

Yolo [26] is a one-stage detector proposed in 2015, and Yolo means "you only look once". It uses one network to complete three tasks: the classification of the foreground and background, the classification of the object and finding object bounding boxes. Yolo uses features from the entire image to predict each bounding box and predicts all bounding boxes across all classes for an image simultaneously. Yolo divides the input into $S * S$ grid. If the center of an object falls into a grid cell, this cell is responsible for detecting this object. Each grid needs to predict B bounding boxes position (x, y, w, h) and their confidence. The object detection result can be obtained by the pre-determined threshold and the class-specific confidence score.

One-stage detector simplifies the framework of two-stage detector, which can realize end-to-end learning and significantly improve the speed of training and inference. In the early stage of one-stage methods, there may be problems of low accuracy and poor detection effect on small objects. However, with the continuous improvement of one-stage methods, the accuracy has been dramatically improved, and many networks detect the object fast and well, such as SSD [27], RetinaNet [28], YOLOX [29].

2.4 Federated Learning

With the continuous development of deep learning, the model's parameters increase exponentially, and the data requirements for training the model are also increasing. There is a problem of data silos among data owners, and with the increasing emphasis on data privacy and data security, the problem of data silos is becoming more serious. With the introduction of data protection laws in various countries,

such as general data protection regulation (GDPR) [30], california consumer privacy act (CCPA), the security of users' personal information and data has been greatly improved, but this also poses new challenges to the data required for deep learning. In traditional machine learning, model training is carried out through large data sets which is collected at a central data center, and steps such as data collection, data cleaning, model training, and model deployment are required, but data collection and model deployment may violate the legal requirements of GDPR. At the same time, there will also be data privacy leakage and data transmission overhead problems in data collection. In 2016, Google proposed the concept of federated learning [31] based on distributed learning, that is, the data is unevenly distributed on many nodes, and a high-quality global model is trained through these data. This section will introduce two types of classification in federated learning.

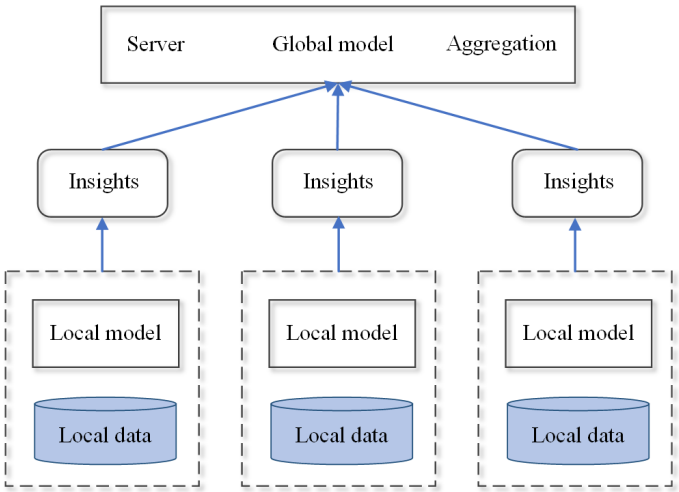


Figure 2.7: Illustration of federated learning, each client does not share data but shares the insights from training data to get a global model.

2.4.1 Horizontal Federated Learning

Horizontal federated learning [32] means that the data between different clients have the same feature space but different sample spaces. Each client has different data in the same feature space, which means that more data with the same features can be used for training without sharing the raw data through federated learning. K different clients train a global model under the coordination of a central server, while each client keeps its own data locally. The main steps of horizontal federated learning are that initialize all models, the client model calculates the gradients according to the local data, and the client encrypts the gradient and sends it to the server. The server decrypts and aggregates the received gradient information and sends the aggregated information back to the client. The client updates the model parameters with the received gradients. Repeat the above process until the objective function converges. In the process of federated learning, each client only sends encrypted gradient information to server without sharing any raw data. Horizontal federated learning is also called feature-aligned federated learning.

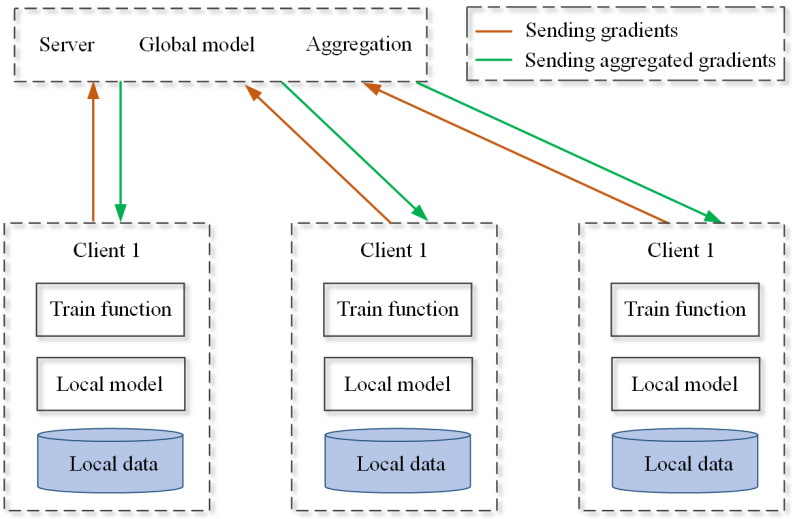


Figure 2.8: Illustration of horizontal federated learning, the clients send the gradient to the server, the server sends the aggregated gradient back to the clients, and the clients update the model parameter. There is no communication between the clients.

2.4.2 Vertical Federated Learning

Vertical federated learning [32] means that the data between clients has the same sample space but different feature spaces. This means that for the same sample, different clients have different features of the sample. Therefore, federated learning can aggregate more features of the same samples and use more features for gradient descent while protecting privacy. However, a trusted third party is needed to find common samples among different clients in vertical federated learning. In vertical federated learning, each client first initializes parameters. The server sends the public key to all clients, each client calculates the intermediate results of its own related features, and each client performs public key encryption communication to exchange the intermediate results. After the exchange, each client calculates its own gradient and loss, but the gradient and loss at this time are in homomorphically encrypted data that cannot be used to update the local model. After that, the client sends the encrypted gradient and loss to the server, the server uses the private key to decrypt the gradient and loss and sends it to the client, and the client updates the gradient. Vertical federated learning is more complicated than horizontal federated learning since different clients need to communicate. Therefore, homomorphic encryption is required to ensure the normal operation of vertical federated learning. Vertical federated learning is also known as sample-aligned federated learning.

2.4.3 Cross-device

Google introduced federated learning [33] through mobile and IoT devices in 2018. In [34], the authors divide federated learning into two types, cross-device and cross-silo, based on the number of participating clients and clients' reliability. The cross-device type is more in line with the original definition of federated learning. In cross-

device federated learning, clients are usually mobile and IoT devices, and clients' main feature is enormous number and high unreliability. Since the clients' network environment is mostly wi-fi or cellular communication, communication is often the primary bottleneck. In each round, 5% or more clients are expected to fail or drop out due to battery, network and other reasons.

2.4.4 Cross-silo

The clients of cross-silo federated learning are data centres of different organizations. The main feature of these clients is high reliability and minor numbers (typically 2-100). Since the data centres have a reliable and stable network environment, they can maintain a reliable connection with the server so that all clients can participate in each round. Through cross-silo federated learning, data from different organizations can be used to train the model, and the problem of data silos between different organizations can be solved.
It should be noted that the data is locally generated and kept decentralized, whether cross-silo or cross-devices federated learning. Each client has local data and can not access other clients' data.

2.5 Lane Detection

The field of vision-based lane detection has grown considerably in the last decade. Road lane detection mainly includes traditional methods based on filtering and Hough transform and methods based on deep learning.

2.5.1 Traditional Lane Detection

Traditional lane detection generally obtains road images through onboard cameras. The RGB image is generally converted to a grayscale image followed by a filter to reduce the noise in the image. Afterwards, an edge detector will be used to obtain the edge information from the processed image and then obtain lane information through the hough transform.

Tran Trung-Thien [35] proposed an adaptive lane detection method through the HSI colour model in 2010. They convert RGB images to HSI images. Using the image's three components of H, S, and I, the lane position can be marked more accurately without any module for reducing noise and filtering in the image. Wang Jianfeng [36] proposed a lane detection method combined with ideas region of interest and random Hough transform (RHT), which improves the processing speed compared to performing Hough transform on the input image directly. Ghazali Kamarul [37] in 2012 used an algorithm based on the H-MAXIMA transform and an improved Hough transform to detect lane lines, which can detect rapid changes in lanes. He also adopted the idea of RoI, dividing the input image into the near field of view and the far field of view, reducing the search space by regions of interest.

2.5.2 Lane Detection With Deep Learning

With the wide application of deep learning in semantic segmentation and object detection, lane line detection based on deep learning has developed rapidly. In particular, the powerful feature extraction capability and classification performance of the convolutional neural network greatly improve the accuracy of lane detection.

Huval Brody [38] implemented lane detection in 2015 by adding an additional class to the CNN used for vehicle detection. Furthermore, they use DBSCAN to cluster the lane prediction information to get the lane information in the input image. He Bei [39] proposed a Dual-View Convolutional Neutral Network framework in 2016, which is trained on the front-view and the top-view images. The front-view image can remove vehicles and fences, and the top-view image can remove non-club-shaped structures (ground arrows and words). The Dual-View idea of DVCNN makes the lane detection results more accurate. Kim Jiman [40] proposed an end-to-end transfer learning method in 2017 to estimate left and right ego lanes directly and separately. First, use the imageNet dataset to train the VGG network, and the pre-trained VGG network is symmetrically concatenated as a deconvolution network. The modified network is trained to segment each class component (general scene, road scene). Finally, the transferred network is trained to convert the road scene into the left and right ego line, resulting in a network that can predict lane lines. Tabelini Lucas [41] proposed PolyLaneNet in 2021, which uses polynomials to represent lane lines. PolyLaneNet uses deep polynomial regression to make the input a forward-looking image, and the output is polynomials representing each lane marking in the image. PolyLaneNet is faster than the algorithm using segmentation and anchoring mechanisms, but the refinement of lane line features is slightly weaker. Davy Neven [7] proposed a fast lane detection algorithm base on instance segmentation approach in 2018. This paper cast the lane detection problem into an instance segmentation problem, with each lane forming its own instance. Further, the authors apply a learned perspective transformation instead of a fixed “bird’s-eye view” transformation, ensuring robustness against road plane changes. The idea of transforming the lane detection problem into an instance segmentation problem has become the basis of current 3D lane detection.

2.6 3D Lane Detection

Traditional 2D lane detection has poor performance in autonomous driving due to the case of uneven road. Predicting the 3D lane is a necessary task and enables effective and safe driving.
Brody Huval [38] used the data obtained by LiDAR and vehicle cameras to predict ego-lane boundaries of highway through CNN in 2015. Garnett Noa proposed end-to-end 3D-LaneNet [9] in 2019. The 3D-LaneNet uses only the image taken by vehicle cameras as input to predict the 3D lane. It introduces intra-network inverse-perspective mapping and anchor-based lane representation to achieve state-of-the-art performance. Guo Yuliang [10] proposed Gen-LaneNet in 2019. It introduces a new geometric guidance lane anchor representation method to obtain 3D lane in

a more reliable coordinate system. Gen-LaneNet uses a scalable two-stage framework which decouples the learning of image segmentation subnetwork and geometry encoding subnetwork. Compared to 3D-LaneNet, Gen-LaneNet drastically reduces the amount of 3D lane labels required to achieve a robust solution in real-world application. Efrat Netalee proposed 3D LaneNet+ [11] in 2020 to achieve anchor free 3D lane detection. 3D LaneNet+ can detect 3D lanes of arbitrary topology more accurately through introducing the anchor-free method. Compare to 3D-LaneNet, it significantly improve the generalization to complex lane topologies, curvatures and surface geometries. Yan Fan proposed an extrinsic-free, anchor-free 3D lane detection method called SALAD [42] in 2022. SALAD consists of two branches, the semantic-awareness branch and the spatial context branch. And it does not require human-crafting anchors and the supervision of extrinsic parameter.
The above recent works show that the trend of lane line detection is from 2D lane line detection to 3D lane line detection, from complex framework to end-to-end, from using multi-sensor to using monocular camera.

