CHALMERS
UNIVERSITY OF TECHNOLOGY

Scalable High-Definition Map Generation through Crowdsourcing

Creating High-Definition Maps for Autonomous Driving Using Standard-Accuracy Vehicle Fleet Data: Exploration of Two Different Approaches

Yubo Tian & Runjia Qian

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY

Gothenburg, Sweden 2023

www.chalmers.se

Master's thesis 2023

Scalable High-Definition Map Generation through Crowdsourcing

Creating High-Definition Maps for Autonomous Driving Using Standard-Accuracy Vehicle Fleet Data: Exploration of Two Different Approaches

Yubo Tian & Runjia Qian

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2023

Scalable High-Definition Map Generation through Crowdsourcing
Creating High-Definition Maps for Autonomous Driving Using Standard-Accuracy Vehicle Fleet Data: Exploration of Two Different Approaches
Yubo Tian & Runjia Qian

Supervisor: Erik Stenborg, Junsheng Fu, Gabriel Garcia Jaime, Zenseact
Examiner: Lars Hammarstrand, Department of Electrical Engineering

Master's Thesis 2023
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: The image displays a satellite view of a section of Gothenburg's highway, with red lines denoting crowdsourced lane markings. These red lines distinctly differentiate between solid and dashed lane delineations.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2023

Scalable High-Definition Map Generation through Crowdsourcing
Creating High-Definition Maps for Autonomous Driving Using Standard-Accuracy Vehicle Fleet Data: Exploration of Two Different Approaches
Yubo Tian & Runjia Qian
Department of Electrical Engineering
Chalmers University of Technology

Abstract

The creation and update of high-definition (HD) maps have become vital for advanced applications like autonomous vehicles, yet traditional map-making techniques face inherent limitations in cost, efficiency, and scalability. Given the same challenge and goal, this thesis explores two distinct but parallel approaches, one optimization-based and one learning-based method, both leveraging the wealth of crowdsourced data to enhance HD map construction and update processes.

The optimization-based method utilizes a three-stage process involving the handling of raw data, the alignment and optimization of maps from multiple drives including lane marker and traffic sign alignment, and the regular generation and updating of maps. Alternatively, the learning-based method employs a two-stage network that uses a vision transformer as a crowdsourced data aggregator and a detection transformer for decoding individual road elements, linking neighboring frames using pseudo labels. In conclusion, both approaches contribute to ongoing efforts to improve the robustness and cost-effectiveness of HD map creation and updates. They also demonstrate scalability, specifically in their ability to integrate data from multiple vehicles into a cohesive model. When evaluated for accuracy, both the optimization-based and learning-based methods yield impressive results, achieving an approximate accuracy of 1.0 meters. Notably, the optimization-based method can achieve an even higher level of accuracy, reaching up to 0.7 meters in assessments.

Keywords: Crowdsourcing, SLAM, Transformer, Data Association, HD Map

Acknowledgements

First and foremost, I would like to express my gratitude to Zenseact for granting us this invaluable opportunity and resources, enabling us to delve deep into the realm of map-building using crowdsourced data. Secondly, much appreciation goes to Erik, Gabo, and Junsheng for their support throughout this project. Whenever challenges arose, they actively communicated with us and proposed solutions, even sacrificing their personal time. Special thanks to Lars for his insightful suggestions on our research direction. When our research got stuck, Lars would share some helpful readings that got us moving again. Lastly, a sincere thank you to my girlfriend, Vicky. During the toughest times, both mentally and physically, she stood by me, constantly offering her support and encouragement. Even though we were miles apart, I always felt close to her in heart.

Yubo Tian, Gothenburg, Aug 2023

I would like to thank Zenseact for giving me this opportunity to finish this work, and, most important, giving abundant GPU resources for training the neural network. During the completion of the thesis, Lars gave me a lot of practical opinion for the completion of learning based neural network, Erik, Junsheng and Gabriel also provided a lot of support when I encounter problems like datasets. I would also thanks to my girlfriend and my parents, although we can't meet, but the emotional support they provided me helped me to cheer up every time when I failed. I also want to thank my thesis project teammate. At the halfway point of our thesis project, my deep learning model was struggling to train effectively. During this time, I may have put a lot of pressure on them with the possibility of not graduating, but they still worked with me to complete these tasks.

Runjia Qian, Gothenburg, Aug 2023

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

HD	High Definition
SD	Standard Definition
HAD	Highly Automated Driving
IMU	Inertial Measurement Unit
SLAM	Simultaneous Localization and Mapping
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
VO	Visual Odometry
VIO	Visual Inertial Odometry
L-M	Levenberg-Marquardt
G-N	Gauss-Newton
CUDA	Renewable-based Energy Sources
CNN	Convolutional Neural Network
MHA	Multi-Head Attention
ViT	Vision Transformer
DETR	Detection Transformer
AP	Average Precision
GPU	Graphics Processing Unit

Contents

List of Figures

1

Introduction

This chapter starts with a background on HD map construction, covering both its historical evolution and current methods relevant to our study. Next, we discuss the main goals of this research and what it aims to contribute. We then review existing literature to position our research within the broader academic discussions on this topic. Finally, we define the limits and scopes of our study. These sections provide readers with a clear understanding of the foundational topics and research focus that will be expanded upon in the following chapters.

1.1 Background

High-definition (HD) map, or Highly Automated Driving (HAD) map, refers to an electronic map that has high accuracy, freshness, and richness, with both absolute and relative precision within 1 meter. The information contained in an HD map is extensive, including road characteristics such as road type, curvature, and lane line positions, as well as environmental object information such as roadside infrastructure, obstacles, and traffic signs. Additionally, it can incorporate real-time dynamic information such as traffic flow and traffic light status, when available. HD maps provide essential location-information layers that enable the automated driving system to achieve safer driving and more proactive driving decisions [1].

The maps commonly used in GPS navigation systems are currently referred to as standard-definition (SD) maps [1]. The main difference between HD maps and SD maps lies in the level of detail and precision they provide. SD maps provide the basic layout of roads and intersections, as well as some features such as points of interest. However, they lack the detailed information and precision of HD maps. For example, an SD map may indicate the presence of a road, but it cannot provide details such as the number of lanes, the exact positions of lane markings, or the precise locations of traffic signals, which are crucial for autonomous driving. Knowing the number of lanes on the road helps autonomous vehicles determine their position, plan lane changes, and understand applicable traffic rules. Certain maneuvers like turning may only be allowed from specific lanes. Additionally, knowing the exact locations of traffic lights and traffic signs allows autonomous vehicles to stop or decelerate accurately, preventing incidents such as running a red light or stopping too far from an intersection, which can lead to accidents. Essentially, HD maps are designed for machines (autonomous vehicles) to have a more nuanced understanding of the environment. Meanwhile, SD maps are primarily intended for humans to gain a

general understanding of road layouts and surrounding areas, although they can also serve as a source of information for certain ADAS functions.

HD maps can play an important role in autonomous driving. However, their purpose is not to replace the vehicle's onboard sensors but to complement them. Onboard sensors provide real-time data about the surrounding environment, such as other vehicles, pedestrians, or obstacles, while high-definition maps offer a detailed, long-range static overview of the environment. The introduction of HD map data aims to enhance safety in a broader range of driving scenarios. Combining information from both sources allows autonomous vehicles to navigate more safely and efficiently.

The construction of HD maps typically involves the following components and processes [2]:

- Data collection: This is the first step in the map creation process and involves collecting a large amount of data about the geographical environment. Data can come from various sources such as satellite imagery, ground surveys, onboard sensors, and user-generated data.
- Data pre-processing: The collected raw data needs to be cleaned and formatted for further analysis. This may involve filtering out irrelevant or erroneous data, converting data formats, and normalizing the data.
- Feature extraction: In this step, map information needs to be extracted from the pre-processed data. For example, determining road positions from GPS data or identifying traffic signs from camera images.
- Map construction: This step involves creating the map based on the extracted feature information. This may require the use of complex algorithms and significant computational resources.
- Map validation and updates: The created map needs to undergo validation and testing to ensure its accuracy and reliability. Additionally, the map needs to be regularly updated to reflect changes in the geographical environment.

Map construction has seen a considerable evolution over time, particularly in the distinction between traditional and modern techniques [2]. Both these methods come with their own set of advantages and disadvantages, which significantly influence the efficiency, cost, and overall quality of the produced maps.

Traditional methods of map construction, such as manual ground surveying or utilizing surveying vehicles equipped with HD tools like LiDAR and high-accuracy positioning equipment, certainly bring a high degree of precision. However, they are constrained by several factors. The extensive data collection required from every road, for instance, can be a challenging task due to limited human resources and surveying vehicles. Besides, these methods demand significant human effort for annotating and ensuring the quality control of the newly collected feature information. As a result, these constraints tend to limit the scale of HD maps that can be produced, rendering the process both resource-heavy and labor-intensive.

Modern mapping methods, on the other hand, leverage technology to overcome these limitations, with crowdsourcing being a prime example. Crowdsourced mapping presents an efficient and cost-effective alternative to traditional techniques. It

utilizes the vast amount of vehicles already present on the roads, ensuring broader area coverage, even in regions that might be overlooked by traditional mapping. Furthermore, it capitalizes on real-time data, which allows for quick and timely updates on map changes, such as the construction of new roads or changes in traffic regulations. In addition, despite offering extensive coverage and real-time updates, the costs associated with data collection in crowdsourcing are significantly lower than traditional methods, making it a more economical choice.

In conclusion, crowdsourcing, as an exemplar of modern mapping techniques, addresses the challenges posed by traditional mapping methods. It efficiently uses readily available resources and offers a cost-effective and time-efficient approach to map production. This method ensures the production of comprehensive and accurate maps without overburdening resources.

1.2 Aim & Objective

The main goal of our research is to develop effective methods for processing vehicle data covering large areas and long distances and finally build a HD Map for autonomous driving. To achieve this goal, we explore two principal methodologies for constructing HD maps using crowdsourced data: optimization-based approach and learning-based approach.

The optimization-based approach is a traditional method that has been widely used in the field of autonomous driving, effectively processing data from multiple sensors to generate accurate environmental models. In contrast, the learning-based approach is a newer method that has made significant progress in recent years, distinguished by its ability to learn from data and consequently improve performance.

In this project, we will explore the two approaches separately. We will design and implement two different solutions for each approach. Then, real vehicle crowdsourced data will be used to test these two solutions. Finally, the results obtained from the two methods will be compared with the ground truth to evaluate their performance.

Following are the detailed research questions that emanate from these objectives:

- Given limited hardware resources, how can the visual-inertial odometry maintain high accuracy over long distances (e.g, $\geq$ 10 km) and long period (e.g, $\geq$ 30 min)? If the problem needs to be split up into smaller parts, how to merge all the smaller parts while still keep the smoothness of the trajectory and map?
- How should GNSS information be fused with other sensor information? How to ensure that the vehicle trajectory output from the odometry does not deviate significantly when GNSS information is missing or unstable?
- How much gain in accuracy can deep learning methods make in comparison to existing methods on aggregation for HD-map? How can we construct a deep learninng network to work well with summarizing the crowdsourced data, which, different from images and texts, as a data in the form of the set of multiple description of the same object?

1.3 Related Work

The realm of autonomous navigation and mapping is continually shaped by innovative methodologies and research paradigms. This section aims to explore two notable domains that have garnered attention in recent years: Visual SLAM and Deep Learning methods in the context of localization and mapping. Visual SLAM seeks to utilize visual data for concurrent localization and map construction, an approach that has been the subject of extensive research and various implementations. Its evolution is characterized by iterative refinements influenced by advancements in sensors, algorithms, and computational capacities. Meanwhile, the application of deep learning in localization and mapping emerges from the broader field of machine learning. By processing substantial volumes of data, it offers potential avenues for feature extraction and environment adaptability, albeit in ways distinct from traditional methods. The ensuing subsections will present an overview of pertinent studies and developments within these two areas, underscoring their respective contributions and challenges in the broader spectrum of HD mapping.

1.3.1 Visual SLAM

Over recent decades, Visual Simultaneous Localization and Mapping (SLAM) has evolved significantly, holding a pivotal role in robotics and autonomous driving scenarios. Essentially, Visual SLAM operates to compute a device's position within an environment while concurrently mapping its surroundings using primarily visual data.

A foundational aspect of visual SLAM is Visual Odometry (VO). VO is the technique of estimating the relative position of a device by analyzing sequential visual input. Within VO, there are methods that rely exclusively on cameras and others that couple visual data with inertial information. Traditional visual-based SLAM techniques, predominantly applied in confined indoor settings, hinge on the extraction of geometric features such as sparse points, lines, and planes from the environment. Renowned systems like ORB-SLAM and its variations [3]–[5] have found application beyond indoor confines, demonstrating robustness even in expansive outdoor scenarios. The precision of these systems has further been enhanced through adaptations such as the inclusion of line features or leveraging semi-dense edges.

The integration of visual data with inertial sensors has ushered a new era for VO, leading to the development of Visual-Inertial Odometry (VIO). In the realm of VIO, there are two prominent frameworks: loose coupling and tight coupling. The loosely-coupled approach sees sensors, such as the IMU and the camera, functioning in isolation. Each produces its estimates, which are subsequently blended for an overarching output. Conversely, the tight coupling strategy merges raw data from both sources from the outset, yielding more accurate and cohesive estimates. Methods such as MSCKF and VINS-Mono serve as prime examples of the latter approach, jointly optimizing both visual and inertial data.

It's noteworthy to mention the varying demands of SLAM across different settings. While indoor environments, with their constraints, may be navigated with simpler

algorithms, expansive outdoor terrains necessitate more sophisticated solutions. Recognizing the intricacies of autonomous driving, modern SLAM techniques leverage road markers, traffic signals, and other static landmarks to refine vehicular positioning. The fusion of other sensory data, like Wheel Encoder and GNSS (or RTK), provides an additional layer of precision. GVINS [6], a recent innovation, exemplifies this fusion, integrating visual-inertial data with GNSS measurements.

1.3.2 Deep Learning Methods in Mapping

In the domain of high-definition mapping, various approaches are adopted. Some derive maps via semantic segmentation of aerial and satellite images [7]–[10], However, performing segmentation on aerial and satellite images suffer from interference such as occlusions from trees, vehicles, shades of buildings, different methods were proposed to against these, [11] proposed dilation and message module utilizing contextual information to address inconsistent edges in generated maps. [12] offers a deep learning framework that combines initial segmentation using a fully convolutional network (FCN) with iterative road network searches via CNN, which enhances road connectivity and centerline completeness. [13] improved topological accuracy using a multi-branch convolutional approach combined with connectivity refinement.

Methods that utilize onboard sensors, such as vehicle cameras and Lidar, are also common. For example, one approach involves semantic segmentation of images and point-cloud data, followed by projection onto the bird's-eye view (BEV) plane[14]. Notably, recent methods opt to delay semantic segmentation; they first project the encoded features onto the BEV plane and subsequently extract map data from these projected features, also facilitating vectorized map building through post-processing[15].

Map representations vary as well. Some methods render them as segmented pixels, while others employ query mechanisms to extract map elements from features, representing them as distinct map element instances. For instance, VectorMapNet generates instance-specific details by predicting a sparse set of polylines in the bird's-eye view[16]. However, this approach of sequentially decoding keypoints in a polyline can accumulate error and require more computational power. To mitigate this, recent works have proposed to simultaneously predict all keypoints in the polyline, using attention mechanisms to establish relationships between these keypoints for instance-level map creation[17], [18].

1.4 Scope

This study presents several limitations that warrant careful consideration. The first limitation concerns the data sources. All relevant vehicular data, such as GNSS, IMU, Visual Odometry, and Camera Detections, as well as the HD map used for reference, are provided by Zenseact. The reliance on a single provider could potentially introduce biases or constraints that limit the generalizability of the research outcomes.

Secondly, the project's focus is narrowed predominantly to the construction of high-

way maps. This specificity arises from the nature of the dataset, which predominantly comprises information collected from highway routes. Highways are constructed according to particular standards and are thereby more regularized compared to other road types. This focus could limit the applicability of the methodologies to more complex or irregular road networks.