3

Methods

In this report, we introduce an automatic labeling pipeline using a prerecorded HD map. As references, a semi-automatic approach is used to generated 3D lanes annotations through combining manually annotated 2D lane images with projected depth maps from aggregated LiDAR point clouds. For the deep learning part, we use 3D-LaneNet [9] as our network architecture which is a strong and well-known baseline for 3D lane detection. We choose Flower framework [43] to implement federated learning experiments, since Flower is one of the only FL frameworks that supports both simulation and real federation mode and it is one of the well-developed and maintained centralized FL framework. Each of these methods will be described in details in the following.

3.1 Network Architecture

In the 3D-LaneNet network architecture, the input is an image taken by a front-looking camera mounted on vehicle, and the output is the predicted 3D lane information of the road in this image. The network architecture of 3D-LaneNet is shown in the Figure 3.1. The network of 3D-LaneNet includes two pathways, namely the image-view pathway and the top-view pathway. The authors call this architecture as the dual-pathway backbone. It should be noted that although there are two pathways, the input of the network is only one RGB image, and the input of the top-view pathway is the feature map obtained through the projective transformation layers.

The 3D-LaneNet is mainly composed of input layer, convolution layer, pooling layer, and fully connected layer. The feature maps of some layers of the image-view pathway will be used as the input of the top-view pathway through the projective transformation layer, which will be described in detail later. We are using the Tensorflow version of 3D-LaneNet used in last year's project [44]. They rewrote the unofficial Pytorch version of 3D-LaneNet in Gen-Lanenet [10] to the tensorflow version.

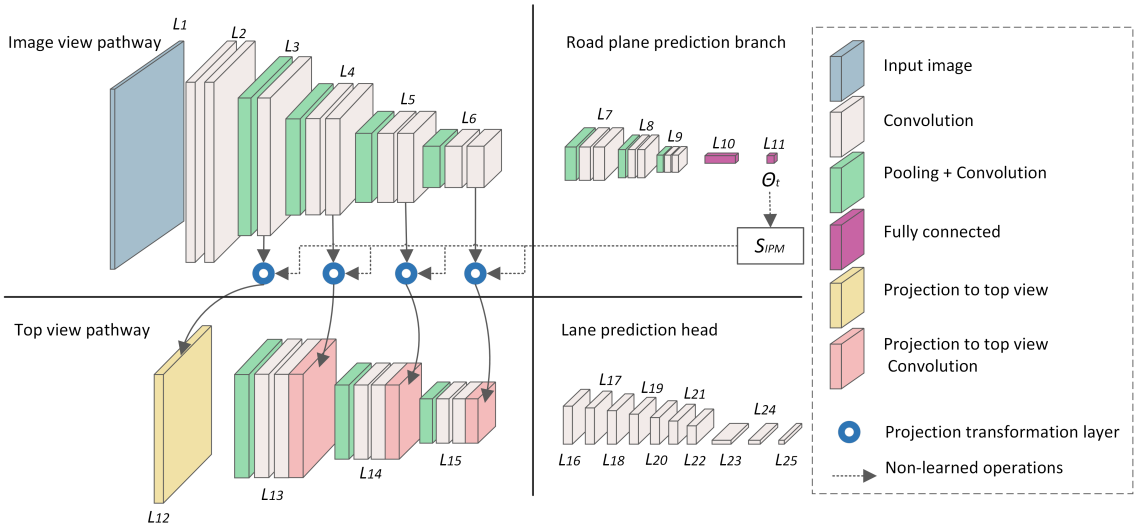


Figure 3.1: 3D-LaneNet network architecture. The network comprises a dual pathway (image-view pathway and top-view pathway) and a projective transformation layer. This figure is inspired by [9].

Table 3.1: Illustration of the dimensional information of the 3D-LaneNet network architecture. w and h are the width and length of the RGB image input to the network. $\hat{w}$ and $\hat{h}$ are the width and length of the first layer input in top-view pathway.

Layer	Width	Height	Filter
1	w	h	N/A
2	w	h	64
3	$w/2$	$h/2$	128
4	$w/4$	$h/4$	256
5	$w/8$	$h/8$	512
6	$w/16$	$h/16$	512
7	$w/32$	$h/32$	256
8	$w/64$	$h/64$	128
9	$w/128$	$h/128$	64
10	$\text{dim} = 1 \times 1 \times 64$		
11	$\text{dim} = 1 \times 1 \times 1$		
12	$\hat{w} = 128$	$\hat{h} = 208$	N/A
13	$\hat{w}/2$	$\hat{h}/2$	128
14	$\hat{w}/4$	$\hat{h}/4$	256
15	$\hat{w}/8$	$\hat{h}/8$	256
16	$\hat{w}/8$	24	64
17	$\hat{w}/8$	22	64
18	$\hat{w}/8$	20	64
19	$\hat{w}/8$	16	64
20	$\hat{w}/8$	12	64
21	$\hat{w}/8$	8	64
22	$\hat{w}/8$	4	64
23	$\text{dim} = 1 \times \hat{w}/8 \times 256$		
24	$\text{dim} = 1 \times \hat{w}/8 \times 64$		
25	$\text{dim} = 1 \times \hat{w}/8 \times N(2k+1)$		

3.1.1 Image-view Pathway

The input of the image-view pathway is an RGB image, and its main purpose is to process and save the features from the input image. When this pathway obtains the input, the RGB image will be convolved and pooled. The image will go through multiple pooling and convolution layers in this pathway. From the network architecture, we can find that $L3$, $L4$, $L5$, and $L6$ all consist of pooling and convolution layers. The feature maps of these four layers will be input to the projective transformation layer in combination with the camera information obtained by the road plane prediction branch to create the top-view feature maps.

3.1.2 Road Plane Prediction Branch

The road plane prediction branch belongs to the image-view pathway, which processes the feature map output from the $L6$. In this branch, after the feature map is processed by the combination of the pooling and the convolution layers, it will be processed by two fully connected layers. Finally, the output of the image-view pathway is the pitch and height of the front-looking camera mounted on the vehicle. These two parameters will be transferred to the projective transformation layer as input along with the previously generated feature maps. The camera's pitch and height can determine the rotation matrix to convert from the camera coordinate to the road coordinate system.

3.1.3 Projective Transformation Layer

The projective transformation layer is an essential part of 3D-LaneNet. The feature map generated from $L3$ to $L6$ and the output of the image-view pathway are processed by the projective transformation layer as the input of the top-view pathway. Its main function is to convert the input vehicle camera image into the top or birds-eye-view (BEV) image.

The idea of the projective transformation layer is based on inverse perspective mapping [45], which converts a forward-view image into a top view or BEV image, which is widely used in the field of camera-based autonomous driving. In 3D-LaneNet, the top view image is generated by the feature maps of $L3$ to $L6$ and the camera height and pitch from the output of road plane prediction branch. The RGB image input in the image-view pathway is subjected to convolution and pooling operations, the number of channels in the image remains unchanged, and because of the displacement invariance of the convolution neural network, the location feature of the input image is also preserved.

In 3D-LaneNet, the author uses inverse-perspective mapping (IPM) to make each pixel in the top-view image correspond to a predefined position on the road, thus generating a rectangular grid of size $w * h$. Its size is the pixel size of the top-view image, and each point in the rectangular grid corresponds to a pixel in the top-view image.

In this experiment, we specify the size of the top-view image in the road coordinate, that is, the size of the space we extract from the image view image, the horizontal coordinate x ranges is [-10m, 10m], and the vertical coordinate y ranges is [5m, 101m], that is, a 20m*96m space is extracted from the camera view image.

At the same time, since the top-view image doesn't need height information, the z-value in the road coordinate is 0, and y-value in top-view is the projection of y-value in the camera coordinate on the ground. In the new coordinate system, the y-direction is the camera's forward direction, the z-direction is the normal of the road ground, and the origin is the projection of the camera position on the road

ground.

In the conversion process, the main parameters used are the height and pitch of the camera. Through these two parameters and the fixed parameters of the camera, the image-view can be converted to the top-view. Since we used the same camera in the dataset and the camera was error-corrected when we made the dataset. So in this experiment, the transformation matrix only needs the camera height and pitch. After transforming to the top-view, the image looks irregular, caused by the principle of camera perspective. Although each grid in the matrix grid we created corresponds to a pixel of the top-view image, the pixels of the feature map and top-view image are not one-to-one due to the stretching during the conversion process. So for each channel, the pixel values in the matrix grid are calculated by bilinear interpolation. The final transformed and calculated matrix grid is the top-view image.

The projective transformation layer is essential for the 3D-LaneNet dual-pathway backbone. The main reason for adopting the dual-pathway architecture is that the network can extract the feature from the image-view and top-view images at the same time. In the top-view image, the appearance of the lanes is similar, and the lanes are more obvious, which is conducive to extracting lane features. The image-view image has a larger visual area that can capture some features that are difficult to extract from the top-view image, such as trees and skylines. By fusing the feature information of the two paths, the network can better infer the 3D lane information from images.

3.1.4 Top-view Pathway

The input of the top-view pathway is the top-view feature maps received from the projective transformation layer. In this pathway, the network will perform convolution and pooling operations and extract the features from top-view images. Then the network will send the features map input to the lane prediction head to make the final prediction of the 3D lanes.

3.1.5 Lane Prediction Head

The input of the lane prediction head is the feature map from $L15$, and the output is the 3D lane from $L25$. The output is the anchor-based lane representation information (based in the road coordinate). Since 3D-LaneNet is equivalent to converting the lane detection problem into a object detection problem, the final output 3D lane prediction includes the lateral offset and height offset of the lane and the existence of lanes confidence score in each anchor.

The anchor of each sample is a set of equally spaced X-lines $\{A^i\}_{i=1}^{N}$ in road coordinate system. Since it is 3D lane information, the anchor contains height information, and the height of all anchors is zero in 3D-LaneNet. In the experiment, the 3D lane

is limited to the lateral space of [-10m, 10m]. When the number of anchors is 16, it means that the lateral offset of each anchor is about 1.33 meters.

For a 3D lane in an anchor, there will be predefined K points in the y-direction to represent lane information, and the y-value of these K points for the same network is determined. As shown in Figure 3.2, an image is dividend into N anchors, and each anchor has K points to describe the lane position. Generally the visible distance of images of vehicle cameras is approximately up to 100 or 120 meters, the value of K is 10 here, and the predefined y-values are [6.5, 10, 15, 20, 30, 40, 50, 60, 80, 100]. Since the y-values of K points in an anchor are determined, the mathematical representation of each anchor is $\left(x_j^i + X_A^i, y_j, z_j^i\right) \in \mathbb{R}^3$, where x_j^i is the x direction offset at the jth K point, y_j is the predetermined y value of the jth K point, z_j^i represents the offset in the z direction at the y_j point, X_A^i is the predefined x-value of the ith anchor.

According to the above description, the value of X_A^i is predefined, and the value of y_j is also predefined, so for the lane in an anchor, its description can be simplified as $(\mathbf{x}^i, \mathbf{z}^i) = \left\{\left(x_j^i, z_j^i\right)\right\}_{j=1}^{K}$, that is, only the x direction offset and the z direction offset. In the lane prediction head, the final output of $L25$ has size $(2 * K + 1) * N$. K is the predefined number of y-values in the vertical anchor; $2 * K$ indicates the number of parameters used for represent a lane $(\mathbf{x}^i, \mathbf{z}^i)$; and $+1$ indicates the confidence of the existence of lanes; N represents the number of anchors for a sample.

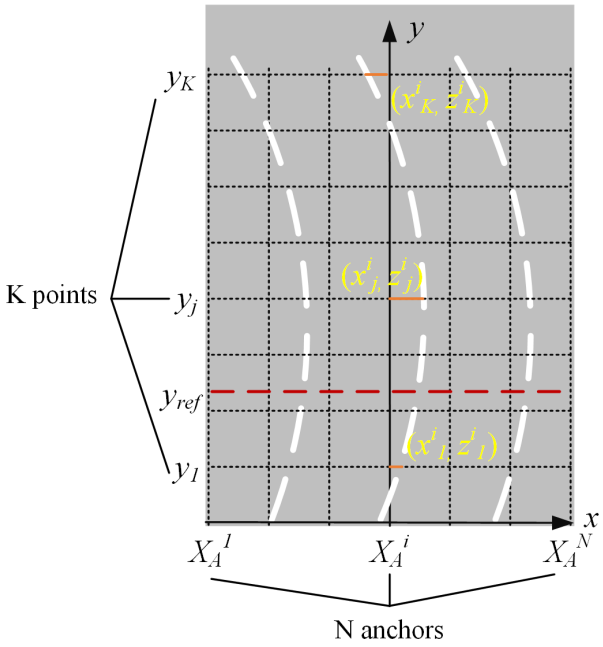


Figure 3.2: Illustration the anchor on a given image and the lane prediction at the predefined y-values. This figure is inspired by [44].

A y_{ref} will be set during the training process of the 3D-LaneNet network. If the

lane in the ground truth has an intersecting relationship with the $y = y_{ref}$, this lane will be assigned to the nearest anchor on the y_{ref}. If there is no intersection of lane lines and $y = y_{ref}$, the paper's authors believe these lanes can be ignored during the training process. If the same anchor is assigned multiple lanes, only the longest lane is selected for training during the training process.

3.1.6 Loss Function

Through the introduction of the network architecture of 3D-LaneNet, we can find that the network output includes the height and pitch of the camera; $(\mathbf{x}^i, \mathbf{z}^i) = \left\{\left(x_j^i, z_j^i\right)\right\}_{j=1}^{K}$ are used to represent the lane position and $\{p^i\}_{i=1}^{N}$ of the probability of the existence of the lane in the anchor.

The loss function of 3D-LaneNet (as shown in the equation 3.1) consists of three parts according to the network outputs. The variables with hat are the predicted values by the network, and the variables without hat represent the ground truth.

$$\begin{aligned}\mathcal{L} = &- \left[\sum_{i=1}^{N}\left(p^i \log\left(\hat{p}^i\right) + \left(1-p^i\right)\log\left(1-\hat{p}^i\right)\right)\right] \\ &+ \sum_{i=1}^{N} p^i \cdot \left(\left\|\boldsymbol{x}^i - \hat{\boldsymbol{x}}^i\right\|_1 + \left\|\boldsymbol{z}^i - \hat{\boldsymbol{z}}^i\right\|_1\right) \\ &+ |\theta - \hat{\theta}| + |h - \hat{h}|\end{aligned} \tag{3.1}$$

The first row represents the cross-entropy loss for the probability that a lane exists in a given anchor. $\{\hat{p}^i\}_{i=1}^{N}$ means the existence probability of the lane in the prediction result, the value range is [0, 1]; $\{p^i\}_{i=1}^{N}$ means the existence probability of the lane in the ground truth, and its value is simplified to 1 or 0. The second row represents the L1-norm penalty of the network's prediction of the lane positions. Since the y-values are predefined, and the network's output is $(\mathbf{x}^i, \mathbf{z}^i)$ for a lane in an anchor, only the L1-norm penalty for x and z is calculated here. At the same time, the L1-norm penalty is multiplied by the lane probability of ground truth, that is, only the existing lanes are penalized. The third row represents the penalty for the prediction of camera parameters, where θ represents the camera's pitch, and h represents the camera's height. This part calculates the absolute value of the difference between the predicted value and the ground threshold. The last thing to note is that the weights of these three parts of the loss function are the same.

3.2 Flower Framework

In this thesis, the federated learning part uses the Flower framework [43]. The Flower is a flexible end-to-end federated learning framework. We can simulate federated learning on the cluster through the Flower framework in this report to reduce

the problems encountered in the actual deployment in edge devices. The Flower framework supports several popular machine learning frameworks, such as TensorFlow, PyTorch, MXNet and Scikit-learn, and Flower can be used as the components of these frameworks. Since the design of the Flower framework considers the heterogeneous device problem in federated learning, it can run stably in environments with various participating hardware devices, operating systems, and programming languages. At the same time, the high scalability of the Flower ensures that large-scale federated learning can be performed. Moreover, Flower's scalable feature allows us to perform large-scale federated learning on numerous vehicles in different regions. Flower framework can be regarded as including two parts server and client.

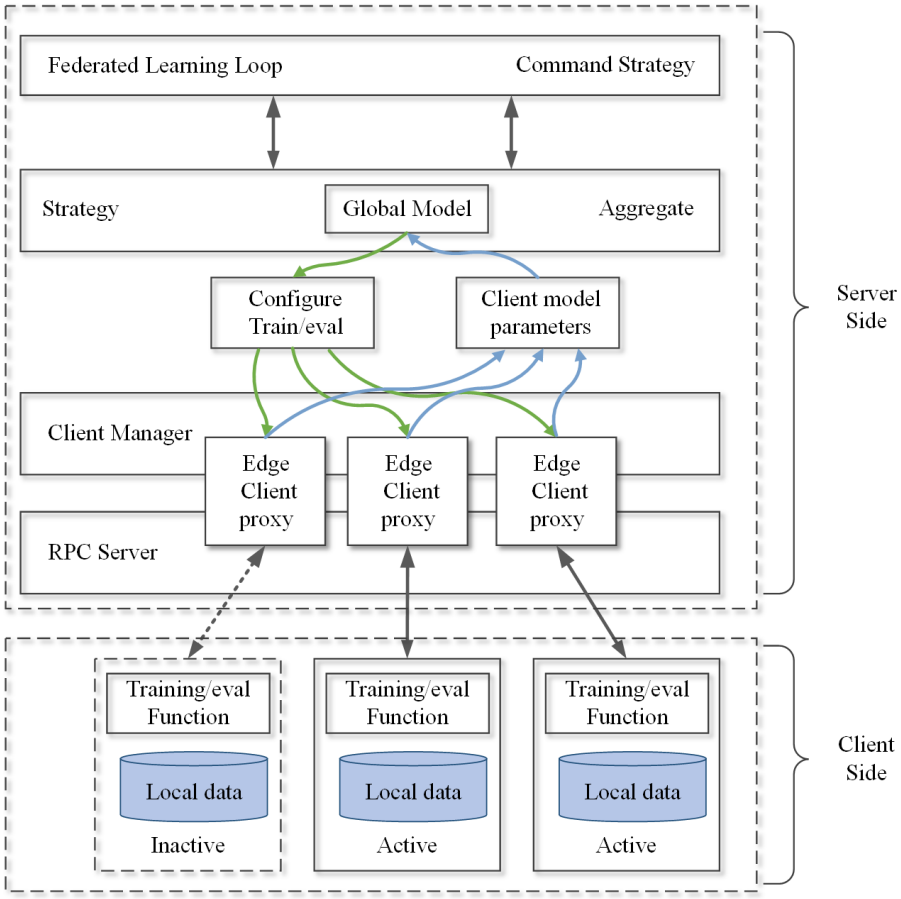


Figure 3.3: Illustration the architecture of the Flower framework. The server includes FL loop, strategy, client manager, RPC server, and global model, and the client includes client model and local data.

3.2.1 Server

In the Flower framework, the server side is the crucial part. It directly determines how to configure federated learning and how to aggregate parameters. And hyperparameters can also be assigned to the client models by the server. In the Flower framework, the server side consists of the client manager, the FL loop, and a strategy

which can be customized by users. FL loop is one of the cores in federated learning using the Flower framework, which coordinates the entire federated learning process. It requires the current strategy to configure the next round of federated learning. The strategy determines the process of federated learning, such as whether to perform centralized or federated evaluation and what algorithm is used to aggregate parameters. In a federated round, the server samples the clients through the client manager and aggregate their parameters and evaluate their models.

3.2.2 Client

In the Flower framework, the client is mainly used to receive messages from the server and send client model parameters to the server. The message content accepted by the client includes training instructions, evaluation instructions, and model hyperparameters (learning rate, epoch). The client uses the local training function and data to train the model according to the message sent by the server. When a client receives the federated evaluation instruction, it uses the local evaluation function for evaluation. And the client side sends the model parameters to the server to update the global model after a round of local training.

3.2.3 Federated Averaging

The federated averaging (FedAvg) [33] is a classic algorithm in federated learning. In the FedAvg algorithm, the server randomly selects a certain number of clients, and then each selected client calculates the average gradient through its local data and update local model parameters. The server updates the global model parameters by aggregating the local model parameters by weighting the local models based on the number of their contributing samples.

The objective function of federated averaging is equation 3.2. K means that there are K clients participate in a federated learning training. And P_k represents the training samples in the kth client, and n_k represents the number of samples in the kth client. The equation shows that the global model's loss is the weighted average of the local models' loss. It's worth noting that a shared starting model is used for each round of FedAvg. Since [33] found that averaging the models with shared initialization significantly improved compared with independent initialization.

$$\begin{aligned} f(w) &= \sum_{k=1}^{K} \frac{n_k}{n} F_k(w) \\ where \quad F_k(w) &= \frac{1}{n_k} \sum_{i \in P_k} f_i(w) \end{aligned} \tag{3.2}$$

3.3 Datasets Generation

This section will describe the collection and creation of the datasets developed for this thesis. Due to the lack of a 3D lane detection benchmark, a labeled dataset with ground truth is created for developing the baseline model following the method in [44]. Afterwards, a prerecorded HD map is used to label a dataset fully automatically. The data of both datasets are mainly collected from highways because the prerecorded HD map that we used in this project, has better coverage of highway areas. Another reason is that the vehicle is more likely to run steadily on the highway, resulting in a more reliable calibration. For the final generated training data, we need them to be in the road coordinate system C_{road} described in section 2.1.6.

3.3.1 Dataset with Ground Truth Label

The dataset with ground truth labels is created by combining depth maps from aggregated LiDAR point clouds with manually labeled 2D lane annotations. The aggregated depth maps consists of the depth value (distance relative to the camera) of each pixel given a single (monocular) RGB image. With the help of it and the camera extrinsic matrix, the pixels annotated as lanes in 2D annotations can eventually be transformed into 3D point clouds that correspond to the ground truth of lanes in 3D space. Some algorithms are applied to post-process the generated point clouds, in order to remove noise and create continuous representations of lanes in 3D. An overview of how 3D lanes are created by combining the 2D lane annotations and the aggregated depth maps is shown in Figure 3.4. In general, DBSCAN (Density-Based Spatial Clustering of Applications with Noise) and RANSAC (Random Sample Consensus) are used to filter out the outliers in the generated 3D lane point clouds. Details can be found in section 3.7.1 in [44].

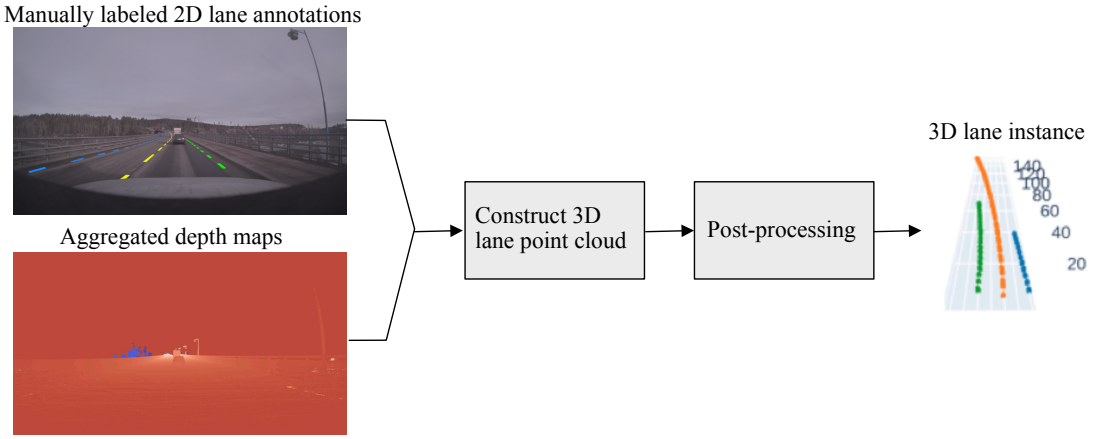


Figure 3.4: An overview of how 3D lanes are created by combining the 2D lane annotations and the aggregated depth maps. The 2D annotations used are so-called instance annotations, hence different lanes are treated as different instances and represented as different colors in the figure, both in the 2D annotations and the generated 3D annotations.

In the 2D annotation, which also applies to the actual situation, there are different types and colors of lane marking, such as solid, dashed, and botts dots lane markings. This thesis project ignores these different types of lane markings, and only their topologies are reserved. In addition, necessary interpolations are performed on all lanes to make lane markings such as dashed ones continuous representations in 3D space for the later generation of anchors for training data.