While the primary focus of this research is on the examination of static road features such as lane lines and traffic signs, it's worth noting for completeness that dynamic road elements like moving vehicles or pedestrians are not included in this study. Although most readers may not expect these elements to be part of the generated maps, this constraint is mentioned to clarify the scope and potential limitations in real-world applicability.

Another noteworthy limitation lies in the two-dimensional constraints of the study. The research does not consider the third dimension, specifically elevation and height-related elements like overpasses and underpasses, thereby potentially affecting the robustness and utility of the generated HD maps.

Lastly, the methodologies employed in the study, namely the optimization-based and learning-based approaches, are each evaluated solely against a ground truth. This approach undermines the potential for gaining insights into the relative strengths and weaknesses of each methodology when they are not directly compared. This lack of a head-to-head comparison introduces a significant gap in the evaluative framework, potentially affecting recommendations for future work or real-world applications.

By explicitly acknowledging these limitations, this study aims to offer a balanced and realistic framework for interpreting the results, while also providing a roadmap for addressing these challenges in future research endeavors.

2

Theory

This chapter provides an introduction to the theoretical knowledge that will be applied in the methodology discussed in Chapter 3. The chapter begins by explaining the characteristics of different sensors and the information they provide, using the sensor data employed in this project as a starting point. Subsequently, background knowledge of coordinate systems is presented. Estimating the state of a vehicle is a longstanding and crucial topic in the field of robotics.The Graph-Based SLAM framework is a commonly used solution for simultaneously estimating the vehicle's state and landmark positions. In most practical estimation problems, outliers exist, so this chapter also covers the outlier rejection techniques utilized in this project. Optimization algorithms ensure that the SLAM problem can be solved effectively, yielding satisfactory results.

We also introduce theories related to the learning-based approach. The learning-based neural network we propose is composed of two existing network structures, specifically the vision transformer and the detection transformer, which have been combined and modified to suit our specific needs. A key constituent of these networks is the attention mechanism, with a particular emphasis on the multi-head attention mechanism. Starting with the fundamental vanilla attention mechanism, we will delineate how these basic elements function and explain their composition within both transformers starting from section 2.9.

2.1 Vehicle State

Before diving into the state representation of the vehicle, it is worth mentioning the assumption we make about the environment's dimensionality. In the context of autonomous driving, a two-dimensional representation is usually sufficient for the map or environment. This assumption is based on the observation that the movement of ground vehicles on highways mostly occurs on a relatively flat plane, with changes in the z-direction (elevation) often being negligible for the tasks of localization, path planning, and navigation.

The state of vehicle i at timestamp k is defined by its position and orientation in the map coordinate system, denoted as:

$$\mathbf{x}_{i,k} = \begin{bmatrix} {}^{\mathrm{W}}x_{i,k} & {}^{\mathrm{W}}y_{i,k} & {}^{\mathrm{W}}\theta_{i,k} \end{bmatrix}^T, \quad \mathbf{x}_{i,k} \in \mathbb{R}^3 \tag{2.1}$$

where $\left({}^{\mathrm{W}}x, {}^{\mathrm{W}}y\right)$ are the coordinates of the vehicle in the global coordinate system (two-dimensional), ${}^{\mathrm{W}}\theta$ denotes the orientation of the vehicle, indicating the current rotation angle. This angle is expressed relative to the x-axis of the global coordinate system (in the positive east direction) and is expressed in radians.

2.2 Sensor Properties & Measurement Model

In autonomous driving systems, vehicles utilize data obtained from various sensors to gather information about the surrounding environment and their ego motion. This information is crucial for the vehicle to engage in autonomous planning and control. As described in [19], the sensors in the vehicle can be categorized into two types: exteroceptive sensors and proprioceptive sensors.

Exteroceptive sensors enable the vehicle to gather information about its surroundings, including the detection of objects, obstacles, road conditions, and other vehicles. On the other hand, proprioceptive sensors are focused on monitoring the internal state of the vehicle itself. These sensors provide feedback regarding the vehicle's motion, orientation, acceleration, and other internal parameters.

In this project, the exteroceptive sensors include cameras and GNSS receiver. And the proprioceptive sensors mainly refer to IMU and wheel encoder.

2.2.1 Camera

Cameras, specifically forward-looking cameras in this project, provide essential information for autonomous driving. They capture images of the environment, which serve multiple purposes. The images contain visual cues such as traffic signs and road markings that are key to understanding the driving context. At the same time, the image sequence can also be used to infer the vehicle's motion, i.e., visual odometry.

The camera data that we handle in this project are not raw RGB images. Instead, it is processed data provided by Zenseact, which has been run through perception and tracking modules to extract high-level features and information.

2.2.1.1 Traffic Signs & Lane markers

The perception module detects and classifies traffic signs and lanemarkers from the camera images. The detected traffic signs and road markings are represented as objects in the image, with attributes including their class (e.g., stop sign, speed limit sign), position in the image, and, when applicable, the associated value (e.g., the speed limit). These detections are crucial for the vehicle to understand and follow traffic rules.

For the detected traffic signs at timestamp k, the observations of traffic sign j from vehicle i can be represented as:

$$\mathbf{z}_{i,k}^{\mathrm{TS}j} = \begin{bmatrix} {}^{\mathrm{C}}x_{i,k} & {}^{\mathrm{C}}y_{i,k} \end{bmatrix}^T, \quad \mathbf{z}_{i,k}^{\mathrm{TS}_j} \in \mathbb{R}^2, \tag{2.2}$$

where $({}^{\mathrm{C}}x_{i,k}, {}^{\mathrm{C}}y_{i,k})$ denotes the pixel coordinates of the detected traffic sign in the camera image. The traffic sign j is identified by the vehicle i at the timestamp k. It should be noted that this format applies not only to traffic signs, but also to other road elements detected through the camera, such as lane markers. Therefore, any detected road elements will be similarly represented as $\mathbf{z}_{i,k}^{\mathrm{Element}j}$ in the subsequent discussion.

2.2.1.2 Visual Odometry

In this project, the camera provides motion information mainly via Visual Odometry (VO). VO functions by analyzing sequences of images taken by the camera, extracting features from them, and subsequently tracking these feature movements across multiple frames. Through this process, the camera offers estimates of the vehicle's relative motion in its frame.

For the visual odometry at timestamp k, the observations of the vehicle's motion j from vehicle i can be represented as:

$$\mathbf{z}_{i,k}^{\mathrm{odom}} = \begin{bmatrix} {}^{\mathrm{C}}\Delta x_{i,k} & {}^{\mathrm{C}}\Delta y_{i,k} & {}^{\mathrm{C}}\Delta \theta_{i,k} \end{bmatrix}^T, \quad \mathbf{z}_{i,k}^{\mathrm{odom}} \in \mathbb{R}^3, \tag{2.3}$$

where $({}^{\mathrm{C}}\Delta x_{i,k}, {}^{\mathrm{C}}\Delta y_{i,k}, {}^{\mathrm{C}}\Delta \theta_{i,k})$ denotes the observed relative motion in the x, y direction and orientation respectively, in the camera frame. These observations are essentially equivalent to the odometry readings, as explained in your previous section, but derived from the camera data.

While Vehicle Odometry, sourced from IMUs and Wheel Encoders, is available for motion estimation, we give precedence to VO in this project for a few reasons. The highways, our primary environment, often present a myriad of distinguishable features and are typically well-lit. In such scenarios, consumer-grade cameras can produce reliable VO outcomes. In contrast, consumer-grade IMUs might experience notable noise interference, leading to error accumulation. Under favorable lighting and when the environment is feature-rich, VO can frequently surpass IMU and Wheel Encoder-based odometry.

2.2.2 GNSS

Global Navigation Satellite Systems (GNSS), such as GPS, GLONASS, Galileo, and BeiDou, provide a global and direct measure of a vehicle's location. They can be utilized as a key component for an autonomous vehicle's navigation system. The raw data obtained from the GNSS receiver includes the current longitude, latitude, and heading of the vehicle, represented as:

$$\mathbf{g}_{i,k} = \begin{bmatrix} \lambda_{i,k} & \phi_{i,k} & \theta_{i,k} \end{bmatrix}^T, \quad \mathbf{g}_{i,k} \in \mathbb{R}^3 \tag{2.4}$$

In this case, $\lambda_{i,k}$ represents the longitude, $\phi_{i,k}$ the latitude, and $\theta_{i,k}$ the heading of vehicle i at timestamp k. The heading angle $\theta_{i,k}$ is expressed in radians.

However, since the geographic coordinates (longitude, latitude) do not directly correspond to the vehicle's state representation used in this framework, a conversion is required. To ensure consistency with the rest of the data, the received GNSS coordinates are transformed into the East-North-Up (ENU) Cartesian coordinate system. This conversion can be expressed as.

$$^{\mathrm{W}}\mathbf{g}_{i,k} = \mathcal{T}\left(\mathbf{g}_{i,k}, \mathbf{p}_{\mathrm{ref}}\right) = \begin{bmatrix} ^{\mathrm{W}}x_{i,k}^{\mathrm{GNSS}} & ^{\mathrm{W}}y_{i,k}^{\mathrm{GNSS}} & ^{\mathrm{W}}\theta_{i,k}^{\mathrm{GNSS}} \end{bmatrix}^T \tag{2.5}$$

Here, $\mathcal{T}$ is a function that transforms from the geodetic frame to the ENU Cartesian frame, and $\mathbf{p}_{\mathrm{ref}}$ denotes the longitude and latitude of the reference point for the global frame.

2.3 Coordinate System

Coordinate systems play a crucial role in integrating sensor data obtained from various sources, such as lidar, cameras, GNSS, and IMUs. Conversions of coordinate system enable seamless integration of data from different sensors, each potentially using its unique coordinate system. By transforming sensor measurements into a unified coordinate system, data fusion becomes feasible, enhancing the overall accuracy and consistency of the map.

For the camera coordinate system used in this project, the origin is located at the focal point of the camera. Following the orientation of the camera, the X-axis points forward, the Y-axis points left, and the Z-axis points up. Within this camera coordinate system, the positions of traffic signs and lane markings are represented. The process of acquiring these positions usually involves several steps: initially, each frame of the camera image is inputted; subsequently, learning-based segmentation is utilized to identify the pixel positions of the corresponding elements; finally, the pixel positions are projected onto the camera coordinate system via depth estimation.

The vehicle coordinate system, often referred to as the body-fixed coordinate system or the car coordinate system, serves as a fixed reference frame relative to the vehicle itself. Its origin is set at a specific point in the vehicle, the center of the rear axle in our case. The choice of origin can depend on the specific application or convention. The X-axis often points forward from the vehicle in the direction of travel. The Y-axis usually points to the left or right of the vehicle, perpendicular to the direction of travel. The Z-axis points upward, perpendicular to the ground. When performing the transformation between camera or sensor coordinate systems and the world coordinate system, the vehicle coordinate system often serves as an intermediary. For example, objects detected by a camera can be initially localized in the vehicle coordinate system based on the camera's position on the vehicle, and subsequently projected onto the world coordinate system based on the vehicle's position in the world.

It is worth noting that in this project, the camera coordinate system and the vehicle coordinate system can be considered as the same coordinate system. This approach greatly simplifies the complexity of data processing and calculations. If the camera coordinate system and the vehicle coordinate system are treated as two different coordinate systems, separate calculations of camera and vehicle positions and orientations would be required, followed by transformations. Although considering the camera coordinate system and the vehicle coordinate system as the same coordinate system may introduce some errors, these errors are typically smaller than others. Other processing steps in our map construction process can mitigate or reduce them.

In autonomous driving technology, the world coordinate system provides a global reference framework essential for positioning vehicles within a broader environment. This global framework is typically grounded on a geographical coordinate system established on the Earth's surface. Geographical coordinate systems account for the actual shape of the Earth, usually perceived as an ellipsoid, and use longitude, latitude, and altitude to identify any location on Earth. The significance of geographical coordinate systems in autonomous driving stems from their ability to provide a global and consistent reference for positioning anywhere on Earth. This is particularly crucial for long-distance navigation, where vehicles may have to traverse considerable distances or commute between different regions. A commonly used geographical coordinate system is WGS 84, employed by the Global Positioning System (GPS) and numerous other mapping applications. Regardless of the vehicle's location, GPS receivers in autonomous vehicles use this system to deliver consistent, accurate location data. In addition to positioning, geographical coordinate systems are also deployed in creating digital maps and geospatial databases for autonomous driving. These maps, rooted in geographical coordinates, can store a variety of information regarding the driving environment, including the locations of roads, intersections, and other significant features. Translating this data into a geographical coordinate system ensures compatibility and interoperability with GPS and other geospatial data sources.

Converting geographical coordinate systems to Cartesian coordinate systems is a critical task in autonomous driving technology. The vehicle's position data provided by GNSS receivers are typically in geographical coordinates (longitude and latitude). However, the data from many sensors on autonomous vehicles, such as cameras, LiDARs, and IMUs, are represented in Cartesian coordinates. Bringing all the data into the same coordinate system allows for more consistent data fusion, leading to more accurate and reliable estimations. It's important to note that while the original perception remains unchanged, the quality of the fused or filtered result is enhanced. Additionally, the control and planning algorithms for autonomous driving are usually designed in Cartesian coordinate systems. This is because the vehicle's motion equations can be represented using linear or nonlinear differential equations under a Cartesian coordinate system, which is convenient for the design and implementation of control algorithms.

The conversion from a geographic coordinate system to a Cartesian coordinate system principally encompasses two categories: global (or regional) conversion and

local conversion. Each category has its representative projection method and serves unique purposes:

The conversion from a geographic coordinate system to a Cartesian coordinate system can be broadly categorized into three main types: global conversion, regional conversion, and local conversion, each with its specific projection method and unique purposes:

- **Global Conversion:** The Earth-Centered, Earth-Fixed (ECEF) system represents a global Cartesian coordinate system where the origin is at the Earth's center. This frame rotates alongside the Earth and stands as the primary Cartesian frame utilized by systems such as GPS. ECEF coordinates prove beneficial for delineating positions across vast distances or when considering measurements from satellites.
- **Regional Conversion:** The Gauss-Krüger projection method is typically harnessed for regional conversions. Although this method furnishes accurate outcomes in proximity to the central meridian (the chosen linearization point), its precision wanes considerably as one veers away from it. The SWEREF 99 system, prevalent in Sweden, adopts the Gauss-Krüger method to devise a series of planar coordinate systems. This technique designates a central meridian as the x-axis origin, with the equator functioning as the y-axis origin. The system is expounded further in the cited source [20].
- **Local Conversion:** Local conversions, epitomized by the East-North-Up (ENU) coordinate system, find frequent application in autonomous driving scenarios. The ENU approach inaugurates a local Cartesian frame at a designated point on the Earth's surface. In this configuration, the axes align towards the east ('E'), north ('N'), and upward ('U'), perpendicular to the surface. This local system adeptly mirrors longitude, latitude, and elevation in geographic coordinates by mapping them onto a tangent plane abutting the Earth's surface.

2.4 Simultaneous Localization and Mapping

According to [21][22], filter-based and optimization-based methods are two main categories of Simultaneous Localization and Mapping (SLAM). Filter-based methods, represented by the Extended Kalman Filter (EKF) and Particle Filter (PF), update the state estimate recursively using a prediction-update cycle. They stand out for their real-time state updates and their relatively simple principles and implementation. However, these methods have notable limitations, including potential error accumulation over time and difficulty handling nonlinear and non-Gaussian problems. Even though certain techniques can be employed to tackle nonlinear problems, they often introduce additional errors. On the other hand, optimization-based methods perform global optimization over the entire state history, representing the problem as a graph with nodes and edges denoting states and constraints, respectively. These methods, albeit computationally intensive, are capable of delivering high accuracy, effectively handling nonlinear problems, and reducing error accumulation by con-

sidering the entire state history. Their major drawbacks lie in high computational complexity and less robust real-time capabilities.

When it comes to constructing high-precision maps, optimization-based SLAM methods are often preferred despite their computational demands. They address the need for high-precision results and aptly handle large-scale nonlinear problems inherent in map construction. Modern hardware and algorithms have mitigated the impact of their high computational complexity. Additionally, the process of building high-precision maps does not have stringent real-time requirements. Instead, the emphasis is on obtaining accurate and stable results, making optimization-based methods a more suitable choice for this application.

According to [23], the SLAM process can be described by a discrete-time motion and observation equations as:

$$\begin{cases} \mathbf{x}_k = f\left(\mathbf{x}_{k-1}, \mathbf{u}_k\right) + \mathbf{w}_k & \mathbf{w}_k \sim N\left(0, \mathbf{R}_k\right) \\ \mathbf{z}_{k,j} = h\left(\mathbf{y}_j, \mathbf{x}_k\right) + \mathbf{v}_{k,j} & \mathbf{v}_k \sim \mathcal{N}\left(0, \mathbf{Q}_{k,j}\right). \end{cases} \tag{2.6}$$

The first term of 2.6 is usually called motion model. The motion model predicts the next state $\mathbf{x}_k$ based on its current state $\mathbf{x}_{k-1}$ and control input $\mathbf{u}_k$. Note that u_k can be either the reading of motion sensor or control input in this case. The second term is called measurement model. The measurement model describes the observation data $\mathbf{z}_{k,j}$ generated when a landmark point $\mathbf{y}_j$ is observed in the current state $\mathbf{x}_k$. In the motion and measurement models, we assume that the two noise terms $\mathbf{w}_k$, $\mathbf{v}_{k,j}$ satisfy a Gaussian distribution with zero mean. $\mathbf{R}_k$, $\mathbf{Q}_{k,j}$ are the covariance matrices of noise term. Through the above introduction, the SLAM problem is actually to infer the pose $\mathbf{x}$ and the map $\mathbf{y}$ through the noisy data $\mathbf{z}$ and $\mathbf{u}$.

Optimization-based methods are also referred to as batch methods, as opposed to the previously mentioned filter-based methods commonly known as incremental methods. In batch methods, the data is integrated into a large optimization problem that includes all state variables and all observed data. Considering all time steps from 1 to N and assuming M landmark points, the poses at all time steps and the coordinates of the landmark points can be described as:

$$\mathbf{x} = \left\{\mathbf{x}_1, ..., \mathbf{x}_N\right\}, \quad \mathbf{y} = \left\{\mathbf{y}_1, ..., \mathbf{y}_M\right\}.$$

From a probabilistic perspective, the entire SLAM problem can be seen as estimating the probability distribution of the state variables $\mathbf{x}$ and $\mathbf{y}$, given the input data $\mathbf{u}$ and the observed data $\mathbf{z}$:

$$P\left(\mathbf{x}, \mathbf{y} | \mathbf{z}, \mathbf{u}\right). \tag{2.7}$$

To estimate the conditional distribution of the state variables, using Bayes' rule, following formula exist:

$$P(\mathbf{x}, \mathbf{y} \mid \mathbf{z}, \mathbf{u}) = \frac{P(\mathbf{z}, \mathbf{u} \mid \mathbf{x}, \mathbf{y}) P(\mathbf{x}, \mathbf{y})}{P(\mathbf{z}, \mathbf{u})} \propto P(\mathbf{z}, \mathbf{u} \mid \mathbf{x}, \mathbf{y}) P(\mathbf{x}, \mathbf{y}). \tag{2.8}$$

The left side $P(\mathbf{x}, \mathbf{y} \mid \mathbf{z}, \mathbf{u})$ is posterior probability, and $P(\mathbf{z}, \mathbf{u} \mid \mathbf{x}, \mathbf{y})$ is called likelihood. $P(\mathbf{x}, \mathbf{y})$ is called prior. For nonlinear system, it is normally difficult to find posterior distribution directly. However, it is feasible to solve a state so that the posterior probability under this state is maximized. This approach is also called Maximum a Posteriori (MAP):

$$(\mathbf{x}, \mathbf{y})^*_{\text{MAP}} = \operatorname{argmax} P(\mathbf{x}, \mathbf{y} \mid \mathbf{z}, \mathbf{u}) = \operatorname{argmax} P(\mathbf{z}, \mathbf{u} \mid \mathbf{x}, \mathbf{y}) P(\mathbf{x}, \mathbf{y}) \tag{2.9}$$

Bayes' theorem indicates that maximizing the posterior probability is equivalent to maximizing the product of the likelihood and the prior. However, in the absence of a prior, the problem transforms into finding the maximum likelihood, which is referred to as Maximum Likelihood Estimation (MLE):

$$(\mathbf{x}, \mathbf{y})^*_{\text{MLE}} = \operatorname{argmax} P(\mathbf{z}, \mathbf{u} \mid \mathbf{x}, \mathbf{y}) \tag{2.10}$$

Given the knowledge of the observed data, MLE can be understood as determining the state or conditions under which the currently observed data is most likely to be generated.

For batch time series data, assuming the inputs and observations at each time step are mutually independent implies that the inputs are independent of each other, the observations are independent of each other, and the inputs are independent of the observations. Therefore, the joint distribution in Equation 2.8 can be decomposed as:

$$P(\mathbf{x}, \mathbf{y} \mid \mathbf{z}, \mathbf{u}) = \prod_k P(\mathbf{u}_k \mid \mathbf{x}_{k-1}, \mathbf{x}_k) \prod_{k,j} P(\mathbf{z}_{k,j} \mid \mathbf{x}_k, \mathbf{y}_j). \tag{2.11}$$

This indicates that the motion and observation at each time step can be treated independently. Considering the observation model, since we assume that the noise terms follow a Gaussian distribution, the conditional probability of the observation data is:

$$P(\mathbf{z}_{j,k} \mid \mathbf{x}_k, \mathbf{y}_j) = N(h(\mathbf{y}_j, \mathbf{x}_k), \mathbf{Q}_{k,j}) \tag{2.12}$$

which is still a Gaussian distribution. Considering the MLE of a single observation, the maximum likelihood of a Gaussian distribution can be obtained by minimizing the negative logarithm. Specifically, for an arbitrary multi-dimensional Gaussian distribution $\mathbf{x} \sim \mathcal{N}(\mu, \mathbf{\Sigma})$its probability density function is:

$$P(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^N |\mathbf{\Sigma}|}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^\top \mathbf{\Sigma}^{-1}(\mathbf{x}-\mu)\right) \tag{2.13}$$

.

Taking the negative logarithm of both sides, one will get:

$$-\ln P(\mathbf{x}) = \frac{1}{2}(\mathbf{x}-\mu)^\top \mathbf{\Sigma}^{-1}(\mathbf{x}-\mu) + \frac{d}{2}\ln(2\pi) + \frac{1}{2}\ln|\mathbf{\Sigma}| \tag{2.14}$$

The logarithmic function exhibits monotonically increasing behavior, thus maximizing the original function is equivalent to minimizing its negative logarithm. The second and third term of the right side of Equation 2.14, being independent of x, can be disregarded. Consequently, by minimizing the first quadratic term, we obtain the maximum likelihood estimation of the state. Substituting the observation model from the SLAM problem into this expression leads to the optimization of:

$$\begin{aligned}(\mathbf{x}_k, \mathbf{y}_j)^* &= \arg\max \mathcal{N}\left(h\left(\mathbf{y}_j, \mathbf{x}_k\right), \mathbf{Q}_{k,j}\right) \\ &= \arg\min \left(\left(\mathbf{z}_{k,j} - h\left(\mathbf{x}_k, \mathbf{y}_j\right)\right)^T \mathbf{Q}_{k,j}^{-1}\left(\mathbf{z}_{k,j} - h\left(\mathbf{x}_k, \mathbf{y}_j\right)\right)\right)\end{aligned} \tag{2.15}$$

Equation 2.15 is to minimize the noise term, which is equivalent to the quadratic form of the error. This quadratic form is referred to as the Mahalanobis distance. In fact, it is the Euclidean distance weighted by $\mathbf{Q_{k,j}^{-1}}$. In optimization problem, $\mathbf{Q_{k,j}^{-1}}$ and $\mathbf{R_k^{-1}}$ are called as information matrix, which is exactly the inverse of covariance matrix. Error terms with larger information matrices (smaller covariance) are assigned greater weights, indicating that the optimization algorithm places more emphasis on these terms and strives to minimize their errors as much as possible. Conversely, terms with smaller information matrices (larger covariance) are assigned smaller weights.

According to the motion and observation models presented earlier in Equation 2.6, the errors between the input and observation data at each instance and the corresponding model can be defined as:

$$\begin{aligned}\mathbf{e}_{u,k} &= \mathbf{x}_k - f\left(\mathbf{x}_{k-1}, \mathbf{u}_k\right) \\ \mathbf{e}_{z,j,k} &= \mathbf{z}_{k,j} - h\left(\mathbf{x}_k, \mathbf{y}_j\right),\end{aligned} \tag{2.16}$$

Then, minimizing the weighted Mahalanobis distance between the estimated value and the measurements from sensors is equivalent to finding the maximum likelihood estimation. The negative logarithm allows to turn the product into a summation:

$$\min J(\mathbf{x}, \mathbf{y}) = \sum_k \mathbf{e}_{u,k}^T \mathbf{R}_k^{-1} \mathbf{e}_{u,k} + \sum_k \sum_j \mathbf{e}_{z,k,j}^T \mathbf{Q}_{k,j}^{-1} \mathbf{e}_{z,k,j} \tag{2.17}$$

The solution obtained by solving such a least squares problem is equivalent to the maximum likelihood estimation of the state. However, due to the presence of noise,

the estimated trajectory and map may not perfectly satisfy the motion and observation equations in the SLAM framework. By iteratively refining the estimated state, the overall error decreases until reaching a local minimum, which characterizes the typical nonlinear optimization process. Solving such a nonlinear optimization problem involves the application of knowledge from the field of nonlinear optimization. Additionally, in order to obtain a more robust result, methods for handling outliers are also employed. These aspects will be discussed in the later sections.

2.5 Factor Graph

The previous section introduces the least squares problem as 2.17, which is composed of a sum of many error terms. However, the objective function only describes the optimization variables and numerous error terms without explicitly indicating their relationships. Graph-based optimization, on the other hand, represents the optimization problem as a **factor graph**, providing a visual representation of the problem.

A graph consists of multiple **variable nodes** and **factor nodes** connecting these variable nodes. Variable nodes represent the state variables of the system, namely, the pose of the vehicle (position and orientation) and the position of specific landmarks or features in the environment. Factor nodes represent the constraints or error terms derived from sensor measurements. For example, a factor may represent the vehicle's motion between two time steps (odometry constraint), or it may represent the observation relationship between the vehicle and a landmark (observation constraint). Factor nodes define a cost or error function that is minimized during the optimization process to estimate the variable configuration that best satisfies all factors. These two types of nodes are connected by edges, which represent the relationships between factors and variables. Therefore, one can construct a corresponding graph representation for any nonlinear least squares problem like 2.17 .

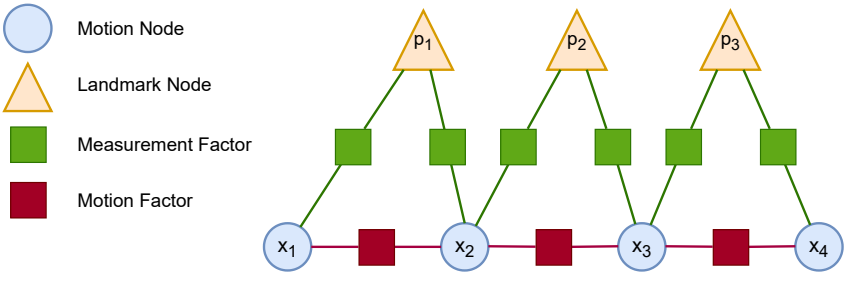

Figure 2.1: Graph-based optimization

The graph is depicted in figure 2.1. The blue circles represent vehicle's pose nodes, the orange triangles represent landmark nodes, the green squares represent measurement factor, and the claret squares are motion factors. Through this graph, the structure of the entire SLAM problem can be visualized. For example, landmark

point p_2 is observed by both x_2 and x_3, and there exists a motion constraint between x_2 and x_3.

To incorporate a factor into a factor graph, typically, variable nodes, measurement values, error functions, and covariance matrices are required. The covariance matrices mentioned here are actually the $\mathbf{R}$ and $\mathbf{Q}$ matrices. The information matrix can be obtained by inverting it. In the rest of this paper, the addition of error terms to optimization problems will be referred to as adding a certain factor to the graph. For a detailed explanation of the application and underlying principles of factor graphs in robotic perception, one can refer to [24].

2.6 Optimization Algorithm

In SLAM, the Levenberg-Marquardt (L-M) algorithm is commonly used for graph optimization to solve the joint optimization problem of robot trajectory and map. The L-M algorithm is employed to solve the least squares problem represented by Equation 2.17. Compared to another commonly used algorithm, Gauss-Newton (G-N), the L-M algorithm demonstrates better stability when dealing with nonlinear problems. L-M achieves this stability by introducing a damping factor to balance the trade-off between gradient descent and G-N methods, ensuring more reliable performance in nonlinear problem settings. Furthermore, the L-M algorithm automatically adjusts the damping factor based on the variation of residuals, allowing it to perform well across different problem scenarios. The following part will provide a brief introduction to the principles of the L-M algorithm. [23] provides a good introduction to the application of the L-M algorithm in SLAM, and [25] gives a detailed mathematical explanation.

Firstly, a least squares problem can be written as follows:

$$\min_{x} F(\mathbf{x}) = \|f(\mathbf{x})\|_2^2 \tag{2.18}$$

where $f(\mathbf{x})$ is the error term, and $F(\mathbf{x})$ is the overall cost function. The goal is to find a set of parameters $\mathbf{x}$ such that the sum of squares of the error function is minimized. If f has a simple form, the problem can be solved directly to obtain an analytical solution. By setting the derivative of the cost function to zero and solving for the optimal values of x, multiple extrema can be obtained. The optimum can be identified by comparing their respective function values.

However, in a SLAM problem, $f(\mathbf{x})$ often takes on complex forms, making it nearly impossible to obtain an analytical solution. For least squares problems that are inconvenient to solve directly, an iterative approach provides a feasible solution. The specific steps are as follows:

1. Set an initial value $\mathbf{x}_0$.
2. For k-th iteration, we find an incremental value of $\Delta\mathbf{x}_k$, such that the objective function $\|f(\mathbf{x}_k + \Delta\mathbf{x}_k)\|_2^2$ reaches a smaller value.
3. If $\Delta\mathbf{x}_k$ is small enough, stop the algorithm.

4. Otherwise, let $\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta\mathbf{x}_k$ and return to step 2.

In simple terms, for the Levenberg-Marquardt (L-M) algorithm, the formula for solving the increment is as follows:

$$\Delta x_k = -\left(\mathbf{J}^T\mathbf{J} + \lambda\mathbf{D}\right)^{-1}\mathbf{J}^T f(\mathbf{x}_k) \tag{2.19}$$

Here, $\mathbf{J}$ represents the Jacobian matrix of the error function with respect to the parameters, $f(\mathbf{x}_k)$ denotes the error term at the current parameters, $\mathbf{D}$ represents a diagonal matrix with diagonal elements equal to the diagonal elements of $\mathbf{J}^\top\mathbf{J}$, and λ is a damping factor. The damping factor λ plays a crucial role in the algorithm. When λ is small, the algorithm behaves like the G-N method. Conversely, when λ is large, the algorithm resembles the gradient descent method. The value of λ is automatically adjusted based on the variation of residuals, enabling the algorithm to adaptively handle different problem scenarios.

In summary, the L-M method, following the aforementioned steps, iteratively solves for a set of robot poses (position and orientation) and feature point positions that best explain all the observed data. These results are obtained through joint optimization, indicating that they represent the optimal explanations for all the observed data. Currently, the L-M algorithm is widely implemented in various third-party libraries, commonly used in the field of SLAM are *Ceres Solver* [26] and *g2o* [27], both of which are C++ libraries. These libraries provide mature and efficient implementations of the L-M algorithm, eliminating the need for implementing the entire optimization algorithm from scratch. In this project, *Ceres Solver* is utilized.

2.7 Outlier Processing

In the least squares problem defined in Equation 2.17, minimizing the sum of squared 2-norms of the error terms is used as the objective function. However, a significant issue is hidden in this approach: if a certain error term contains incorrect data due to sensor failure or mismatches, an edge that should not have been included in the graph may be added. The optimization algorithm itself cannot identify this as erroneous data, and it treats all edges as normal error terms. In graph optimization, the presence of a large error edge with a large gradient implies that adjusting the variables associated with it would result in a significant decrease in the objective function. As a result, the algorithm prioritizes adjusting this edge, which is actually incorrect, leading to incorrect adjustments in other correct edges.