The final step entails determining the ground truth pitch θ and height h_{cam} of the camera relative to the local road plane P_{road} as described in section 2.1.6 and in Figure 2.3. In our case, static pitch and height calibration is used for each separate drive of the vehicle. Therefore, any pitch and height changes due to vehicle dynamics, such as running on hilly roads, are not taken into account. The determined pitch and height will also be used to transform the generated 3D lanes labels from the camera frame eventually into road coordinate C_{road}.

In the end, we generated a dataset containing a total of 2270 images of size 3848×2168. In this dataset, the data were mainly collected in two different regions under two different time intervals. The entire dataset is divided into training and validation sets of sizes 2034 and 236 respectively, where this validation set will be used throughout all the experiments. According to the time of data collection, the training set is divided into two parts with sizes of 620 and 1414, which are used as two datasets representing two clients to participate in the following decentralized training experiments.

3.3.2 Generating Labeled Data using HD map

This section describes our automated labeling pipeline used to generate 3D lane markings for each corresponding images. A conceptual pipeline flowchart is shown in Figure 3.5.

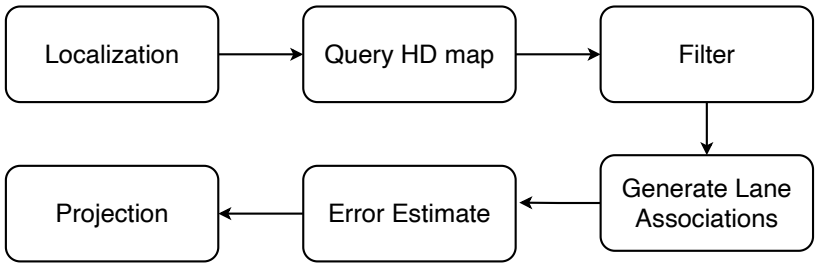


Figure 3.5: Flowchart of the automatic labeling pipeline by using HD map.

3.3.2.1 Localization in HD map

The automatic labeling pipeline starts from the localization in the HD map, so the sensors for positioning need to be extremely accurate. In our case, a highly accurate GNSS/INS, namely an OXTS, is used. The OXTS is more accurate than an ordinary GPS, but it is still prone to errors, especially at altitude.

Once we have obtained the precise vehicle positioning information, we can start querying the HD map. The HD map used in this thesis contains a three-dimensional map similar to the one described in section 2.2. Since creating an HD map is not the focus of this thesis project, an existing map is used. By inputting the vehicle's positioning coordinates (latitude and longitude) into the HD map, we can obtain a smaller map segment containing all the map information close to this position, which could narrow down the search area and limit the number of lanes for further analysis. In our case, we only focus on lane information, so only lane-related information is kept.

3.3.2.2 Localization Filter

For a lane detection task, we are more concerned with the lanes on the current vehicle's forward driving road, which take over major parts of the images captured by a front-facing camera mounted on the vehicle. Therefore, irrelevant lanes need to be further filtered out. For instance, lanes on the oncoming traffic direction, lanes on the bridge above or in the underpass are all irrelevant lane instances.

The filter starts with finding the road segment that is closest to the current vehicle position. As previously described in section 2.2, roads are divided into smaller segments in HD map, and all lanes in each road segment are represented as a polyline, namely its centreline. A polyline is described as a series of nodes connected by short

line segments. As shown in Figure 3.6, it is evident that the road segment containing the lane closest the vehicle's current position is the one with green nodes. After finding the road segment closest to the vehicle, continue searching for its predecessor and successor in a range of distance. As of now, the reserved road sections should include sections both ahead and behind the vehicle. The road segment behind the vehicle will be further filtered out in the projection step described later.

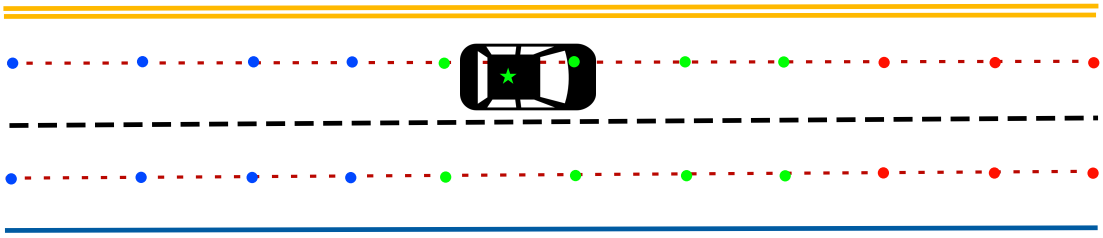


Figure 3.6: A demonstration of how to find the closest lane to the vehicle. The star states the vehicle's current position in the HD map. Dash lines with nodes represent the centrelines around the car, where the different color of nodes means they belong to different road segments. It is evident that the road segment that has the closest lane to the vehicle is the one with green nodes.

3.3.2.3 Generating Lane Associations

After getting all the road segments of interest, we need to extract the lane markings on both sides of each centreline. To generate lane associations for each lane marking, we use their 3D endpoints contained in HD map. Each lane marking has its own ID in the prefix-suffix format, where a prefix or a suffix represents a geo-location which can be used as a geo-point. If these prefix-suffix IDs are not vectors, we split them, group them together, and count how many times each appears. Normally, for a continuous lane marking, both its starting and ending points should only appear at most twice. There should be a merge or a split if it appears more than twice. In this case, we treat all lanes connected to this point as distinct instances. Otherwise, connected lane markings should be one instance. We do this to keep the same guidelines as the 2D lane annotations used in the previous dataset.

3.3.2.4 Error Estimate

Next, we will start projecting the reserved lane markings into the camera coordinate system. Throughout this section, we will use the notation ${}^{B}_{A}\mathbf{T}$ denoting transform the rigid body from frame A to B. So far, all acquired lanes are in geodetic coordinates specified by latitude, longitude, and altitude. To simplify the calculation, we need to first transform it into a local east-north-up (ENU) Cartesian coordinate system with the vehicle position as the origin, as described in section 2.1.2. The method to compute the rotation matrix ${}^{L}_{M}\mathbf{T}$, where M and L stand for map and local Cartesian respectively, is not a part of this thesis, and there is an existing implementation in advance.

To compute the rotation matrix ${}^{V}_{L}\mathbf{T}$ from the local Cartesian system and the ego-vehicle system, the orientation of the vehicle is required. Typically, the orientation of a vehicle is described by three angular values, pitch θ, roll ϕ, and heading/yaw ψ, which are shown in Figure A.1. These three angle parameters are used to calculate the three rotation matrices and multiply them to obtain ${}^{V}_{L}\mathbf{T}$. See A.1 for details about these three matrices.

An inertial measurement unit (IMU) is usually installed to measure and report the orientation of the body of a vehicle. The sensor OXTS, which was previously used for localization, can also provide these three vehicle dynamics parameters, but the pitch and roll values are not always accurate. Hence, we need to find a way to estimate the error of them. The proposed solution is to adopt the map for gathering measurements for these variables. Again, the same method as in section 3.3.2.2 is used to find the centreline closest to the vehicle's current position and find the node closest to the vehicle in it. The closest node here might be interpolated because the nodes in the map may be sparse. Afterwards, the two closest nodes from the current centreline can be estimated based on the position of the found node, one ahead and one behind the vehicle. Finally, the pitch θ of the vehicle can be estimated from the geometric relationship between these two nodes. The roll ϕ can be estimated similarly. Here we assume that the car is moving steadily, which is one of the reasons why we choose to use data on highways. Illustrations are shown in Figure 3.7.

(a) Pitch estimate, car facing left **(b)** Roll estimate, car facing inward

Figure 3.7: Illustration of how the pitch and roll are estimated using map measurements.

Another error comes from the vehicle altitude measured by OXTS. The wrong altitude will cause the road surface to float up or down on the Z-axis of the ego-vehicle coordinate. For consistency, all altitude values used in this thesis come from the HD map. The ego-vehicle coordinate's origin is located at the centre of the rear axle of the test vehicle, and the distance between them is assumed to be 0.37m throughout this project, that is, the vehicle altitude is the sum of the altitude value of the closest point in the map and this distance.

3.3.2.5 Projection

Until now, we have successfully used the rotation matrix ${}^{V}_{L}\mathbf{T}{}^{L}_{M}\mathbf{T}$ to transform the map elements from the geodetic coordinate into the ego-vehicle coordinate. The next step is to project the element further into the camera coordinate system. Cam-

era extrinsic parameters are needed to determine the transforming matrix ${}^{C}_{V}\mathbf{T}$ as described in section 2.1.5, which is measured and provided by Zenseact.

In the camera coordinate system, we can know the relative position of the lane markings to the vehicle. At this moment, we can remove the lane markings behind the vehicle (camera) and only keep the ones within a certain range (3 to 120m) in front of the vehicle (camera). Keeping the lanes in a reasonable range is because the model should only learn the pattern of visible lane markings within the camera's view instead of trying to predict the lanes that are not in the camera frame. Another reason is to guarantee the accuracy of the projection. Small translational errors in ${}^{L}_{M}\mathbf{T}$, ${}^{V}_{L}\mathbf{T}$, or ${}^{C}_{V}\mathbf{T}$ on the order of centimetres can lead to misaligned map data for an entire images. For instance, when transforming the data from the local east-north-up (ENU) Cartesian coordinate system to the ego-vehicle system, an error of one degree in pitch may cause lane markings 80 meters away to be offset by roughly 1.4 meters [14].

The last step of transformation is to convert the lanes in the camera coordinate system to the road coordinate system C_{road} using the determined camera pitch and height in the same way as in section 3.3.1, where the transformation matrix is denoted as ${}^{R}_{C}\mathbf{T}$. To sum up, for each 3D map element $\mathbf{e}_m$ in global coordinates, it generates an element in road coordinates $\mathbf{e}_r$ using

$$\mathbf{e}_r = {}^{R}_{C}\mathbf{T}\,{}^{C}_{V}\mathbf{T}\,{}^{V}_{L}\mathbf{T}\,{}^{L}_{M}\mathbf{T}\mathbf{e}_m \tag{3.3}$$

where $\mathbf{e}_m$ can be any type of map data, but in our case it is lane markings.

In the end, we successfully annotated 3304 out of 4583 images by using HD map. Most of the failed annotation cases are due to the incompleteness of the map, resulting in no lane remaining after the final transformation. One another reason is that inaccurate calibration has resulted in no available lanes within a pre-defined range in the camera coordinate system. Similarly to the dataset with the ground truth label generated in section 1, these data were collected in two same regions under two similar time intervals. The entire dataset is divided into two parts with sizes of 2178 and 1126 according to the time of data collection, behaving two distinct clients to participate in the subsequent decentralized training.

3.4 Experiments

Up until now, we have presented the neural network architecture which we will use and described our method of generating two datasets. We need to give each two datasets a simpler name for convenience. Since human labour is partially used in creating datasets composed of 2D manual annotation and aggregated depth maps, we call it the semi-automatic, or simpler, semi-auto dataset. The other dataset is more straightforward, and we call it the HD map dataset. An overview of the two datasets is shown in Table 3.2. The table shows that a part of the semi-auto dataset is extracted as the common validation set. Since there is limited data available to

generate the semi-auto dataset as ground truth, there is no test set in the whole experiment.

There is a small part of overlapping in the two datasets, the overlapping in the semi-auto dataset is regarded as a reference to evaluate the quality of the automatically generated annotations when we are developing the method for automatically annotating data using HD maps. The evaluation compares the two annotations in 3D space and the image plane after projection. The semi-auto dataset is used as ground truths, and we want the data annotated by the HD map to be as similar as possible.

Table 3.2: An overview of the generated datasets.

		Semi-auto Dataset	**HD Map Dataset**
Description		1. Collected in two different regions under two different time intervals 2. Collected on highways (speed >100km/h)	
		Frames contain both aggregated depth map and 2D lane annotations	Frames contain GPS and IMU sensor records
Training Set	Total	2034	3304
	Node 1	620	2178
	Node 2	1414	1126
Validation Set (Shared)		236 (from semi-auto)	

In order to investigate whether training a 3D lane detection model in a federated learning scenario has an impact on its final performance, and to see if the use of an automatically annotated dataset can bring the model training to a similar level, we set up the following experiments.

First, we need a baseline model trained on the semi-auto dataset in a centralized way. This model will be used as a baseline for all the rest experiments to be compared with. Next, we split the semi-auto dataset into two parts, which will participate in decentralized training as different clients/nodes. In addition, there will be a node as the central server responsible for aggregating the parameters from different nodes and then sharing the result. Note that we are simulating a federated learning scenario here, i.e. different nodes are in the same computer cluster instead of physically distributed. This experiment can reflect the impact of decentralized learning on the model's final performance.