This problem arises from the rapid growth of the squared terms. To address this issue, robust kernel functions are introduced to ensure that the error of each edge does not become excessively large, thereby masking the effects of other edges. The specific approach involves replacing the squared terms in the original error terms with a function that grows less rapidly, while ensuring its smoothness to allow for differentiation.

There are various types of robust kernel functions, commonly used ones include the Huber kernel function and the Cauchy kernel function [28]. The Huber kernel function behaves as a squared error for small errors and as a linear error for large errors. It exhibits certain robustness against larger outliers. The form of the Huber kernel function is as follows:

$$\rho(e) = \begin{cases} \frac{1}{2}e^2 & \text{if } |e| \leqslant \delta \\ \delta\left(|e| - \frac{1}{2}\delta\right) & \text{otherwise} \end{cases} \tag{2.20}$$

When the error term e is larger than the threshold δ, the growth of the function transitions from quadratic to linear form, effectively imposing a limit on the maximum gradient.

The Cauchy kernel function remains smooth across the entire range of residuals, but it exhibits a slower growth rate for large residuals compared to the Huber kernel function. This property grants the Cauchy kernel function stronger robustness against larger outliers. It has the following form:

$$\rho(x) = \frac{1}{2}\ln\left(1 + x^2\right) \tag{2.21}$$

Figure 2.2 illustrates the differences between the Huber and Cauchy robust kernel functions and the original quadratic error. The Cauchy kernel function exhibits a thicker tail compared to the Huber kernel function, resulting in a smaller contribution of large residuals to the total error. However, both functions have a much slower growth rate compared to the quadratic function. More information about robust kernel functions can be found in [29].

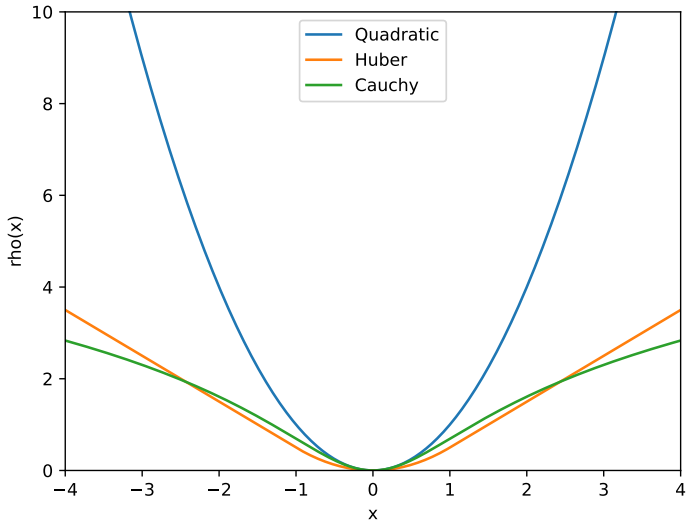

Figure 2.2: Comparison of different robust kernel functions

2.8 Theories for Learning Based Approach

SLAM methods have solidified their role as the cornerstone of localization and mapping in robotics, excelling in geometric and spatial reasoning. These techniques are integral to a myriad of robotic applications, delivering efficient and accurate solutions. While SLAM provides a robust foundation, the integration of deep learning, particularly attention mechanisms, opens up new avenues for enhancing these capabilities. Deep learning algorithms are adept at handling high-dimensional data and capturing intricate patterns, offering a comprehensive way to integrate semantic understanding.

To this end, we also introduce a deep learning-based approach specifically designed for aggregating crowdsourced data. Our deep learning-based approach is tailored to seamlessly assimilate information from disparate sources while mitigating issues related to noise, bias, and redundancy. This strategy not only enhances the robustness of our system but also extends its applicability to a broader range of tasks and environments. From 2.9 onwards, we delve into the core mechanisms that constitute our deep learning neural network.

2.9 Attention Mechanism

The attention mechanism [30], [31], particularly self-attention, has emerged as a pivotal concept in deep learning, facilitating the model's focus on relevant parts of the input. In self-attention, queries Q, keys K, and values V are derived from the same source, typically the preceding layer's output or projected input if it is the first layer in the model. They are formulated as $\mathrm{Q} = \mathrm{X} \cdot \mathrm{W_Q}$, $\mathrm{K} = \mathrm{X} \cdot \mathrm{W_K}$, and $\mathrm{V} = \mathrm{X} \cdot \mathrm{W_V}$, where X is the input and $\mathrm{W_Q}, \mathrm{W_K}, \mathrm{W_V}$ are learned weight matrices. The attention scores are then computed as $\text{Score} = \frac{\mathrm{Q} \cdot \mathrm{K}^T}{\sqrt{d_k}}$, where d_k is the dimensionality of the keys. This is followed by the application of the softmax function to obtain the attention weights, Weights = softmax(Score), and finally, the weighted sum of the values is computed as Head = Weights $\cdot$ V and projected through a linear layer Output = Head $\cdot W_{output}$. This mechanism allows for parallelized processing and adept handling of long-range dependencies, thereby significantly enhancing the model's context-aware representation and performance.

2.10 Multi-Head Attention

Multi-Head Attention (MHA) [31] is a key component of the transformer architecture that enables the model to focus on different parts of the input simultaneously. Compared to the vanilla attention mechanism, MHA is designed to capture various relationships in the data by applying multiple attention mechanisms in parallel. It was introduced to overcome the limitations of single attention mechanisms, which may not capture all the relevant relationships in the data. By using multiple attention heads, the model can recognize multiple types of relationships simultaneously, leading to a richer understanding of the data. Multi-Head Attention gives trans-

former model the ability to capture diverse relationships within the data.

The Multi-Head Attention mechanism derives queries Q, keys K, and values V into h heads using specific weight matrices, W_i^Q, W_i^K, and W_i^V, to capture different relationships in parallel. $\mathrm{Q}_i = \mathrm{X} \cdot W_i^Q$, $\mathrm{K}_i = \mathrm{X} \cdot W_i^K$, and $\mathrm{V}_i = \mathrm{X} \cdot W_i^V$, where X is the input, and queries $Q = \{Q_i \mid i = 0, 1, ...h\}$, keys $K = \{K_i \mid i = 0, 1, ...h\}$, and values $V = \{V_i \mid i = 0, 1, ...h\}$. For each head, attention scores are computed as $\mathrm{Score}_i = \frac{\mathrm{Q}_i \cdot \mathrm{K}_i^T}{\sqrt{d_k}}$, where d_k is the dimensionality of the keys, followed by applying the softmax function to obtain weights $\mathrm{Weights}_i = \mathrm{softmax}(\mathrm{Score}_i)$. The weighted sum of values $\mathrm{Head}_i = \mathrm{Weights}_i \cdot <\mathrm{v}_i$ is computed, and the results across all heads are concatenated $\mathrm{Concatenated} = \mathrm{concat}(\mathrm{Head}_1, \ldots, \mathrm{Head}_h)$, then projected through a final linear layer $\mathrm{Output} = \mathrm{Concatenated} \cdot W_{output}$.

2.11 Vision Transformer

The Vision Transformer(ViT) [32] is a model that applies transformer architecture, originally designed for natural language processing (NLP), to computer vision tasks. Unlike traditional convolutional neural networks (CNNs) that process images using convolutional layers, ViT leverages self-attention mechanisms to capture global dependencies within an image.

The introduction of ViT was motivated by the desire to explore whether the success of transformers in NLP could be translated to the domain of computer vision. Traditional CNNs are inherently local and hierarchical, processing images in a piecewise manner. This can limit their ability to capture long-range dependencies within an image. ViT was introduced to overcome these limitations and provide a more flexible and powerful way to process visual data.

ViT aims to solve the problem of capturing global relationships within an image, which can be critical for many vision tasks. By using multi-head attention mechanisms, ViT can consider the entire image at once and weigh the importance of different parts of the image relative to each other. This allows the model to capture complex patterns and relationships that might be missed by traditional CNNs.

The structure of the Vision Transformer begins with dividing the input image into a grid of non-overlapping patches (e.g., 16x16 pixels), which are then linearly embedded into a sequence of vectors. Positional information is added to these patch embeddings to retain spatial information. The sequence of patch embeddings is then passed through a series of transformer encoder layers, each consisting of a Multi-Head Self-Attention mechanism that allows the model to weigh the importance of different patches relative to each other, and a Feed-Forward Neural Network made up of fully connected layers. Encoded features after the transformer can be decoded to different forms of output to finalize various tasks.

2.12 Detection Transformer

Detection Transformer(DETR) [33] is an end-to-end object detection model that leverages the Transformer architecture. Unlike traditional object detection models that rely on region proposals and complex pipelines, DETR simplifies the process by treating object detection as a direct set prediction problem. Traditional object detection models often involve complex components like anchor boxes, non-maximum suppression, and region proposals. These components can be difficult to tune and integrate. DETR was introduced to simplify the object detection pipeline by eliminating these components and providing an end-to-end trainable model.

DETR is composed of a CNN image backbone and a Transformer encoder-decoder architecture. Usually, a ResNet is used as the CNN backbone, which is used for extracting features from images. The encoder of the transformer captures global relationships among the image features. The decoder of the transformer attends to the encoder's output and a fixed set of learned object queries, predicting both object classes and bounding boxes. DETR also takes a fixed number of object queries that are learned during training. These queries are used to predict the objects in the image, with each query corresponding to a potential detection.

2.13 Hungarian Matching Algorithm

The Hungarian Matching Algorithm, also known as the Kuhn-Munkres algorithm [34], is a combinatorial optimization technique designed to solve the assignment problem in polynomial time. The algorithm aims to find the optimal assignment of a set of "agents" to a set of "tasks" such that the total cost or weight is minimized. The problem can be represented as a weighted bipartite graph, where one set of vertices represents agents, another set represents tasks, and the edges between them are weighted by the cost of assigning a particular agent to a particular task.

The problem is formulated as, given a weighted bipartite graph $G = (A \cup T, E)$, where A is the set of agents, T is the set of tasks, and E is the set of edges with weights $w(a, t)$ representing the cost of assigning agent a to task t, the objective is to find a perfect matching M that minimizes the total cost:

$$\text{Minimize} \sum_{(a,t)\in M} w(a,t)$$

The algorithm is carried on as follows:

1. Initialization: Create a cost matrix C where $C[i][j]$ represents the cost of assigning agent i to task j.

2. Row and Column Reduction: Subtract the smallest element in each row from all elements in that row. Repeat the process for each column.

3. Covering Zeros: Use a minimum number of lines (horizontal and vertical) to cover all zeros in the reduced matrix.

4. Adjusting Matrix: Find the smallest non-covered element x and subtract x from all non-covered elements, and add x to the intersections of the lines.

5. Optimal Assignment: Repeat steps 3-4 until an optimal assignment can be made.

The Hungarian Matching Algorithm provides an efficient and optimal solution to the assignment problem. Its polynomial time complexity makes it suitable for a wide range of applications in both academia and industry.

For a more in-depth understanding, readers are encouraged to consult the original paper by Harold W. Kuhn [34] as well as subsequent literature [35]–[39] that extends and applies the algorithm.

3

Modeling of Optimization-based Method

In the previous chapter, we have introduced the basic theory and applications of SLAM. However, applying these theories to specific tasks, such as building and updating high-precision maps with data from multiple vehicles, requires a deeper consideration and modeling. This chapter will describe the modeling process we have adopted in achieving this goal.

Figure 3.1 shows the workflow that we proposed to construct and update the map. Our approach is divided into three main stages: Individual Drive, Multiple Drives, and Map Generation & Update. Each stage comprises distinct steps that collectively enable the construction and updating of high-precision maps using data from multiple vehicles. Firstly, we will delve into the modeling during the Individual Drive stage, detailing how we handle and clean raw data, use SLAM for trajectory smoothing to optimize vehicle movement, and align each locally generated map to SD Map which is available from OpenStreetMap [40]. Then, we will shift to the modeling during the Multiple Drives stage, exploring how we align maps from multiple vehicles and further optimize them via traffic sign alignment and lane marker alignment. Lastly, we will discuss the generation and updating of maps, including how we leverage the optimized vehicle poses and lane markers to generate and update maps, and how these individual maps are stitched together to form a complete map.

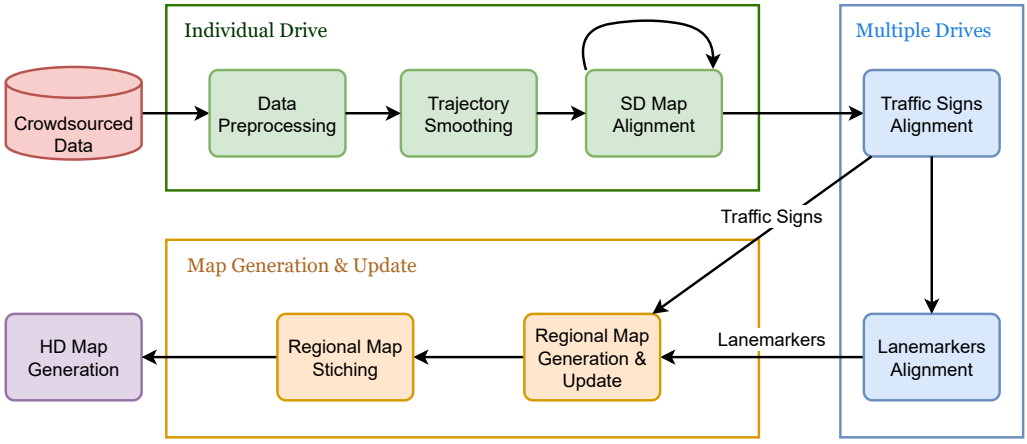

Figure 3.1: General workflow for crowdsourcing HD map generation & update

3.1 Individual Drive

The Individual Drive stage lays the groundwork for our mapping framework by transforming raw sensor data into a globally consistent vehicle localization solution. The purpose of this stage is two-fold: first, to handle the inherent uncertainties and errors in the real-world driving data, and second, to leverage the information from the SD map to reinforce the robustness of our localization. By the end of this stage, the expected output is an individual vehicle map that aligns well with the globally consistent SD Map. This result then serves as a valuable input for subsequent processing stages in our framework.

3.1.1 Data Preprocessing

Data forms the backbone of the SLAM-based approach for map construction and localization. However, raw sensor data often brings along challenges such as inherent inaccuracies, disparities in data formats, and sensor noise. These issues can jeopardize the accuracy of map construction. To counteract these challenges, a rigorous data preprocessing step is employed. This step is crucial as it aims to synchronize and harmonize data streams, reduce noise, and manage outliers. As a result, the quality and consistency of the data are enhanced, setting a firm foundation for precise mapping and localization.

The unique demands of this project necessitate a thorough and detailed approach to data handling. The specific techniques adopted in data preprocessing will be discussed further in the Implementation chapter. Emphasis is given to selecting only high-quality, high-confidence data for this project due to two main reasons:

Firstly, the incorporation of consumer-grade sensors means that data might exhibit more outliers. While these sensors provide an economical avenue for large-scale data collection, their precision and reliability might not match that of industrial-grade devices. Although these sensors produce abundant data, the output can be prone to inaccuracies, noise, and disparities, which can introduce potential discrepancies and errors during the mapping process.

Secondly, this project's reliance on crowdsourced data means accumulating data from multiple vehicles for the same road segment. This approach substantially increases the data volume, often exceeding the actual need. While such redundancy is a safeguard against data loss or corruption, it also presents the challenge of extracting high-quality data from the vast pool.

3.1.2 Trajectory Smoothing

Constructing a precise and reliable trajectory for each vehicle's journey forms a crucial underpinning for map construction. The raw data, including GNSS data, odometry readings, and traffic sign tracking information, offer the fundamental information for this process. However, this data is often tainted with noise and errors, and thus trajectory smoothing is required to refine the vehicle trajectories.

GNSS data, offering global positioning capabilities, forms a cornerstone of our tra-

jectory information. Yet, it presents a unique dichotomy in its error characteristics. Over short timescales or distances, GNSS data tends to have high relative error due to factors such as signal multipath, atmospheric interference, and clock inaccuracies. However, over larger distances and timescales, the absolute error is bounded, and the error for positions far apart in time is typically within a few meters.

On the other hand, odometry provides information about relative displacement, and its error characteristics contrast with those of GNSS. Over short timescales, odometry can be highly accurate, but due to the accumulation of small errors over time, the absolute error can grow unbounded over long distances.

Traffic sign tracking information serves as an additional reference for our trajectory data. By continuously tracking the same traffic sign over consecutive frames, we gain another perspective on relative vehicle motion. This information supplements and adds robustness to our trajectory estimation, especially in environments with GNSS signal blockage or drift in odometry.

To harmonize these different sources of information and minimize their respective errors, we employ a trajectory smoothing method based on SLAM. We formulate the trajectory smoothing problem as a factor graph, with the vehicle's state at each moment constrained by GNSS factors, and the state between adjacent moments constrained by odometry factors. The tracking information from traffic signs in each frame forms additional constraints between the vehicle's pose and the traffic signs. Figure 3.2 presents the structure of the overall problem.

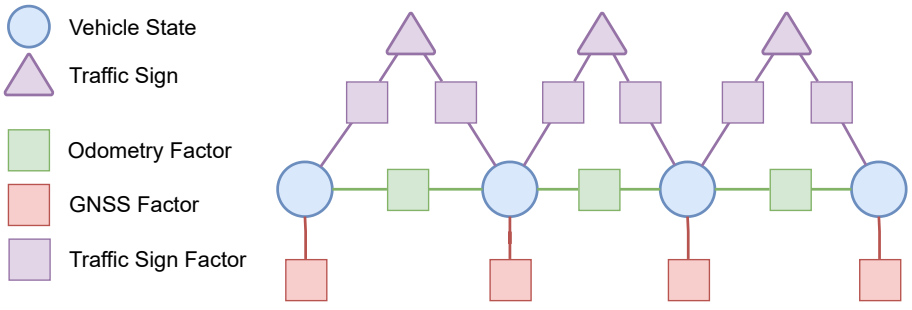

Figure 3.2: Graph of trajectory smoothing

3.1.2.1 Odometry Factor

The odometry factors in Figure 3.2 are represented by green squares, connecting the states of the vehicle at adjacent time steps. The error term $\mathbf{r}_{i,k}^{\text{odom}} \in \mathbb{R}^3$ and covariance matrix $\mathbf{R}_{i,k}^{\text{odom}} \in \mathbb{R}^{3\times3}$ of odometry factor is then defined as:

$$
\begin{aligned}
\mathbf{r}_{i,k}^{\text{odom}} &= \phi_{\text{odom}}\left(\mathbf{x}_{i,k+1}, \mathbf{x}_{i,k}, \mathbf{z}_{i,k}^{\text{odom}}\right) \\
&= \mathbf{x}_{i,k+1} - \mathbf{x}_{i,k} - \mathbf{z}_{i,k}^{\text{odom}} \\
&= \begin{bmatrix} {}^{\text{W}}x_{i,k+1} \\ {}^{\text{W}}y_{i,k+1} \\ {}^{\text{W}}\theta_{i,k+1} \end{bmatrix} - \begin{bmatrix} {}^{\text{W}}x_{i,k} \\ {}^{\text{W}}y_{i,k} \\ {}^{\text{W}}\theta_{i,k} \end{bmatrix} - \begin{bmatrix} \Delta x_{i,k} \\ \Delta y_{i,k} \\ \Delta\theta_{i,k} \end{bmatrix} \\
\mathbf{R}_{i,k}^{\text{odom}} &= \begin{bmatrix} \sigma_{\Delta x}^2 & 0 & 0 \\ 0 & \sigma_{\Delta y}^2 & 0 \\ 0 & 0 & \sigma_{\Delta\theta}^2 \end{bmatrix}
\end{aligned}
\tag{3.1}
$$

The covariance matrix of odometry factor is set for all time steps k. The covariance matrix includes the variances of Δx, Δy, and $\Delta\theta$ from the odometry, and it is set as a diagonal matrix, assuming that the three elements are independent of each other.

Equation 2.19 mentioned that Jacobian matrix is necessary to solve iteratively. The Jacobian matrices of the odometry factor with respect to $\mathbf{x}_k$ and $\mathbf{x}_{k+1}$, the variables that to be optimized, are as follows:

$$
\begin{aligned}
\frac{\partial \mathbf{r}_{i,k}^{\text{odom}}}{\partial \mathbf{x}_{i,k}} &= \begin{bmatrix} \frac{\partial r_x}{\partial x_k} & \frac{\partial r_x}{\partial y_k} & \frac{\partial r_x}{\partial \theta_k} \\ \frac{\partial r_y}{\partial x_k} & \frac{\partial r_y}{\partial y_k} & \frac{\partial r_y}{\partial \theta_k} \\ \frac{\partial r_\theta}{\partial x_k} & \frac{\partial r_\theta}{\partial y_k} & \frac{\partial r_\theta}{\partial \theta_k} \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \\
\frac{\partial \mathbf{r}_{i,k}^{\text{odom}}}{\partial \mathbf{x}_{i,k+1}} &= \begin{bmatrix} \frac{\partial r_x}{\partial x_{k+1}} & \frac{\partial r_x}{\partial y_{k+1}} & \frac{\partial r_x}{\partial \theta_{k+1}} \\ \frac{\partial r_y}{\partial x_{k+1}} & \frac{\partial r_y}{\partial y_{k+1}} & \frac{\partial r_y}{\partial \theta_{k+1}} \\ \frac{\partial r_\theta}{\partial x_{k+1}} & \frac{\partial r_\theta}{\partial y_{k+1}} & \frac{\partial r_\theta}{\partial \theta_{k+1}} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}
\end{aligned}
\tag{3.2}
$$

3.1.2.2 GNSS Factor

In Figure 3.2, the GNSS factors are represented by red squares, each of which is only connected to the vehicle state at a single moment, indicating that they essentially impose constraints relative to the vehicle's own state. The GNSS factor is defined mathematically as:

$$
\begin{aligned}
\mathbf{r}_{i,k}^{\text{GNSS}} &= \phi_{\text{GNSS}}\left(\mathbf{x}_{i,k}, {}^{\text{W}}\,\mathbf{g}_{i,k}\right) \\
&= \mathbf{x}_{i,k} - {}^{\text{W}}\,\mathbf{g}_{i,k} \\
&= \begin{bmatrix} {}^{W}x_{i,k} \\ {}^{W}y_{i,k} \\ {}^{W}\theta_{i,k} \end{bmatrix} - \begin{bmatrix} {}^{W}x_{i,k}^{\text{GNSS}} \\ {}^{W}y_{i,k}^{\text{GNSS}} \\ {}^{W}\theta_{i,k}^{\text{GNSS}} \end{bmatrix} \\
\mathbf{R}_{k}^{\text{GNSS}} &= \begin{bmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \sigma_y^2 & 0 \\ 0 & 0 & \sigma_\theta^2 \end{bmatrix}
\end{aligned}
\tag{3.3}
$$

3.1.2.3 Traffic Sign Factor

As depicted in Figure 3.2, traffic sign factors are represented by purple squares. Each factor forms a link between a vehicle's state and a detected traffic sign, implying that they establish constraints on the relative positions between the vehicle's trajectory and the location of the traffic sign.

The measurement model establishes the connection between traffic sign observations and their positions in the world coordinate system. This relationship can be expressed as follows:

$$\mathbf{z}_{i,k}^{\mathrm{TS}_j} = h\left(\mathbf{x}_{i,k}, \mathbf{y}_i^{\mathrm{TS}_j}\right) + \mathbf{v} \tag{3.4}$$

$$\begin{bmatrix} {}^{c}x_{i,k} \\ {}^{c}y_{i,k} \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} {}^{\mathrm{w}}x_i^{\mathrm{TS}_j} - {}^{\mathrm{w}}x_{i,k} \\ {}^{\mathrm{w}}y_i^{\mathrm{TS}_j} - {}^{\mathrm{w}}y_{i,k} \end{bmatrix} \tag{3.5}$$

where $\mathbf{v} \sim \mathcal{N}(0, \mathbf{R})$ is the noise term. Based on the measurement model, the traffic sign factor can be defined mathematically as:

$$\mathbf{r}_{i,k}^{\mathrm{TS}_j} = \phi_{\mathrm{TS}}\left(\mathbf{x}_{i,k}, \mathbf{y}_i^{\mathrm{TS}_j}, \mathbf{z}_{i,k}^{\mathrm{TS}_j}\right) = \mathbf{z}_{i,k}^{\mathrm{TS}_j} - h\left(\mathbf{x}_{i,k}, \mathbf{y}_i^{\mathrm{TS}_j}\right) \tag{3.6}$$

$$\mathbf{Q}_{i,k}^{\mathrm{TS}_j} = \sigma_{\mathrm{TS}}^2 \mathbf{I} \tag{3.7}$$

σ^2 is the variance of the observation error and $\mathbf{I}$ is the identity matrix. We assume that the errors in different dimensions are uncorrelated and have same variance.

It's worth mentioning that in Figure 3.2, each traffic sign is linked to the states of the vehicle at several time steps. This relationship arises due to the vehicle repeatedly observing the same traffic sign across sequential frames, thereby enabling traffic sign tracking information. These recurrent observations form a constraint that associates the different temporal states of the vehicle with the position of the traffic sign. Every observation contributes data about the relative position and orientation between the vehicle and the traffic sign. This data generates a complex optimization problem, where the goal is to identify the optimal solutions for both the vehicle states and traffic sign positions under these constraints.

This problem can be addressed effectively by adopting an approach analogous to the bundle adjustment method commonly used in computer vision and photogrammetry. This method is designed to optimize the 3D coordinates of scene features (in this case, traffic signs) and camera parameters (in this case, vehicle states), by minimizing the reprojection error between observed and predicted values. The integration of traffic sign tracking information into the factor graph essentially mirrors a bundle adjustment problem, albeit with the supplementary inclusion of GNSS and odometry information. Through this optimized process, we are capable of determining the best fit for vehicle states and traffic sign positions while taking into account all constraints.

To better illustrate the effects of trajectory smoothing, we have presented the trajectory of a single vehicle based on GNSS, odometry, and post-smoothing in two distinct scales for clarity.

Figure 3.3 provides a macroscopic view of the trajectory, showcasing the overall path taken by the vehicle. From this viewpoint, the smoothed trajectory aligns closely with the GNSS track, while it's evident that the odometry data drifts notably over larger distances. A blue cross on the figure denotes the starting point, marking the convergence of the odometry and GNSS data.

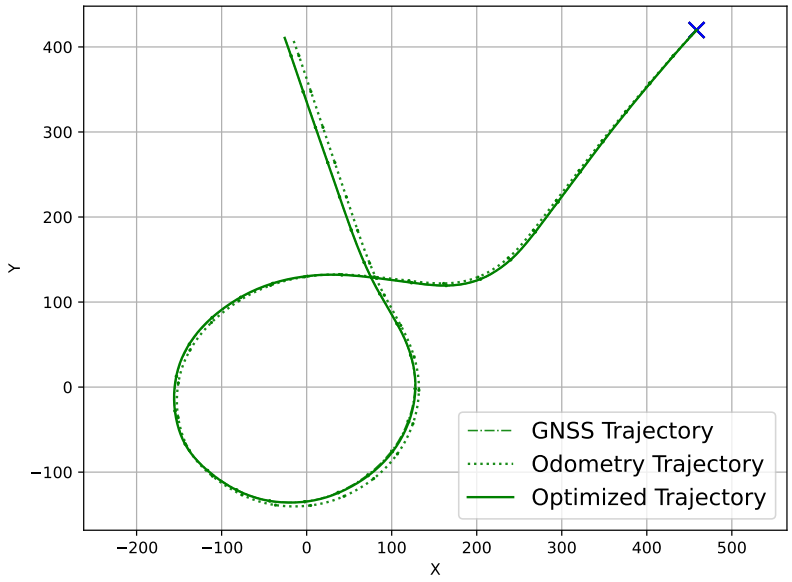

Figure 3.3: Trajectory Smoothing (large scale): The smoothed trajectory aligns with the GNSS track, highlighting the drift in the odometry data. The blue cross indicate the starting position.

In contrast, Figure 3.4 offers a more granular perspective. This close-up view highlights the inherent fluctuations in the GNSS trajectory, with its typical sudden shifts or jitters. The smoothed trajectory is evidently more consistent, incorporating the GNSS positional data's absolute nature with the smoother characteristics inherent to odometry.

Through these figures, it becomes evident that while the GNSS data provides valuable absolute position information, it is susceptible to sudden jitters. In contrast, the odometry-derived trajectory maintains a smooth profile due to its nature of providing consistent relative motion data. However, it is prone to drift over longer distances. The trajectory smoothing process leverages these two data sources, combining their respective strengths. The outcome is an optimized trajectory that is not only smoother but also more representative of the vehicle's actual path.

3.1.3 SD Map Alignment

The HD maps used in this study are generated or updated by crowdsourcing data from multiple vehicles. A key challenge in this process is ensuring that all vehicles perceive the world consistently and generate data that aligns accurately with reality.

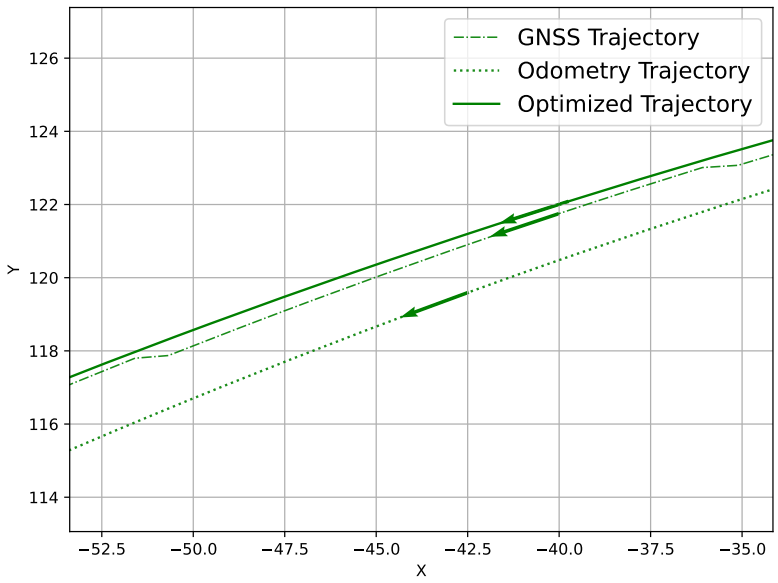

Figure 3.4: Vehicle Trajectory Comparison: The GNSS shows fluctuations, while the smoothed trajectory integrates it with the odometry's consistency.

For instance, if two vehicles traverse the same section of the road, their trajectories should align closely. Moreover, when we transform the detected lane lines to the world coordinate system using each vehicle's position, the resulting lane line locations from the two vehicles should ideally overlap.

Figure 3.5 exemplifies a real-world scenario where significant positional deviations can occur between vehicles, even after individual trajectory smoothing. The solid red and green lines in the figure denote the optimized trajectories of two vehicles, identified as vehicles 3 and 17, while the dashdot lines indicate the respective GNSS trajectories. The small dots represent detected lane points. Despite both vehicles traveling in the same direction on the same road segment, the lane lines they perceive do not align. These discrepancies primarily stem from sensor inaccuracies, especially the GNSS imprecision, leading to varied perceptions of the lane line positions.

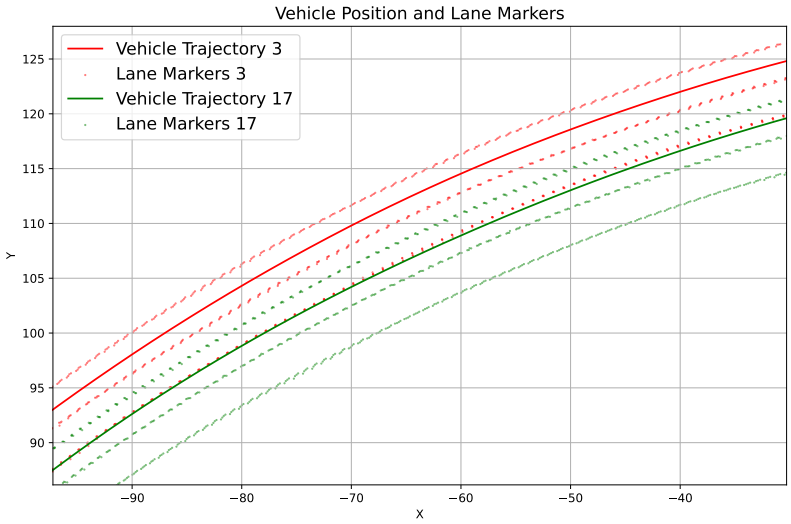

Figure 3.5: The positional deviation between two vehicles that have passed through the same road segment.