Next, we use the HD map dataset for centralized training. By comparing the results of this experiment with the baseline model, it can be concluded whether a similar level of model training can be achieved using the automatically annotated dataset of the HD map. As different datasets are used, the optimal hyperparameters used

in training should also differ.

The last experiment simulates a complete decentralized training on automatically annotated datasets, where we split the HD map dataset into two different clients as described in Table 3.2. Comparing the results of this experiment with the rest helps us to draw the main conclusion of this thesis, i.e. whether it is possible to train a supervised model in a federal learning scenario using fully automatically annotated data to a level similar to that of a model trained using ground truth in a centralized way.

3.5 Metric

This report uses F1 score max and average precision as the primary metrics. This chapter mainly introduce the F1 score and average precision.

3.5.1 F1 score

The F1 score is a common metric used to evaluate the performance of classifiers. Since the F1 score is defined by recall and precision, it can avoid the high precision and low recall classifier. The definition of the F1 score is :

$$\begin{aligned} F_1 &= 2 \cdot \frac{precision \cdot recall}{precision + recall} \\ where \quad precision &= \frac{TP}{TP + FP} \\ recall &= \frac{TP}{TP + FN} \end{aligned} \tag{3.4}$$

In Equation 3.4, TP means true positive, that is, ground truth and prediction are both positive; FP means false positive, that is, ground truth is negative, and prediction is positive; TN means true negative, that is, both ground truth and prediction are negative; FN means false negative, that is, ground truth is Positive, predicted negative. Precision measures the percentage of the model's accurate positive predictions, and recall measures the percentage of true positives that are correctly predicted.
In the object detection task, it is necessary to perform binary classification of the prediction quality through intersection over union (IoU). As shown in Equation 3.4, we get the IoU from the ground truth and predicted overlapping and joint regions. We need a threshold (value between 0 and 1) to judge the prediction result. Predictions exceeding the threshold can be regarded as TP, otherwise FP. At the same time, if the objects cannot be predicted by the model or whose IoU value is too low can be regarded as FN.

3.5.2 Average precision and F1 score max

Average precision is another metric often used to evaluate object detection tasks, the definition of the average precision is Equation 3.5. When the threshold of confidence

in the object detection task varies from 0 to 1, recording the precision as a function of recall $p(r)$ and creating a precision-recall curve. The average precision is the average value of $p(r)$ over the interval r [0,1]. The highest F1 score calculated at each of the recorded positions is the F1 score max.

$$AP = \int_0^1 p(r)dr \tag{3.5}$$

3.6 Evaluation

To evaluate the network predictions on the validation set, we refer to the method in [9]. First, compare the prediction confidence of the network's 16 anchors with a predefined probability threshold p_{th}. Anchors with prediction confidence less than p_{th} are discarded. The anchors with predictions confidence higher than pth will be converted to 3D lanes.
When evaluating the prediction, finding the global optimal match between the predicted 3D lane set and the ground truth lane set is necessary. This report refers to the method in [9], formulating the matching problem as a bipartite matching problem and seeking the solution with the min-cost-flow solver. Since the points used to represent the lanes are too sparse, piecewise linear interpolation is used to resample the lanes to a denser representation. After oversampling, the matching cost of lane m and lane n is :

$$\text{cost}_{mn} = \sum_{j=1}^{K} d_j^{mn} \tag{3.6}$$

where djmn is :

$$d_j^{mn} = \begin{cases} \log\left(\sqrt{\left(x_j^m - x_j^n\right)^2 + \left(z_j^m - z_j^n\right)^2} + 1\right), & \text{if both lanes exist at j} \\ d_{th}, & \text{otherwise.} \end{cases} \tag{3.7}$$

The globally optimal matching is the matching with the lowest cost of ground truth lanes and prediction lanes. Since the lanes in the ground truth do not extend forever to the entire prediction range of 6.5-100 meters, matching partially overlapping lanes is achieved by adding a d_{th} penalty to the lane matching cost. It is worth noting that the plus one in the logarithm of the formula is to make the matching result more stable. For the experiments in this report, d_{th} is set to log(1.5+1).
By completing the global optimal matching by the method described above, the corresponding results of prediction and ground truth are obtained. To measure the performance of the prediction results in the matched lane, we need to introduce a distance threshold. If the distance between the point of the predicted lane and the point of the ground truth lane is less than d_{th}, the point can be considered to match.If at least 75% of the points in a lane in the ground truth have a distance less than d_{th} from the points in the predicted result, the lane can be considered to be detected. So precision and recall are defined as:

$$\begin{aligned} \text{Precision} &= \frac{\text{Pred}_{\text{correct}}}{\text{Pred}_{\text{total}}} \\ \text{Recall} &= \frac{GT_{\text{detected}}}{GT_{\text{total}}}. \end{aligned} \tag{3.8}$$

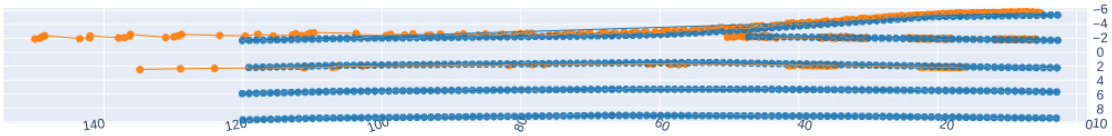
For the precision definition in Equation 3.8, $Pred_{correct}$ is the number of correctly predicted points, and $pred_{total}$ is the total number of predictions. For the recall definition in Equation 3.8, $GT_{detected}$ is the number of detected ground truth lanes, and GT_{total} is the total number of ground truth lanes. Note that the numerators in these definitions are not necessarily the same, so differ from the standard definitions of precision and recall. For example, the numerator may differ if short ground truth lanes match long predicted lanes. In this case, all points of the ground truth lane can be considered to be covered by the predicted lane, so the lane can be considered detected, while the predicted lane is considered incorrect because the short ground truth lane may cover less than 75% of the points .
The next step is to calculate the average precision and F1 score max. We obtain precision-recall curves by adjusting the probability threshold p_{th} from 0.05 to 0.95 with a step size of 0.05 to calculate precision and recall under different probability thresholds. Since the probability threshold ranges from 0.05 to 0.95, therefore, set precision to 0 and recall to 1 at the position where p_{th} is 0, and set precision to 1 and recall to 0 at the position where p_{th} is 1. The average precision is finally calculated by piecewise interpolation. Referring to 3.8, for each matched lane pair with average point wise distance less than d_{th} the average we calcaulate the absolute lateral and vertical error. The probability threshold at this point is set to the same $p_th$as when the maximum F1 score was achieved. And 40 meters is used as the boundary to divide the field of view into near (0-40m) and far (40-100m). Then the absolute lateral and vertical error of the far and near distances are calculated for matched lanes. Finally, the metrics of the evaluation model F1 score max, AP, x-error-close, x-error-far, z-error-close, z-error-far are obtained.

4

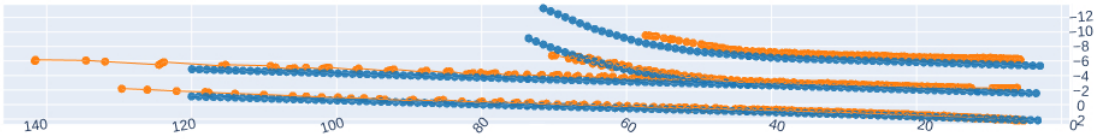
Results and Analysis

4.1 Dataset and Projection Quality

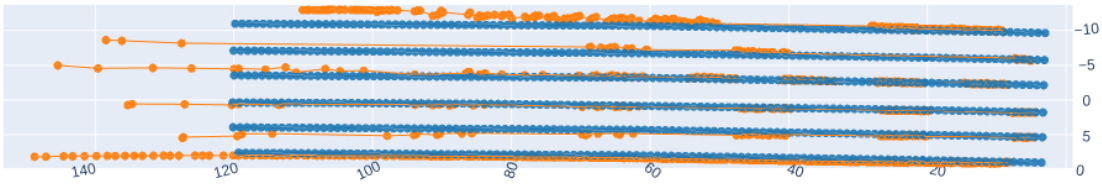
Using our pipeline to label 3D lane markings with an HD map can achieve high-precision automatic labeling, and some can be accurate up to over 100 meters. It even outperforms semi-manual annotations on some distant lane markings. Some comparisons between ground truths and automatically generated 3D lane annotations are shown in Figure 4.1.



(a) A case of lane merging



(b) A case of lane splitting



(c) A case of multiple parallel lanes

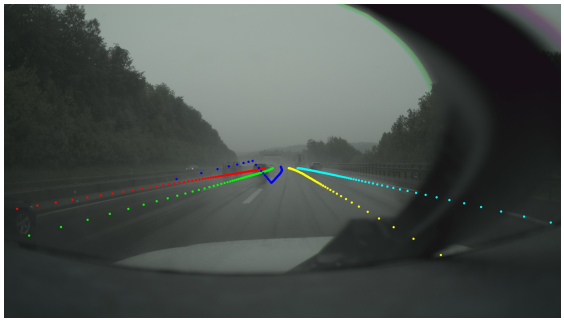
Figure 4.1: Some comparisons between ground truths represented in orange and auto-generated 3D lane annotations represented in blue. As shown in the figure, lanes labeled by HD map are limited up to 120 meters, yet they can be easily expanded further.

Errors are defined as samples with severe misalignment in the map or image, missing lane markings, or containing faulty map elements. We visually inspect the images

annotated by our auto-labeling pipeline, and some of the errors are displayed in Figure 4.2. All these data containing errors are also kept in the final training dataset because we want to know how much the final model will suffer from training with a small amount of noisy data (i.e., weak annotations). Since we used the data mainly collected on highways, where the HD map has better coverage, and the vehicle runs more steadily, resulting in more reliable calibrations and vehicle dynamics, only around 5% of data contains some obvious error.

(a) Bad projection

(b) Inaccurate localization

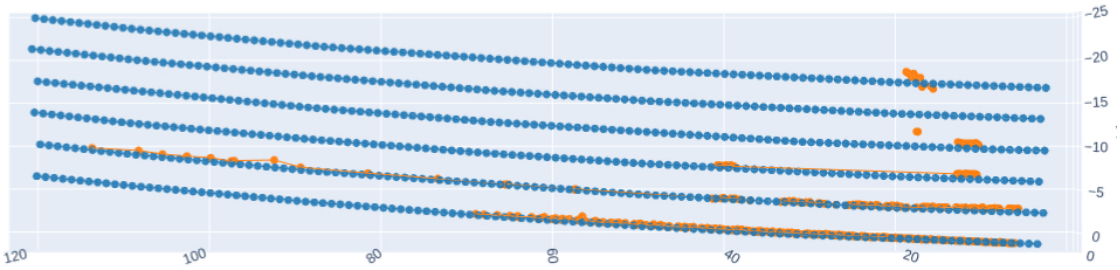


(c) False lane in map

Figure 4.2: Some errors during dataset generation using HD map.

Since two datasets are created based on different methods, the annotations of the same lane instance could turn out to be different. For instance, the occluded lanes are not visible in the semi-auto dataset but are still labeled in the HD map dataset, see Figure 4.3. This may cause the trained model to behave differently (e.g. whether to predict lanes in occlusions). This project does not consider the impact of these factors when evaluating the models.

(a) Picture taken by the forward-looking camera



(b) Comparison of two annotations under occlusion

Figure 4.3: An example shows the comparison of two annotations under occlusion. As shown in the figure, when there are many obstacles such as vehicles ahead of the ego vehicle, the annotation generated by the semi-automatic method contains many invisible lane vacancies while the one generated by the HD map does not.

4.2 Baseline and Federated Learning

First, we use the semi-auto dataset to train the model in centralized way and treat this experiment as the baseline. Since it's the experiment under optimal conditions, that is, training in the data centre and using accurate annotations as the training set. The hyperparameters used in the baseline experiment is shown in the Table 4.1, and we denote this set of hyperparameter as set1.

Table 4.1: The hyperparameters of centralized training with semi-auto dataset. LR means learning rate.