The alignment of localized maps produced by individual vehicles with an SD Map (such as OpenStreetMap [40]) is motivated by several factors.

First, aligning all local maps to a shared reference, such as the SD Map, can potentially reduce the positional discrepancy between the trajectories of different vehicles. This strategy may help address challenges arising from sensor inaccuracies and other errors.

Second, aligning with the SD Map might enhance the consistency of maps derived from different vehicles. Given the variability in how vehicles perceive the road environment due to observational conditions and sensor noise, anchoring them to a common reference can promote a more coherent representation.

The SD Map serves as a foundational structure that not only represents the geometric configuration of real-world roads, but also lays the groundwork for the generation and refinement of high-definition maps. It's worth noting that, while the SD map provides a rough outline of what could become a more comprehensive HD map, its true utility and precision can only be fully assessed when integrated with additional information from multiple sources.

Our proposed approach seeks to address these inconsistencies and is grounded on several key assumptions. While these assumptions are not strictly necessary for aligning individual drive trajectories relative to each other, they strengthen our method by also aligning the maps to the SD Map reference. First, we assume that vehicles can detect the road's left and right edges. By averaging the detected points on these edges, we can compute the road centerline detection points. Second, we assume that the calculated road centerline should ideally align with the SD Map links, regardless of the lane the vehicle is traveling in. Lastly, we presume that the two vehicles travel in the same direction along the same road segment. If

these assumptions hold true, our map will not only be well-aligned internally among individual drives but also externally with the SD Map reference. This alignment enhances the accuracy and reliability of our mapping process.

Figure 3.6 depicts the factor graph structure of the optimization problem at this stage. Compared to the previous stage of trajectory smoothing, the factor graph now includes additional elements: the SD Map links, represented by the yellow dashed triangle, and the Centerline Factor, represented by the orange square. The SD Map links are regarded as the ground truth and will not be updated, hence their representation with a dashed line.

In this context, the absolute position information constrained by GNSS is less important, which leads us to reduce the overall weight of the GNSS factors. However, because the GNSS readings still provide valuable shape information for the vehicle's trajectory, their weight is only decreased, not eliminated.

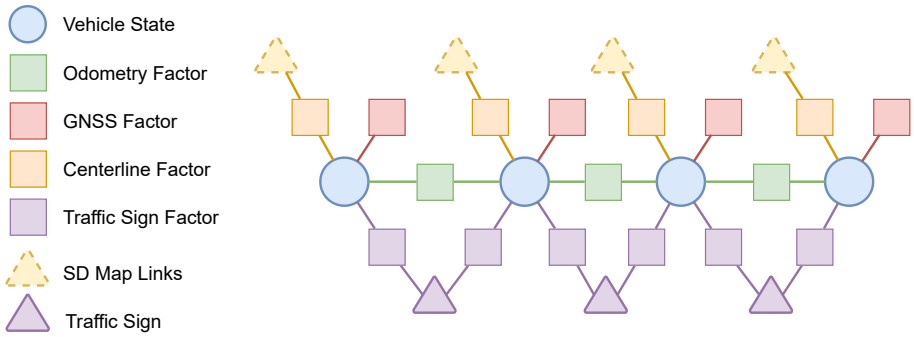

Figure 3.6: Structure of factor graph for alignment to SD map

3.1.3.1 Centerline Factor

The construction of the centerline factor involves establishing a connection between the centerline points and the SD Map links. This process is depicted in Figure 3.7, which illustrates how the road centerline is derived from the detected road edges and compared with the SD Map links.

The road centerline is estimated by detecting road edge positions and subsequently extracting an equal number of sample points, denoted as n, from both the left and right edges. The coordinates of these sample points are averaged to derive the road centerline. Following this, we proceed to map a relationship between the detected centerline points and the SD Map links. For each centerline point, we identify the closest corresponding point on the SD Map links. It is crucial to ensure an adequate density of points on the SD Map links for accurate mapping.

In Figure 3.7, the green points denote road edges, the red points signify the derived road centerline, and the black bidirectional arrows illustrates the established correspondences between the centerline and the SD Map links.

Up to this point, we have identified specific corresponding points on the SD Map links for each detected road centerline point. These points are determined based on

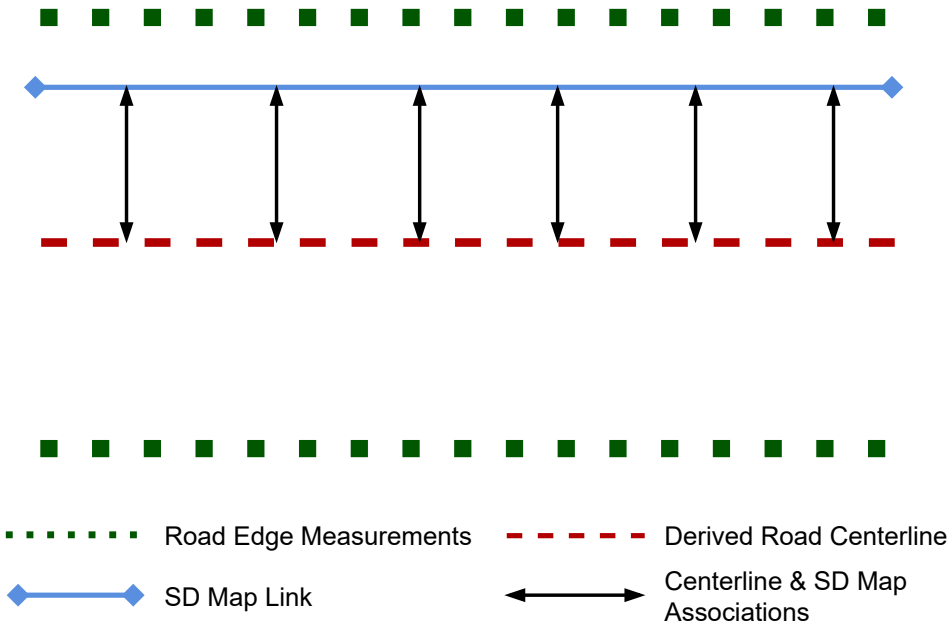

Figure 3.7: The comparison between road edges, road centerline, and SD Map links.

proximity, essentially choosing the closest point on the SD Map link for each centerline point. The objective of introducing the Centerline Factor is to optimize the vehicle's position in a way that minimizes the distance between these corresponding point pairs.

The observations of road centerline j at time k and its corresponding SD Map point are denoted as $\mathbf{z}_{i,k}^{\mathrm{RC}_j}$ and $\mathbf{y}_i^{\mathrm{SD}_j}$. They have the following form:

$$\mathbf{z}_{i,k}^{\mathrm{RC}_j} = \begin{bmatrix} {}^{\mathrm{c}}x_{i,k} & {}^{\mathrm{c}}y_{i,k} \end{bmatrix}^T, \quad \mathbf{z}_{i,k}^{\mathrm{RC}_j} \in \mathbb{R}^2 \tag{3.8}$$

$$\mathbf{y}_i^{\mathrm{SD}_j} = \begin{bmatrix} {}^{\mathrm{w}}x_i & {}^{\mathrm{w}}y_i \end{bmatrix}^T, \quad \mathbf{y}_i^{\mathrm{SD}_j} \in \mathbb{R}^2, \tag{3.9}$$

where $({}^{\mathrm{c}}x_{i,k}, {}^{\mathrm{c}}y_{i,k})$ are the coordinate of centerline observations under camera coordinate system. $({}^{\mathrm{w}}x_i, {}^{\mathrm{w}}y_i)$ are the coordinate of SD Map point under world coordinate system. Therefore, this single centerline factor has the following form:

$$\begin{aligned} \mathbf{r}_{i,k}^{\mathrm{RC}_j} &= \phi_{\mathrm{RC}}\left(\mathbf{x}_{i,k}, \mathbf{y}_i^{\mathrm{SD}_j}, \mathbf{z}_{i,k}^{\mathrm{RC}_j}\right) \\ &= \mathbf{z}_{i,k}^{\mathrm{RC}_j} - h\left(\mathbf{x}_{i,k}, \mathbf{y}_i^{\mathrm{SD}_j}\right) \\ &= \begin{bmatrix} {}^{\mathrm{c}}x_i \\ {}^{\mathrm{c}}y_i \end{bmatrix} - \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} {}^{\mathrm{w}}x_i - {}^{\mathrm{w}}x_{i,k} \\ {}^{\mathrm{w}}y_i - {}^{\mathrm{w}}y_{i,k} \end{bmatrix} \in \mathbb{R}^2. \end{aligned} \tag{3.10}$$

A single centerline factor is based on measurement model, and it has similar form with 3.6. Each car can obtain several road centerline points in each frame, that is to say, there will be several centerline factors. The summation of all factors at time

step k yields the overall centerline factor at that moment, which has the following form:

It is worth noting that the process of SD map alignment, which includes finding corresponding points, integrating into the factor graph, and solving the least squares problem may need to be performed several times to obtain the most accurate and consistent results.

The need for multiple iterations arises from the fact that both the vehicle's state and the nearest neighbor correspondences between the centerline and SD Map links are subject to change after each optimization step. Specifically, each time we solve the least squares problem, the vehicle's state - such as its position and orientation - gets updated. This, in turn, affects which points are considered the nearest neighbors between the centerline and SD Map links. Recalculating these correspondences and then re-optimizing leads to a more accurate alignment. Frequent updates are computationally expensive, which is why this recalibration is reserved for the interval between optimization iterations. Ultimately, the aim of this iterative process is not just to align individual points, but to achieve a comprehensive, global alignment between the entire centerline and the SD Map links. This holistic approach ensures more reliable and precise correspondences, thereby improving the overall quality of the SD Map Alignment.

Figure 3.8 illustrates the anticipated impact of the SD Map alignment process. The composite image consists of two segments: the left portrays the expected state before alignment, while the right represents the desired outcome post-alignment.

In the left segment, Car 1's lanemarker measurements are represented by a red dotted line, and Car 2's measurements are shown by a green dotted line. Discrepancies between these two measurements are evident. The vehicle trajectories are marked by arrows, with red corresponding to Car 1 and green to Car 2. Interspersed within this data is the black solid line, symbolizing the SD Map Links, serving as a consistent reference.

Transitioning to the right segment, the desired results of the alignment process are depicted. The divergence between the red and green dotted lines, along with their associated arrows, appears to be much smaller. This reduction in deviation suggests an improved alignment with the SD Map Links, highlighting the potential benefits of the alignment process.

Following the anticipated visualization, Figures 3.9 and 3.10 display the practical impact of the SD Map alignment procedure on real-world data.

In Figure 3.9, discrepancies between the trajectories of Car 3 and Car 17 and the SD Map are evident. The observed lane markings, represented by the dotted lines of Car 3 and Car 17, deviate significantly from the black solid SD Map Links, illustrating a notable misalignment. Contrastingly, in Figure 3.10, the trajectories of Car 3 and Car 17 and the observed lane markings exhibit a pronounced enhancement in alignment.

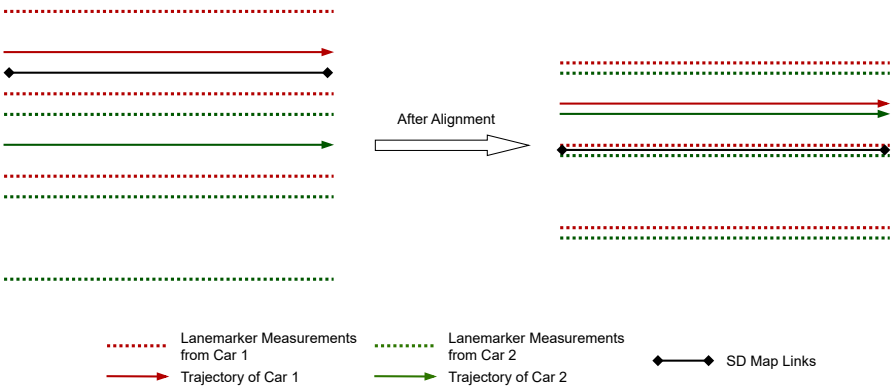

Figure 3.8: Anticipated comparison before and after SD Map Alignment. The right segment suggests potential improvement in data consistency between the two vehicles and the SD Map Links upon alignment.

Conclusively, these figures reinforce the validity of our proposed alignment method, illustrating its capability in harmonizing vehicular data from diverse sources with the SD Map. This alignment process not only demonstrates potential benefits in terms of data consistency but also highlights the potential enhancement in the accuracy of crowdsourced HD maps. Future application on more real-world data sets will further validate the effectiveness of this approach.

3.2 Multiple Drives

The Multiple Drives stage constitutes an essential step towards realizing a comprehensive autonomous driving map, where data from various individual drives is seamlessly integrated. This stage incorporates two crucial steps, namely, traffic sign alignment and lane marker alignment.

The sequence of these steps is deliberately chosen. First, we perform traffic sign alignment. Traffic signs exhibit unique features and are standardized, making their association across different drives largely unambiguous. This attribute allows us to establish a globally consistent framework within which the drives can be aligned accurately. Following traffic sign alignment, we proceed to the lane marker alignment. At this stage, the prior alignment using traffic signs provides us a refined starting point, which significantly enhances the accuracy of lane marker association.

The main objective of this stage is to harmonize multiple individual vehicle maps into a unified representation, with the ultimate goal of achieving an enriched, highly accurate, and consistent autonomous driving map.

3.2.1 Traffic Signs Alignment

Aligning multi-vehicle maps requires a robust and consistent feature, and traffic signs emerge as the most suitable choice. While lane lines are repetitive and terrain features sporadic, traffic signs offer standardization, stability, and frequency. Their unique attributes, like fixed positions unaffected by environmental wear, and

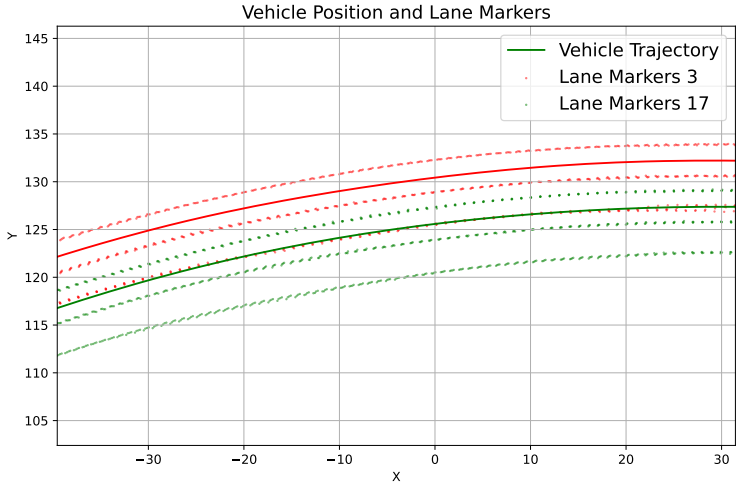

Figure 3.9: Before SD Map Alignment: The trajectories of the two vehicles and the observed lane markings show significant positional deviations, misaligning with the SD Map.

ubiquity across roads, make them an ideal, reliable source for alignment. Their standardized shapes, colors, and symbols simplify identification and localization, enhancing the alignment's effectiveness. Thus, traffic signs provide an feasible solution for aligning maps from multiple vehicles.

3.2.1.1 Find Common Traffic Sign Entities

The Traffic Sign Alignment operates under the reasonable assumption that, as vehicles traverse the same road segment, they might not observe the exact same set of traffic signs; however, their observed sets do overlap considerably. This understanding is based on the fact that traffic signs are fixed entities on the road, intended to be visible to all passing vehicles. Yet, the specific observations from each vehicle could vary due to differences in vantage points, trajectory inconsistencies, or observational errors.