Epochs	LR	LR Decay	LR interval	Batch size
100	5e-5	2	20	8

Since the training set used for local training on clients differs from the training set used for centralized training, we use slightly different learning rate for federated training than for centralized learning. For different clients in a federated learning experiment, we use the same hyperparameters. The hyperparameters in the federated learning is shown in the table 4.2, and we denote this set of hyperparameters as set2.

Table 4.2: The hyperparameters of federated training with semi-auto dataset.

Epochs	LR	LR Decay	LR interval	Batch size
100	1e-4	2	20	8

To find the best setup for federated learning, we explore the influence of different communication rounds on the final result when the training epochs are the same, so here we set the local training epochs at clients to [20, 10, 5, 2], respectively. In other words, when the total epoch is 100, the number of communication rounds is [5, 10, 20, 50]. We selected the best local epoch and communications rounds for federated learning result in Table 4.3, based on this investigation (see Table 4.4 and Figure 4.4). This experiment is described in details further in this section.

Table 4.3 shows the results of centralized learning and decentralized learning (local training epoch is 5) using the semi-auto dataset. From the table, we can find that the results of federated learning are comparable to centralized learning when both models are trained to converge. In particular, the F1-score and average precision metrics used to evaluate object detection performance yield almost identical results, while federated learning is slightly higher than centralized learning in F1-score. A similar phenomenon occurs in the performance of x-error-close, z-error-close and z-error-far, the result of federated learning is slightly better than centralized learning. Regarding x-error-far, the performance of federated learning is slightly weaker but also similar. By comparing the results, we can see that federated learning can achieve similar results to centralized learning while the client does not share their local raw data but their model parameters.

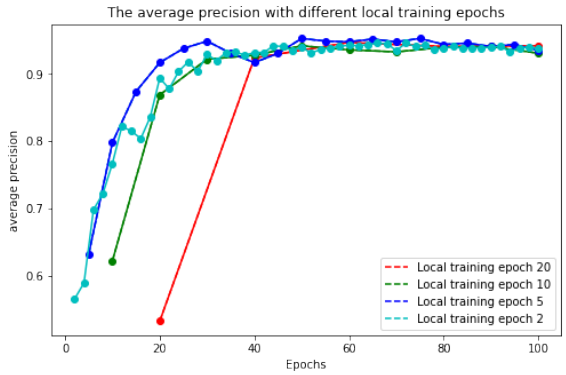
There is a commonality between centralized and federated learning through the Table 4.3 : the closer the 3D lane to the vehicle, the more accurate the prediction results. From the dataset perspective, the lanes close to the vehicle have more accurate manual 2D annotations for the semi-auto dataset. Furthermore, the generated 3D annotations are more accurate. For the HD map dataset, the sensor error is amplified as the distance increases, resulting in more significant errors in the generated 3D annotations in the distance. From the model perspective, since the model's input is an image, the area farther from the vehicle has a smaller proportion of the image, making it more difficult to extract features. Combining the above two reasons, the trained model has better prediction results for lanes closer to the vehicle.

Table 4.3: The result of centralized training and decentralized training (local training epoch is 5) with semi-auto dataset.

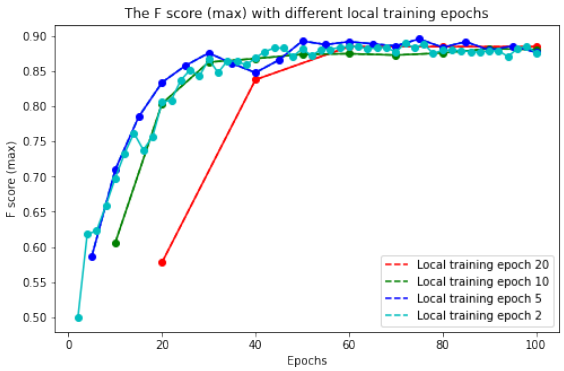
	Centralized training	**Federated training**
AP ↑	0.953	0.953
F1 score (max) ↑	0.894	0.896
x-error-close ↓	0.237	0.225
x-error-far ↓	0.513	0.521
z-error-close ↓	0.078	0.068
z-error-far ↓	0.227	0.188

Table 4.4: The best result on validation set of federated training with semi-auto dataset with different local training epochs and number of communication rounds.

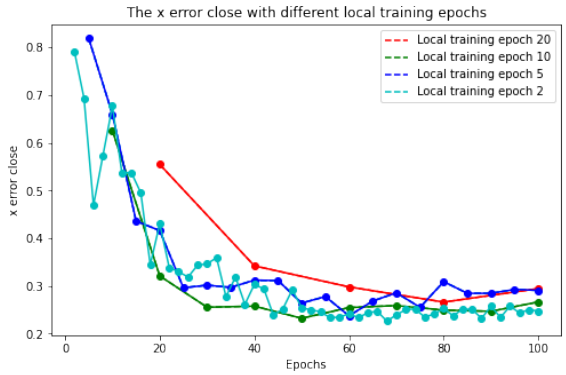
	2 epochs, 50 rounds	**5 epochs, 20 rounds**	**10 epochs, 10 rounds**	**20 epochs, 5 rounds**
AP ↑	0.946	0.953	0.941	0.947
F1 score (max) ↑	0.890	0.896	0.882	0.885
x-error-close ↓	0.250	0.225	0.247	0.298
x-error-far ↓	0.506	0.521	0.518	0.519
z-error-close ↓	0.078	0.068	0.084	0.081
z-error-far ↓	0.227	0.188	0.254	0.190



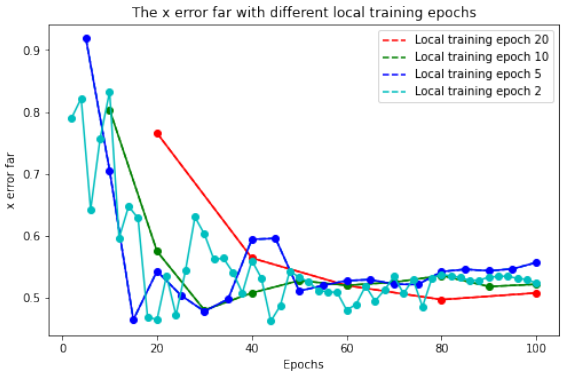
(a) Average Precision (AP)



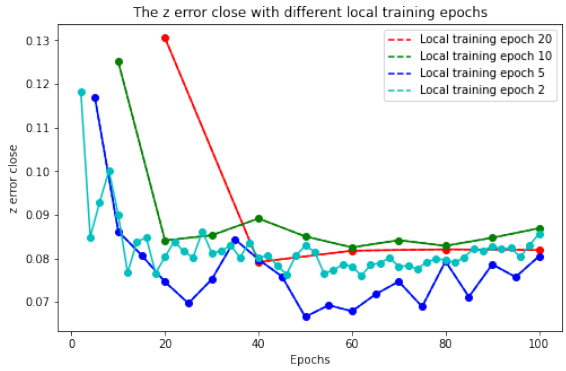
(b) F1 score (max)



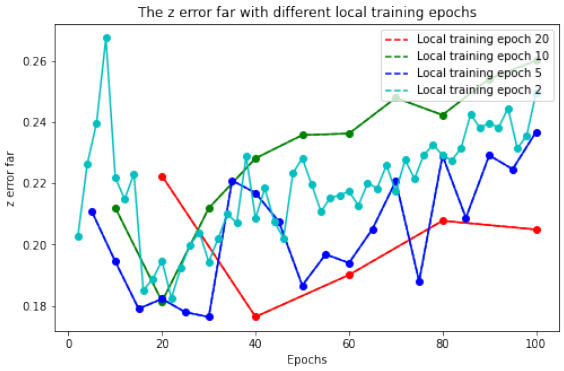
(c) x-error-close



(d) x-error-far



(e) z-error-close



(f) z-error-far

Figure 4.4: The performance of the model under different local training epochs with semi-auto dataset.

The Figure 4.4 shows the performance of the federated experiments with the same total training epochs of 100 but with different local training epochs. The horizontal axis is the number of epochs during the model training, and the vertical axis is the performance of the global model on the validation set at the current epoch. The Table 4.4 shows the best results on validation set of 4 federated experiments. By comparing the different curves in the Figure 4.4 and the different results in the Table 4.4, we can find the impact of different communication rounds and local epochs on

the global model performance.

In x-error-close and x-error-far, we can find that the value of error decreases accordingly with the increase of epochs. The higher the communication rounds, the greater the fluctuation, and vice versa. The fluctuation of x-error-far is more significant than that of x-error-close because the value of x in the relatively far distance is more difficult to predict. The same situation occurs in z-error-close and z-error-far. In particular, the curve of the z-error-far is more divergent because the height information changes with the terrain fluctuation, and its features are not obvious in the input image, so it is not easy to predict the accurate z value of far distance.

From the average precision and F1 score (max), we can find that the sky blue curve with the highest communication rounds fluctuates more than other curves. Since the higher communication round meaning that the local models in the clients are not yet enough converged towards the optimal solution. Therefore, the global model obtained by aggregating the client models does not necessarily point to the optimal solution.

Due to the lower communication rounds for the other three curves, the client models have more local training epochs to converge toward the optimal solution. Comparing the average precision and F1 score (max) of these three curves, when the total epochs are the same, the higher the communication rounds, the better performance of the global model. However, after the model gradually converges, we find that higher communication rounds do not mean a higher score. For example, the curve with local training epochs of 5 is better than other curves in average precision and F1 score (max), including the curve with local training epochs of 2. Table 4.4 also shows that the best results occur under the 5 local training epochs (20 rounds). This means that when we balance the communication rounds and communication overhead of federated learning, we do not need to maintain high communication rounds but determine the proper communication rounds according to the convergence of the model, the communication overhead, and the size of the training set.

4.3 Centralized Training with HD map Dataset

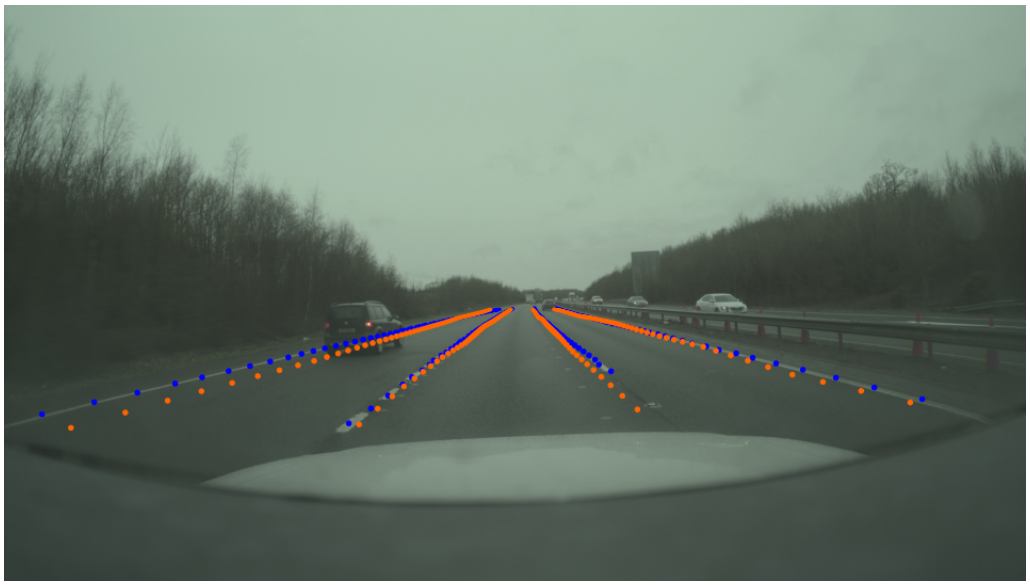
The Table 4.5 shows the results of centralized training using the semi-auto dataset and the HD map dataset. Since the same validation set is used for both experiments, a reasonable evaluation of the model trained with both datasets is possible. Table 4.5 shows that the model trained with the HD map dataset does not perform as well on the validation set as the model trained with the semi-auto dataset. The average precision and F1 score (max) of the model trained with the HD map dataset are relatively low, and the x-error-close is significantly worse than the model trained with the semi-auto dataset. This is due to the annotation error introduced in generating the HD map dataset.

Table 4.5: The result of centralized training with semi-auto dataset and HD map dataset.