Discrepancies can arise when transforming the observed traffic signs from the vehicle's coordinate system to a global coordinate system, not due to inherent flaws in the transformation process, but rather due to errors in the estimated vehicle pose that are propagated to the objects' positions. As a result, the same traffic sign might appear in different geographical locations across observations from various vehicles. Although the preceding alignment step with the SD Map aims to correct these positional differences, residual deviations can still persist.

Figure 3.11 illustrates the aforementioned issue. The red and blue circles in the figure contain observations of traffic signs from two different vehicles (labeled as 3 and 17), which are of the same type (type = 0). By observing the positions of these traffic signs, it is reasonable to assume that if these signs can be aligned through a transformation, then applying the same transformation would also result in the alignment of lanemarkers measurements from both vehicles.

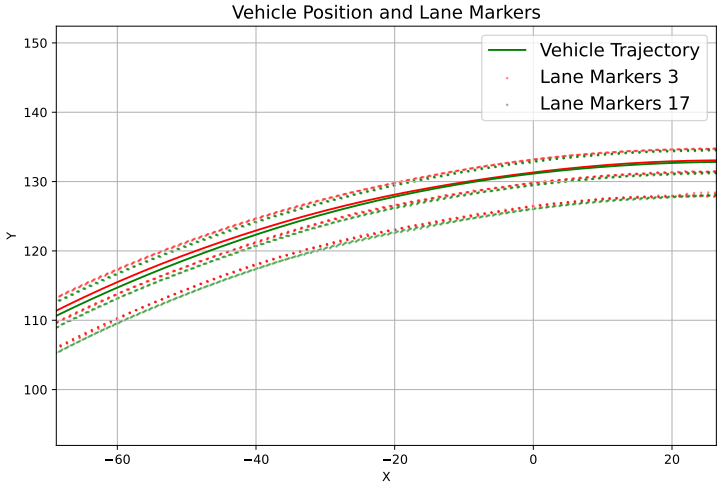

Figure 3.10: After SD Map Alignment: The trajectories of the two vehicles and the observed lane markings show significantly reduced positional deviations, aligning well with the SD Map.

To facilitate this, we designed a comprehensive process to identify and match traffic signs observed by different vehicles that correspond to the same physical entity. The process of identifying traffic signs that belong to the same entity, as depicted in Figure 3.12.

This process begins with generating potential matches. All signs from two separate vehicles are categorized by type, and potential matches within each type are identified, generating all possible combinations of traffic signs with identical types. For each combination, the average position and dimensions are calculated, and the discrepancy between individual signs and these averages is evaluated. A potential match is validated if the discrepancy falls within a predefined threshold. This method accounts for every plausible combination of traffic signs, as illustrated in Figure 3.13 (a)

Then, potential matches are sorted based on a metric derived from the average discrepancies in position and size within each combination. A lower value indicates a higher degree of match, and the matches are sorted in ascending order of this metric. Figure 3.13 (b) illustrates this sorting process.

Once sorted, potential matches are systematically inspected, and traffic sign entities are assigned. Each sign within a potential match is evaluated based on its positional accuracy and visibility. Unassigned signs form new entities and are added to the existing entity set. This process continues until all potential matches have been evaluated, and each assigned sign within the current match is tagged. Figure 3.13 (c) provides a visual representation of this step.

Finally, the process concludes with the output of a set of traffic sign entities, each comprising a group of traffic sign IDs observed by different vehicles but representing the same entity. This output set is then integrated into the factor graph, setting the stage for the subsequent alignment step.

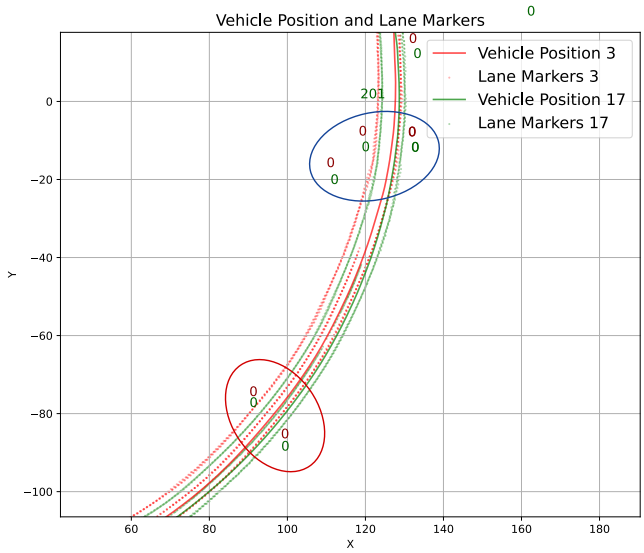

Figure 3.11: The discrepancy in the positions of traffic signs. Each number represents the type of traffic sign, while its position represents the position of traffic sign. The red and blue circles respectively represent two groups of traffic signs that should be in the same position.

3.2.1.2 Traffic Sign Entity Factor

In the previous section, we identified traffic signs that belong to the same entity across different vehicles. This information sets constraints between observations from distinct vehicles. By integrating these constraints into the overall factor graph, we can perform joint optimization of the trajectories of multiple vehicles. The aim is to eradicate or reduce map errors caused by deviations in vehicle trajectories.

Figure 3.14 elucidates the modifications in the factor graph during this operation. The green and red boxes depict separate factor graphs for two distinct vehicles. The structure of each vehicle's factor graph remains congruent with Figure 3.6, yet there is no initial connection between the factor graphs of different vehicles. The association illustrated by the traffic sign entities is shown in the graph as purple dashed arrows. The traffic signs at both ends of an arrow belong to the same entity, suggesting that their positions in the map should coincide. By leveraging this relationship, we can introduce a factor node between these two variable nodes.

This factor node takes the following form:

$$\begin{aligned} \mathbf{r}_{i,j}^{\mathrm{TS}_m,\mathrm{TS}_n} &= \phi_{\text{entity}}\left(\mathbf{y}_i^{\mathrm{TS}_m}, \mathbf{y}_j^{\mathrm{TS}_n}\right) = \mathbf{y}_i^{\mathrm{TS}_m} - \mathbf{y}_j^{\mathrm{TS}_n} \\ &= \begin{bmatrix} x_i^{\mathrm{TS_m}} \\ y_i^{\mathrm{TS_m}} \end{bmatrix} - \begin{bmatrix} x_j^{\mathrm{TS_n}} \\ y_j^{\mathrm{TS_n}} \end{bmatrix} \\ \mathbf{Q}_{i,j}^{\mathrm{TS}_m,\mathrm{TS}_n} &= \sigma_{entity}^2 \mathbf{I} \end{aligned} \tag{3.11}$$

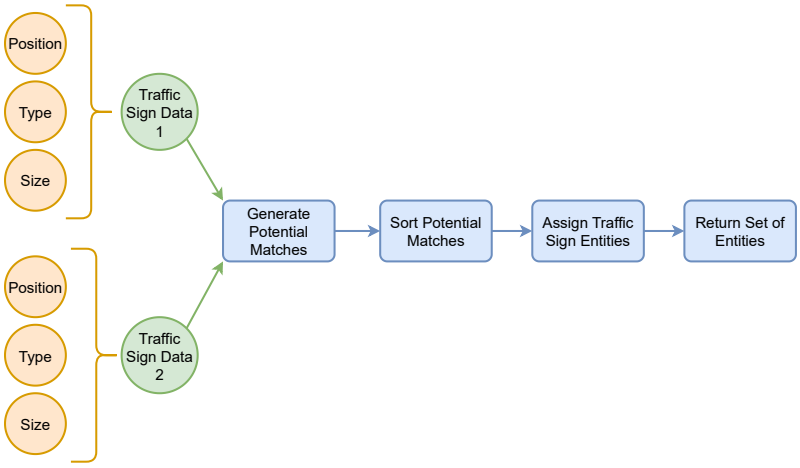

Figure 3.12: Flowchart for finding identical traffic sign entities

where $\left(\mathbf{y}_i^{\mathrm{TS}_m}, \mathbf{y}_j^{\mathrm{TS}_n}\right) \in \mathcal{E}_n$ is the traffic sign entity n and associated traffic signs, $\mathbf{y}_i^{\mathrm{TS}_m} \in \mathbb{R}^2$ is the position of the traffic sign m from vehicle i. This factor node actually expresses that the traffic signs belonging to the same entity should coincide, meaning that the difference between their coordinates should be zero.

Figure 3.15 presents the outcome after traffic sign alignment. It's evident that the traffic signs and lane markings observed by the two vehicles passing the same section are now aligned. The observations appear more coherent and organized.

Contrasting this with Figure 3.11, the issues before alignment become more apparent. Prior to the alignment process, the observed traffic signs had noticeable discrepancies, likely resulting from different vehicle perspectives and sensor inaccuracies.

This transformation underscores the significance and effectiveness of categorizing, matching, and verifying traffic signs by type. By leveraging the fixed and standardized nature of traffic signs, key inconsistencies in vehicle observations have been rectified. This improved alignment not only boosts the accuracy of the generated maps but also paves the way for subsequent steps in map generation and refinement. More accurate and consolidated data ensures a more precise representation of the road environment.

3.2.2 Lanemarkers Alignment

Upon completion of the Traffic Sign Alignment, we observe that there still exist discrepancies in the lanemarkers detected by different vehicles. The positions of traffic signs have been refined and coherently aligned across different local maps. However, the lanemarkers are still subject to variabilities and inaccuracies from various sources such as different vehicle perspectives, sensor noise, and environmental factors. Figure 3.16 exemplifies these disparities, with noticeable deviations present between the detected lanemarkers of different vehicles.

Inconsistencies between the lanemarkers from different vehicles may lead to inaccuracies when integrating the local maps into the global map. Moreover, due to the uncertainty of which vehicle's trajectory is the most accurate, it becomes impractical

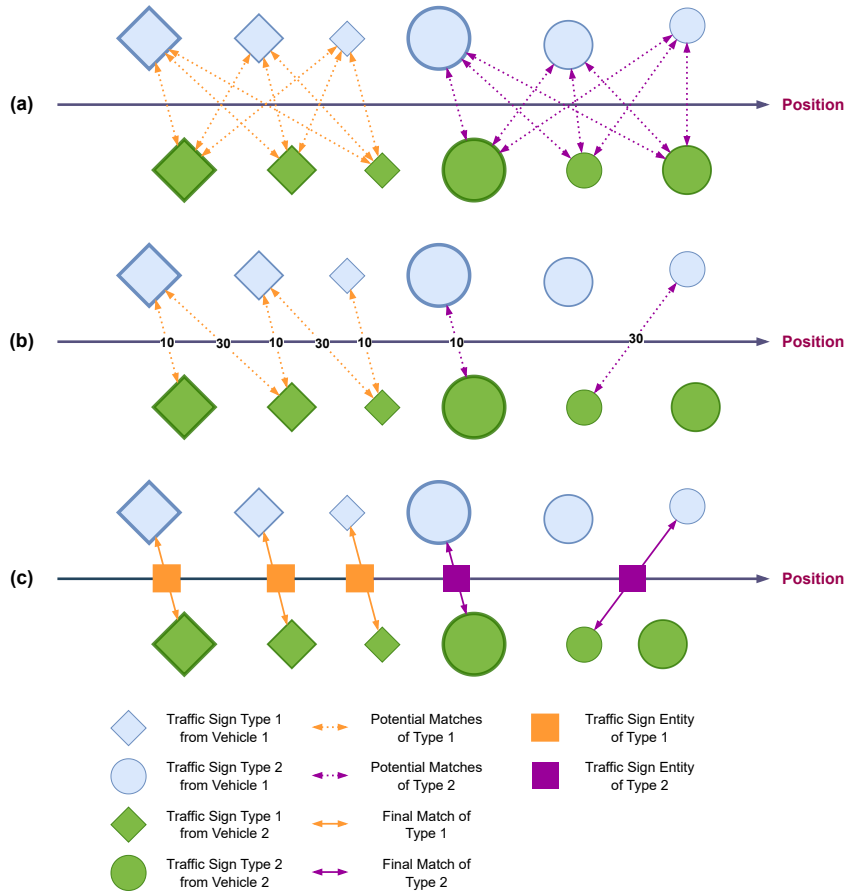

Figure 3.13: Illustration of finding common traffic sign entities. (a) Generate all potential matches based on types. (b) The remaining matches after filtering. Numbers represent matching scores, with lower scores indicating better matches. (c) Retain only one optimal match for each traffic sign and determine traffic sign entities based on this match.

to attempt to discern the exact position of the lanemarkers. In such cases, striving for absolute positional accuracy may not be the most reasonable approach. Instead, we adopt a pragmatic strategy of using the first vehicle's data as a basis, with the goal of aligning subsequent vehicle observations with it.

The Lanemarkers Alignment process is introduced to address these issues and provide a more coherent and reliable map. As mentioned above, its main purpose is not to derive the exact position of lanemarkers, but rather to reduce the discrepancy in the relative positioning of lanemarkers observed by different vehicles, effectively aligning them. Without this alignment step, the integration and update of lanemarkers could be problematic and error-prone.

To solve this problem, we adopt a SLAM approach, which necessitates the definition of an error function. The formation of this error function relies on the establishment of a correspondence between the lanemarker observations from two different vehicles' drives. Figure 3.17 shows the whole process of building such a correspondence. We first fit a polyline to the lanemarker observations from Car 1, aiming to capture

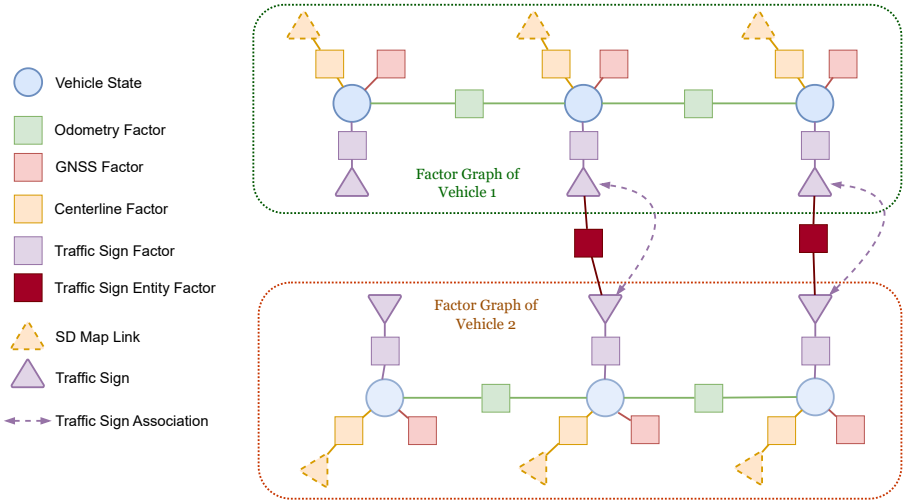

Figure 3.14: Structure of factor graph for alignment to SD map

the overall structure of the lane while reducing the influence of noise and irregularities. For each lanemarker observation of Car 2, we identify potential corresponding segments in the polyline from Car 1. This identification is subject to three conditions: 1) a line perpendicular to the polyline and going through the observation falls within the segment, 2) the observation and segment share the same type, and 3) the distance between the observation and the segment falls below a predefined distance threshold. Among the candidates that satisfy all three conditions, the corresponding segment chosen is the one with the minimal distance to the observation.

Upon establishing the association between the observed lanemarker points and the segments in the polyline, we proceed to the construction of the error function. Figure 3.18 illustrates this process. The error term is essentially the distance between the lanemarker observation and its associated polyline segment. Variables that need optimization are depicted by dashed lines in the figure, which include the pose of the vehicle corresponding to the observed point, and the positions of the two endpoints of the segment.

Given the aforementioned error function, we propose the factor graph for Lanemarkers Alignment as illustrated in Figure 3.19. The overall structure of this factor graph bears a significant resemblance to the structure depicted in Figure 3.6, with the exception that the lanemarker polyline point in this scenario requires updating.

The lanemarker factor has the following mathematical expression:

$$\begin{aligned} \mathbf{r}_{i,k}^{\mathrm{LM}_j} &= \phi_{\text{lanemarker}}\left(\mathbf{x}_{i,k}, \mathbf{y}_{m,1}, \mathbf{y}_{m,2}, \mathbf{z}_{i,k}^{\mathrm{LM}j}\right) \\ &= \mathbf{d}_{\text{obs-proj}} + w \cdot (\mathbf{d}_{\text{displacement},1} + \mathbf{d}_{\text{displacement},2}) \\ \mathbf{Q}_{i,k}^{\mathrm{LM}_j} &= \sigma_{\text{lanemarker}}^2 \mathbf{I}, \end{aligned} \tag{3.12}$$

where $\mathbf{d}_{\text{obs-proj}}$ represents the squared distance between the observed lanemarker position in the world frame and its projection onto the associated polyline segment.

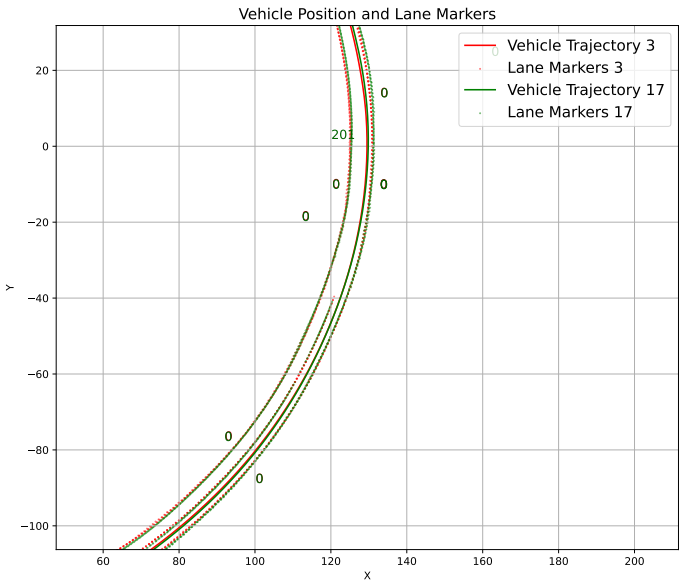

Figure 3.15: After Traffic Signs Alignment: The traffic signs and lane markings observed by the two vehicles passing through the same section are aligned.

It measures the deviation of the observed lanemarker position from the polyline segment. The specific calculation process of this item is as follows:

$$
\begin{aligned}
x_{\text{rot}} &= \cos({}^{\text{W}}\theta_{i,k}) \cdot z_{i,k}^{\text{LM}j,x} - \sin({}^{\text{W}}\theta_{i,k}) \cdot z_{i,k}^{\text{LM}j,y} \\
y_{\text{rot}} &= \sin({}^{\text{W}}\theta_{i,k}) \cdot z_{i,k}^{\text{LM}j,x} + \cos({}^{\text{W}}\theta_{i,k}) \cdot z_{i,k}^{\text{LM}j,y} \\
x_{\text{world}} &= x_{\text{rot}} +^{\text{W}} x_{i,k} \\
y_{\text{world}} &= y_{\text{rot}} +^{\text{W}} y_{i,k} \\
t &= \frac{(x_{\text{world}} - y_{m,1}^{x}) \cdot (y_{m,2}^{x} - y_{m,1}^{x}) + (y_{\text{world}} - y_{m,1}^{y}) \cdot (y_{m,2}^{y} - y_{m,1}^{y})}{\left\| (y_{m,2} - y_{m,1}) \right\|^2} \\
x_{\text{proj}} &= y_{m,1}^{x} + t \cdot (y_{m,2}^{x} - y_{m,1}^{x}) \\
y_{\text{proj}} &= y_{m,1}^{y} + t \cdot (y_{m,2}^{y} - y_{m,1}^{y}) \\
\mathbf{d}_{\text{obs-proj}} &= \sqrt{(x_{\text{world}} - x_{\text{proj}})^2 + (y_{\text{world}} - y_{\text{proj}})^2}
\end{aligned}
\tag{3.13}
$$

In Equation 3.13, x_{rot} and y_{rot} refer to the lanemarker observation coordinates in the world frame, obtained by rotating the original observation in the vehicle frame using the vehicle's orientation. These rotated coordinates are translated to x_{world} and y_{world} using the vehicle's position. The term t is the projection parameter used to project the world-frame lanemarker observation onto the polyline defined by its endpoints. Finally, the projected point's coordinates, x_{proj} and y_{proj}, are computed with this parameter.

The $\mathbf{d}_{\text{displacement},1}$ represents the squared displacement of the first endpoint of the

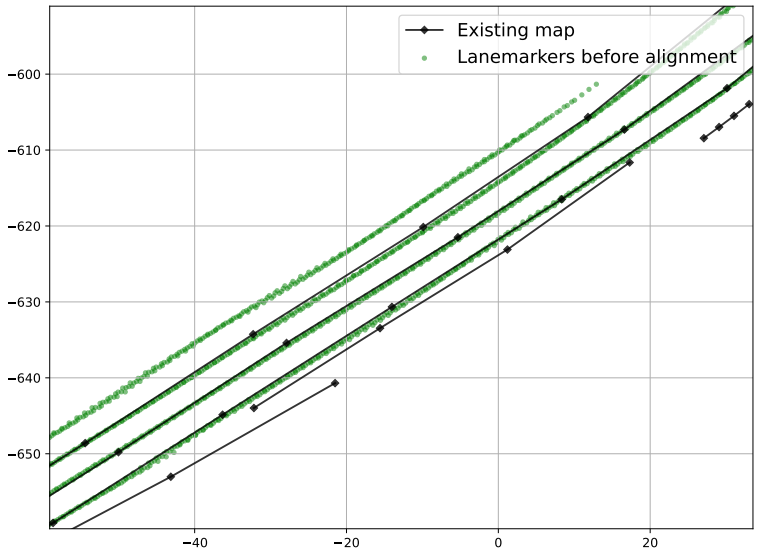

Figure 3.16: Discrepancies in lanemarkers from different vehicles persist even after traffic sign alignment.

polyline segment from its initial position. This displacement measure provides a regularization term that penalizes drastic changes in the polyline's shape. $\mathbf{d}_{\text{displacement},2}$ represents the squared displacement of the second endpoint of the polyline segment from its initial position. Like $\mathbf{d}_{\text{displacement},1}$, this term serves as a regularization measure to prevent extreme alterations in the polyline's structure.

$$\begin{aligned} \mathbf{d}_{\text{displacement},1} &= \sqrt{(y_{m,1}^{x} - y_{m,1}^{x,\text{init}})^2 + (y_{m,1}^{y} - y_{m,1}^{y,\text{init}})^2} \\ \mathbf{d}_{\text{displacement},2} &= \sqrt{(y_{m,2}^{x} - y_{m,2}^{x,\text{init}})^2 + (y_{m,2}^{y} - y_{m,2}^{y,\text{init}})^2} \end{aligned} \tag{3.14}$$

Figure 3.20 shows that the alignment process has successfully aligned the observed lane markings from different vehicles, resulting in a more coherent representation of the roadway. In contrast to the earlier discrepancies visible in Figure 3.16, the improved depiction offers a uniform portrayal of the lane structure.

Map alignment quality is essential for autonomous driving systems, as it forms the foundation for navigation. Minor errors in the map could create issues such as lane marking inconsistencies, which could lead to misinterpretations by the autonomous system. For instance, a misaligned or disconnected lane marker could be incorrectly interpreted as signaling a lane change, merge, or end of a drivable path, leading to navigation errors and potential safety risks.

Additionally, foundational errors like positional discrepancies between newly added vehicle data and the existing map can make map updates challenging. Without

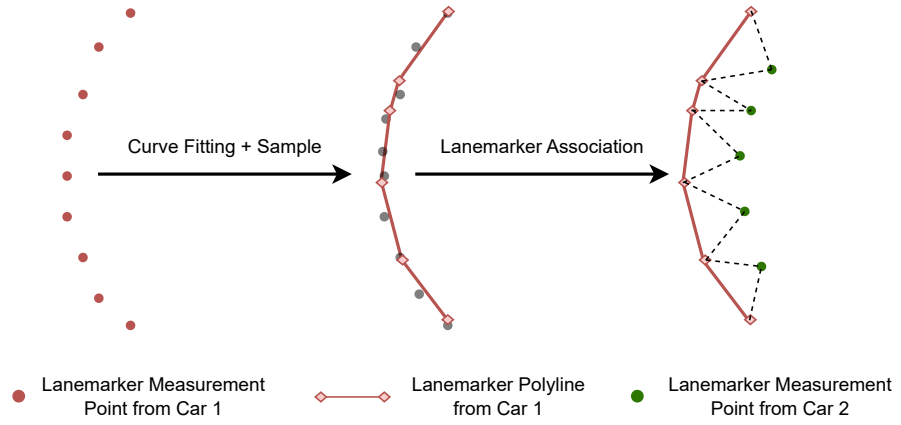

Figure 3.17: The process of finding associated line segment for each lanemarker measurement

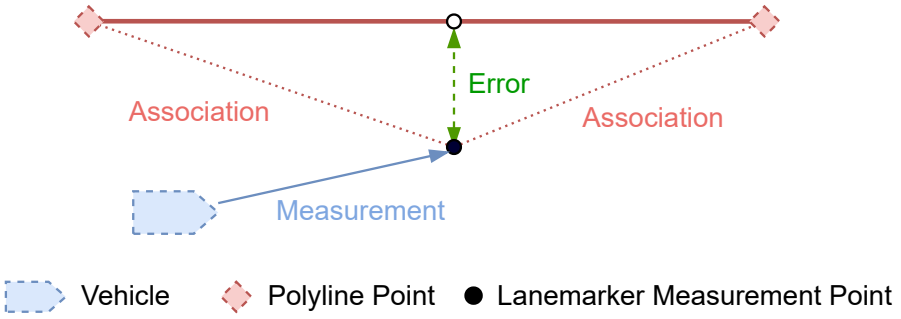

Figure 3.18: The process of constructing error function for lanemarkers alignment

accurate initial lane marker alignment, the system may have difficulty establishing a consistent basis for incorporating new observations. In such cases, the system might focus on continuously correcting these foundational errors, reducing the efficiency of map updates and increasing the risk of accumulating errors over time. Accurate initial alignment provides a stable foundation, simplifying the task of adding new data and helping to maintain the overall reliability and accuracy of the map.

3.3 Map Generation & Update

After successfully aligning the observations of both traffic signs and lanemarkers to our global map, we now proceed to the next crucial stage in our map-building processthe generation and updating of the map. This step combines the correctly aligned traffic sign and lanemarker data to create an integrated representation of the environment. Our approach is designed to utilize robust methods that aim to continually update and refine the map. While our results are preliminary, the system is engineered with the intent of accommodating new observations and adapting to changes in the environment. The goal is not only to enhance the accuracy of the map but also to strive for its current and reliable status, thereby facilitating better navigation and decision-making for autonomous driving systems. In the following

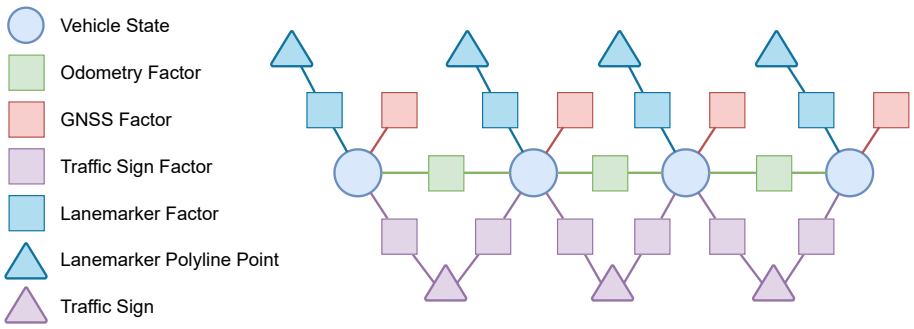

Figure 3.19: Structure of factor graph in lanemarkers alignment

sections, we will discuss the specifics of how we aim to generate and continually update this map, detailing the techniques and algorithms we employ, and elaborating on how they are intended to contribute to the overall efficacy of our mapping process.

3.3.1 Regional Map Generation & Update

In the context of map generation and update, we adopt a regional approach, where the entire area of interest is divided into numerous smaller regions for easier management and processing. By significantly reducing computational complexity, our strategy enables us to manage the large scale of the target area in a systematic way. For each region, a reference point is selected, and an East-North-Up (ENU) coordinate system is constructed based on this point. The alignment data for traffic signs and lanemarkers, as discussed in the previous sections, are represented within this regional coordinate framework, ensuring that each regional map is created in a localized context. This approach contributes to precision and consistency in map generation.

The overall strategy for map generation and update consists of two stages. Initially, data from the first vehicle is used to create an initial map, serving as a base layout and providing a starting point. In the subsequent stage, the initial map is designed to be continually updated using data from subsequent vehicles. The approach aims to make the map dynamic and responsive, with the intent of adapting to changes and incorporating new observations in the environment.

For the initialization process of the map, a straightforward and intuitive method is adopted. The first vehicle is selected, and its optimized lanemarker and traffic sign position data is directly incorporated into the map. Given these data elements have already been transformed into the world coordinate system through previous optimization steps, they can be readily added to the map.

The positions of these traffic elements are marked onto a two-dimensional map according to their positions in the world coordinate system. The creation of this initial map is essentially a process of digitizing real-world traffic elements into a virtual environment. By integrating these elements into a unified map, a basic layout reflecting the fundamental structure of the actual driving environment is obtained.

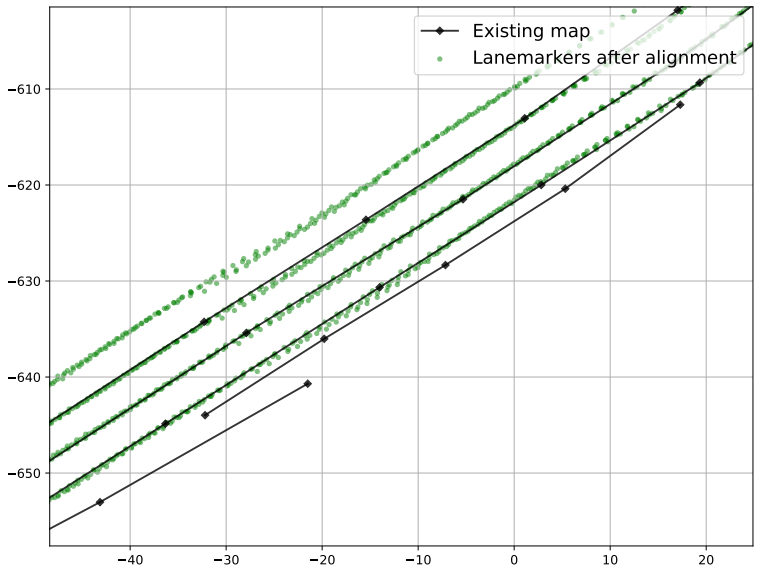

Figure 3.20: After Lane Markers Alignment: The lane markings observed by the vehicle are aligned with the existing lane information in the map.

With this starting point, an initial map provides a basis for further work. While this initial map only reflects the observation data of the chosen vehicle at the beginning, it can be updated and improved iteratively to adapt to changes and new observations in the environment. The subsequent discussion about map update will be divided into two parts, focusing on the two main components involved in this process: the updating of traffic signs and lanemarkers. The first part will delve into the methods employed to update the map with new traffic sign data, while the second part will detail the process of updating the lanemarker polylines.

3.3.1.1 Traffic Signs Update

The process of updating traffic signs on the map is relatively straightforward, relying on the established entities of traffic signs from previous stages. The fundamental idea behind this process is to merge observations associated with the same entity and create new entities for observations that haven't been linked to any existing ones.

This approach is rooted in the confidence level associated with each traffic sign observation. In the data filtering stage, a high confidence threshold is used to filter out false detections or erroneous observations of traffic signs. Thus, if a vehicle obtains a traffic sign observation that was not observed by a previous vehicle, it is less likely to be a false detection and should be considered a valid, existent traffic sign.

The basis for associating new traffic sign observations with existing entities is pred-

icated on the criteria of position, type, and size of the traffic signs. These criteria provide a reasonable level of precision in linking observations to the correct entity. The positions of traffic signs associated with the same entity are averaged to