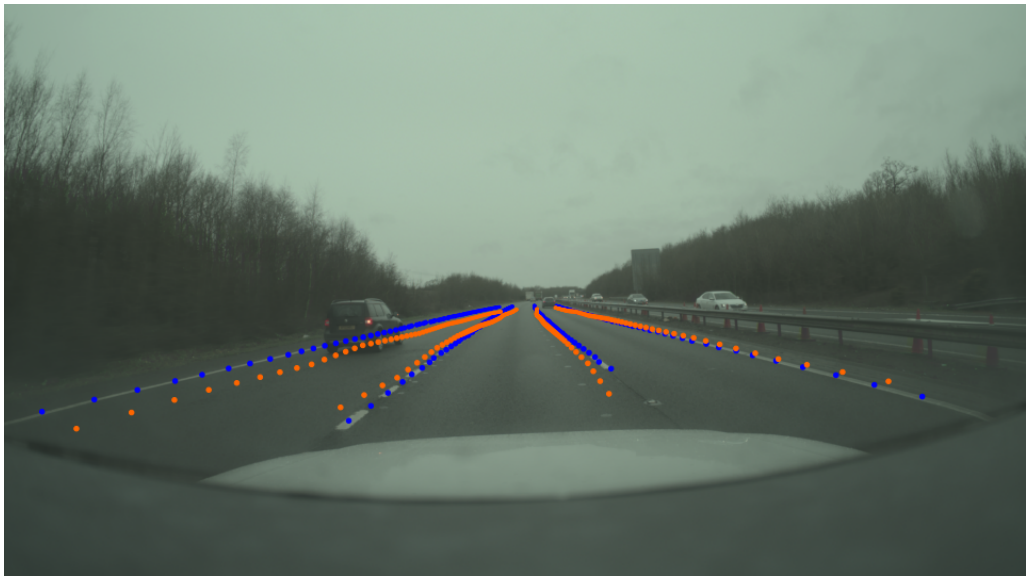
	semi-auto dataset	**HD map dataset**
AP ↑	0.953	0.909
F1 score (max) ↑	0.894	0.864
x-error-close ↓	0.237	0.410
x-error-far ↓	0.513	0.598
z-error-close ↓	0.078	0.117
z-error-far ↓	0.227	0.478

Figure 4.5 is the reprojection of the predictions (orange points) obtained by the model trained on the semi-auto dataset and the HD map dataset on the image. The predictions of the model trained on the HD map dataset have more significant deviations from the ground truth (blue points), especially discontinuous lane lines, for example, the third lane from left to right.

For the lane lines close to ego vehicle, the prediction of the model trained on the semi-auto dataset are more consistent with the ground truth, while the predictions of the model trained on the HD map dataset are more biased. The same is true for the prediction of far lanes. This is due to annotation bias when generating 3D annotations through the fully-automatic method, so the trained model performs relatively poorly predicting distant lanes. Therefore, as the 3D annotation error of the training set decreases, the performance of the model will be further improved.



(a) Prediction of model trained by semi-auto dataset



(b) Prediction of model trained by HD map Dataset

Figure 4.5: Ground truth (blue points) and predictions (orange points) from model trained by semi-auto dataset and HD map dataset.

4.4 Federated Learning With HD Map Dataset

In this report, we want to explore the possibility of generating annotation on edge devices (i.e., autonomous vehicles) and training models on edge nodes. So if the model can be properly trained with the fully automatic dataset in the federated learning way, then model training can be done without human intervention and data sharing on client nodes. The Table 4.6 shows the result of the centralized model with semi-auto dataset and federated learning using semi-auto dataset and HD map dataset.

Table 4.6: The result of centralized training with semi-auto dataset and HD map dataset and decentralized training with HD map dataset.

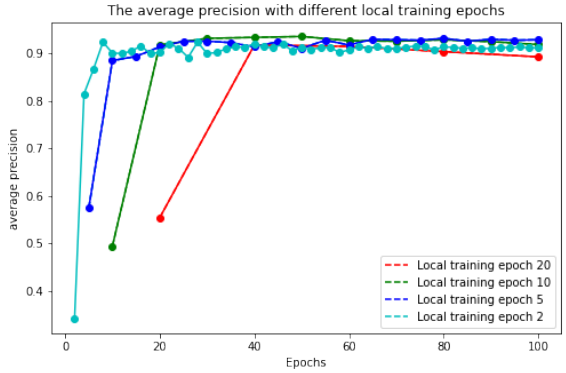
	Centralized training with semi-auto dataset	**Centralized training with HD map dataset**	**Decentralized training with HD map dataset**
AP ↑	0.953	0.909	0.929
F1 score (max) ↑	0.894	0.864	0.894
x-error-close ↓	0.237	0.410	0.456
x-error-far ↓	0.513	0.598	0.597
z-error-close ↓	0.078	0.117	0.112
z-error-far ↓	0.227	0.478	0.298

Table 4.5 shows that when the training set uses the HD map dataset, the model trained by federated learning performs better than the model trained by centralized learning except for x-error-close metric. Although the model trained in federated learning using the HD map dataset performs slightly lower according to AP, x-error-close, z-error-close, z-error-far than the model obtained under optimal conditions (centralized training and semi-auto data). However, these gaps will decrease as the quality of 3D annotations in the HD map dataset improves. In this set of experiments, we use the HD map dataset to train the model in federated learning to demonstrate the feasibility of directly generating 3D annotation on edge devices and training the global model in federated learning.

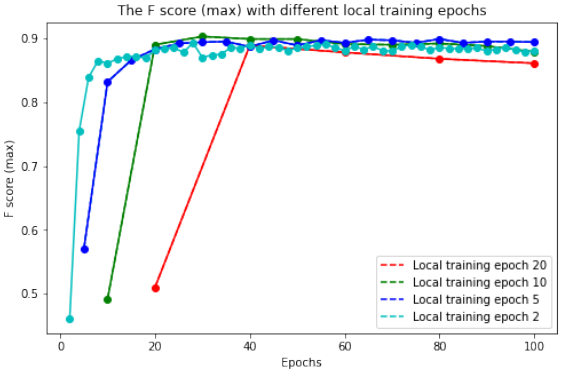
To find the best setup for federated Learning, we also explore the influence of different communication rounds on the final result when the total training epochs are the same, so here we set the local training epochs at clients to [20, 10, 5, 2], respectively. Table 4.7 shows the best results on validation set of 4 federated experiments. And Figure 4.6 shows the model performance of the federated experiments with the same total training epochs of 100 but different local training epochs.

Table 4.7: The result of federated training with HD map dataset in different local training epochs.

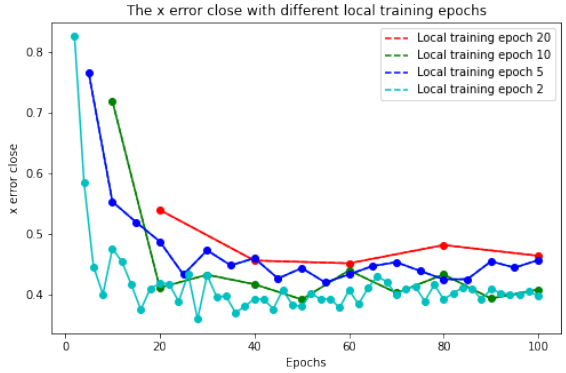
	2 epochs, 50 rounds	**5 epochs, 20 rounds**	**10 epochs, 10 rounds**	**20 epochs, 5 rounds**
AP ↑	0.914	0.929	0.919	0.918
F1 score (max) ↑	0.882	0.894	0.879	0.889
x-error-close ↓	0.399	0.456	0.407	0.456
x-error-far ↓	0.584	0.597	0.648	0.630
z-error-close ↓	0.111	0.112	0.114	0.113
z-error-far ↓	0.281	0.298	0.283	0.276



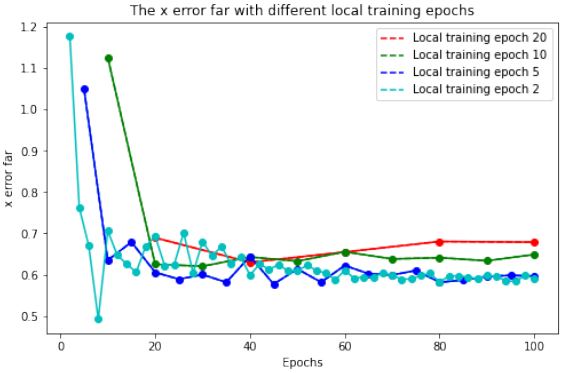
(a) Average precision



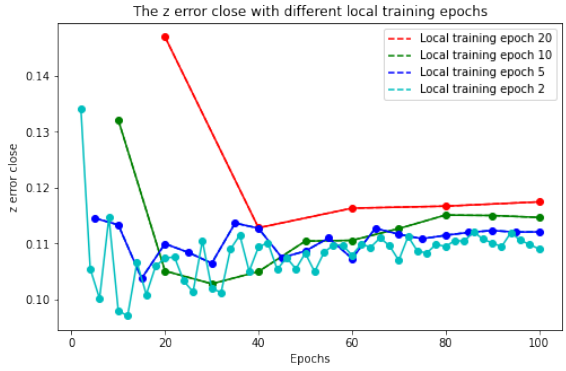
(b) F1 score(max)



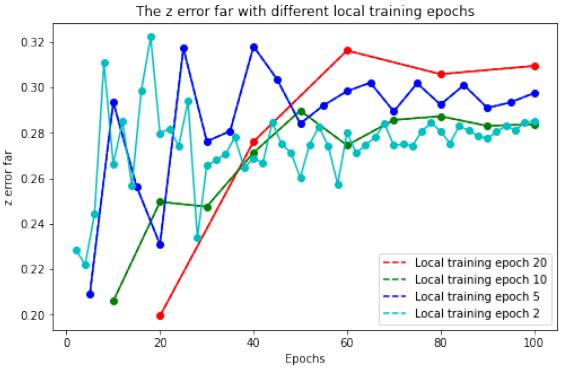
(c) x-error-close



(d) x-error-far



(e) z-error-close



(f) z-error-far

Figure 4.6: The performance of the model under different local training epochs with HD map dataset.

Table 4.7 shows the best performance of the model on the validation set as the changing of local training epochs. For average precision and F1 score (max), as the communication rounds increases (the local training epoch decreases), the results of both metrics become better. But the best model performance occurs at local training epochs is 5, not training epochs is 2. However, several other metrics, such as x-error-close, x-error-far, z-error-close, z-error-far, achieve the best performance when the local training epochs is 2. This means that the choice of communication

rounds and local training epochs in a federated setting needs to take into account the metrics used.
Through the curves of the six metrics in the Figure 4.6 we can find a commonality that the higher the communication rounds, the greater the fluctuation of the curve. This is because the convergence direction of the edge model does not entirely point to the optimal solution. Taking the average precision at 20 epochs as an example, in the early stage of model training, the more communication rounds, the higher average precision. Since more communication in the early stage can enable the global model to learn the features of all clients. However, in the later stage of training, both the average precision and the F1 score (max) achieve the best results when the local training epoch is 5. Table 4.7 also shows that the best average precision and F1 score (max) appear in the local training epochs is 5. Therefore, in federated learning, we do not need to pursue a high communication rounds but should consider different factors such as communication overhead, and the size of the training set. For the x-error-close, x-error-far and z-error-close metrics, the model's performance is also similar, that is, as the communication rounds increases, the model converges faster. However, for the z-error-far, the models of the four experiments are divergent since the distant height features are not obvious in the input image.
From these experiments we think that a good practice in a federated setting could be to have more frequent communication rounds and shorter local trainings in the beginning and as the federation process continues, have less frequent rounds of communication, and consequently slightly longer local epochs but still not very rare communication rounds, to maximize the training speed and performance.

5
Conclusion

This thesis has introduced a pipeline for automatically labeling 3D lane annotations in the camera frame utilizing GPS, IMU, and an HD map. The results show that auto-labeled 3D lane marking annotations can be accurate up to 100 meters ahead of the vehicle compared to the referencing labels generated by combining manually labeled 2D lane annotations with projected depth maps from aggregated LiDAR point clouds. This method enables proper training of supervised 3D lane detection models on edge devices by using fully automatically generated annotations and achieve promising performance. This work also explores the performance of models trained in federated learning (FL) settings. The results show that the performance of the model trained in FL settings under the FedAvg strategy is very close to the one trained in the traditional centralized manner when using the same dataset. At the same time, this thesis also explores the impact of different communication rounds in federated learning on the performance of the global model. The results show that higher communication rounds can make the model converge faster, but higher communication rounds do not mean a better model performance. If we can compute pitch and roll at different distances range using map when we create HD map annotations, it should be possible to reduce annotation errors.
To sum up the above, after combining automatically labeling 3D lane annotations method and federated learning, it is possible to automatically label local data and train edge models on edge devices that guarantee data retention. The central server can obtain a model comparable to centralized training by aggregating edge model parameters and updating edge model parameters.

5.1 Future Work

Our proposed HD map automatic labeling pipeline can currently perform decent annotation on data containing ideal sensor records. However, as shown in 4.1, there is still a chance that false labels will be generated. Except for errors caused by incomplete maps, most errors are caused by inaccurate vehicle positioning or vehicle orientation information provided by the IMU. In addition to using passive schemes to exclude these inaccurate data, some active strategies can be experimented with, such as dynamic calibration and visual odometry. For instance, in [14], they found the transformation between vehicle and camera ${}^{C}_{V}\mathbf{T}$ contains the highest amount of error of the transforms needed for the labeling pipeline; therefore, they adopted a method of estimating a correction by viewing a series of images from a recording and manually testing different corrections to improve the final projection. This method

is not scalable because it requires more than 100,000 samples per vehicle and manual selection. However, it enlightens us that we can use some learnable models to estimate these correction parameters. In addition, introducing more sensor data and further sensor fusion should also improve the annotation quality to a certain extent.

Our project simulates the simplest federated learning scenario, where only two client devices and one central server participate in the learning process. The computing resources are physically close to each other (in the same computer cluster). Experimenting with more clients participating in the training and a real distributed scenario should be exciting and meaningful. In this case, new problems such as communication difficulties could arise. It would also be interesting to experiment with data collected by clients in various environments instead of only on highways, which is more challenging.

When more and more clients participate in the process, they will generate more data, then other properties of this data could also be valuable, such as the time attribute. Therefore, we hope to see further research on exploring self-supervised learning to utilize the massively unlabeled data at the edge (streams of data or videos) to improve the final performance of the models. For instance, in [44], they proposed to use the property of Spatio-Temporal Consistency in videos to train a 3D lane detection model in a semi-supervised manner, and they successfully demonstrated that their model could indeed be improved in the case of limited labeled data.

Bibliography

[1] N. H. T. S. Administration *et al.*, "2015 motor vehicle crashes: Overview," *Traffic safety facts: research note*, vol. 2016, pp. 1–9, 2016.

[2] S. Singh, "Critical reasons for crashes investigated in the national motor vehicle crash causation survey," Tech. Rep., 2015.

[3] A. Bar Hillel, R. Lerner, D. Levi, and G. Raz, "Recent progress in road and lane detection: A survey," *Machine vision and applications*, vol. 25, no. 3, pp. 727–745, 2014.

[4] T. Gupta, H. S. Sikchi, and D. Charkravarty, "Robust lane detection using multiple features," in *2018 IEEE intelligent vehicles symposium (IV)*, IEEE, 2018, pp. 1470–1475.

[5] S. Kammel and B. Pitzer, "Lidar-based lane marker detection and mapping," in *2008 ieee intelligent vehicles symposium*, IEEE, 2008, pp. 1137–1142.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[7] D. Neven, B. De Brabandere, S. Georgoulis, M. Proesmans, and L. Van Gool, "Towards end-to-end lane detection: An instance segmentation approach," in *2018 IEEE intelligent vehicles symposium (IV)*, IEEE, 2018, pp. 286–291.

[8] D. Chang, V. Chirakkal, S. Goswami, M. Hasan, T. Jung, J. Kang, S.-C. Kee, D. Lee, and A. P. Singh, "Multi-lane detection using instance segmentation and attentive voting," in *2019 19th International Conference on Control, Automation and Systems (ICCAS)*, IEEE, 2019, pp. 1538–1542.

[9] N. Garnett, R. Cohen, T. Pe'er, R. Lahav, and D. Levi, "3d-lanenet: End-to-end 3d multiple lane detection," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 2921–2930.

[10] Y. Guo, G. Chen, P. Zhao, W. Zhang, J. Miao, J. Wang, and T. E. Choe, "Gen-lanenet: A generalized and scalable approach for 3d lane detection," in *European Conference on Computer Vision*, Springer, 2020, pp. 666–681.

[11] N. Efrat, M. Bluvstein, S. Oron, D. Levi, N. Garnett, and B. E. Shlomo, "3d-lanenet+: Anchor free lane detection using a semi-local representation," *arXiv preprint arXiv:2011.01535*, 2020.

[12] M. Paluri, M. Paluri, D. Mahajan, R. Girshick, and V. Ramanathan, *Advancing state-of-the-art image recognition with deep learning on hashtags*, Mar. 2020. [Online]. Available: `https://engineering.fb.com/2018/05/02/ml-applications/advancing-state-of-the-art-image-recognition-with-deep-learning-on-hashtags/`.

[13] N. Garnett, R. Uziel, N. Efrat, and D. Levi, "Synthetic-to-real domain adaptation for lane detection," in *Proceedings of the Asian Conference on Computer Vision*, 2020.

[14] K. Behrendt and R. Soussan, "Unsupervised labeled lane markers using maps," in *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, 2019, pp. 0–0.

[15] D. ISO *et al.*, "Road vehicles–vehicle dynamics and road-holding ability–vocabulary," 2013.

[16] R. Liu, J. Wang, and B. Zhang, "High definition map for automated driving: Overview and analysis," *Journal of Navigation*, vol. 73, no. 2, pp. 324–341, 2020. DOI: 10.1017/S0373463319000638.

[17] A. Zang, X. Chen, and G. Trajcevski, "High definition maps in urban context," *Sigspatial Special*, vol. 10, no. 1, pp. 15–20, 2018.

[18] J. Jiao, "Machine learning assisted high-definition map creation," in *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, IEEE, vol. 1, 2018, pp. 367–373.

[19] R. Matthaei and M. Maurer, "Autonomous driving–a top-down-approach," *at-Automatisierungstechnik*, vol. 63, no. 3, pp. 155–167, 2015.

[20] S. Ulbrich, A. Reschka, J. Rieken, S. Ernst, G. Bagschik, F. Dierkes, M. Nolte, and M. Maurer, "Towards a functional system architecture for automated vehicles," *arXiv preprint arXiv:1703.08557*, 2017.

[21] Z. Zou, Z. Shi, Y. Guo, and J. Ye, "Object detection in 20 years: A survey," *arXiv preprint arXiv:1905.05055*, 2019.

[22] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, Ieee, vol. 1, 2005, pp. 886–893.

[23] P. Felzenszwalb, D. McAllester, and D. Ramanan, "A discriminatively trained, multiscale, deformable part model," in *2008 IEEE conference on computer vision and pattern recognition*, Ieee, 2008, pp. 1–8.

[24] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[25] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2117–2125.

[26] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[27] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*, Springer, 2016, pp. 21–37.

[28] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.

[29] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun, "Yolox: Exceeding yolo series in 2021," *arXiv preprint arXiv:2107.08430*, 2021.

[30] P. Voigt and A. Von dem Bussche, "The eu general data protection regulation (gdpr)," *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, vol. 10, no. 3152676, pp. 10–5555, 2017.

[31] J. Konečn, H. B. McMahan, D. Ramage, and P. Richtárik, "Federated optimization: Distributed machine learning for on-device intelligence," *arXiv preprint arXiv:1610.02527*, 2016.

[32] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 2, pp. 1–19, 2019.

[33] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*, PMLR, 2017, pp. 1273–1282.

[34] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, *et al.*, "Advances and open problems in federated learning," *Foundations and Trends® in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

[35] T.-T. Tran, C.-S. Bae, Y.-N. Kim, H.-M. Cho, and S.-B. Cho, "An adaptive method for lane marking detection based on hsi color model," in *International Conference on Intelligent Computing*, Springer, 2010, pp. 304–311.

[36] J. Wang, Y. Wu, Z. Liang, and Y. Xi, "Lane detection based on random hough transform on region of interesting," in *The 2010 IEEE International Conference on Information and Automation*, IEEE, 2010, pp. 1735–1740.

[37] K. Ghazali, R. Xiao, and J. Ma, "Road lane detection using h-maxima and improved hough transform," in *2012 Fourth International Conference on Computational Intelligence, Modelling and Simulation*, IEEE, 2012, pp. 205–208.

[38] B. Huval, T. Wang, S. Tandon, J. Kiske, W. Song, J. Pazhayampallil, M. Andriluka, P. Rajpurkar, T. Migimatsu, R. Cheng-Yue, *et al.*, "An empirical evaluation of deep learning on highway driving," *arXiv preprint arXiv:1504.01716*, 2015.

[39] B. He, R. Ai, Y. Yan, and X. Lang, "Accurate and robust lane detection based on dual-view convolutional neutral network," in *2016 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2016, pp. 1041–1046.

[40] J. Kim and C. Park, "End-to-end ego lane estimation based on sequential transfer learning for self-driving cars," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2017, pp. 30–38.

[41] L. Tabelini, R. Berriel, T. M. Paixao, C. Badue, A. F. De Souza, and T. Oliveira-Santos, "Polylanenet: Lane estimation via deep polynomial regression," in *2020 25th International Conference on Pattern Recognition (ICPR)*, IEEE, 2021, pp. 6150–6156.

[42] F. Yan, M. Nie, X. Cai, J. Han, H. Xu, Z. Yang, C. Ye, Y. Fu, M. B. Mi, and L. Zhang, "Once-3dlanes: Building monocular 3d lane detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 17 143–17 152.

[43] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, T. Parcollet, P. P. de Gusmão, and N. D. Lane, "Flower: A friendly federated learning research framework," *arXiv preprint arXiv:2007.14390*, 2020.

[44] E. Brorsson and S. Ulander, "Semi-supervised 3d lane detection through spatio-temporal consistency learned from videos," 2021. [Online]. Available: `https://hdl.handle.net/20.500.12380/304236`.

[45] H. A. Mallot, H. H. Bülthoff, J. Little, and S. Bohrer, "Inverse perspective mapping simplifies optical flow computation and obstacle detection," *Biological cybernetics*, vol. 64, no. 3, pp. 177–185, 1991.

A

Appendix 1

A.1 Vehicle Axis System

When computing the rotation matrix ${}^{V}_{L}\mathbf{T}$ from the east-north-up local Cartesian system and ego-vehicle system, three vehicle dynamics describing the orientation of the current vehicle are needed. Following the ISO 8855 standard, the vehicle axis frame is shown in Figure A.1. This frame is attached to the vehicle and rotates with it in all three axes with the heading/yaw, pitch and roll of the vehicle.

- The x-axis is parallel to the vehicle's heading and is in the forward direction of the vehicle.
- The y-axis is perpendicular to the X axis and is in the left direction of the vehicle.
- The z-axis is perpendicular to the X and Y axes and is in the up direction of the vehicle.

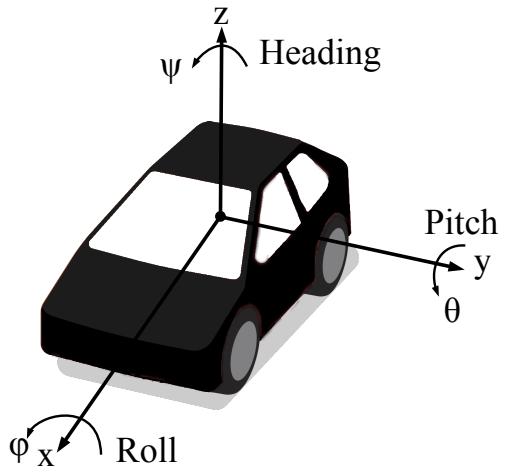


Figure A.1: Vehicle axis system ISO 8855.

The rotation matrix ${}^{V}_{L}\mathbf{T}$ is built by multiplying three rotation matrices, each representing a rotation around one of the coordinate systems axes. The rotation matrices of the x-, y-, and z-axis are computed as follows.

$$\mathbf{R}_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix}, \tag{A.1}$$

$$\mathbf{R}_y = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}, \tag{A.2}$$

$$\mathbf{R}_z = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{A.3}$$

The rotations are applied in the order of z, y, x to compute the final rotation ${}^{V}_{L}\mathbf{T}$ as

$${}^{V}_{L}\mathbf{T} = \mathbf{R}_z\mathbf{R}_y\mathbf{R}_x. \tag{A.4}$$

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY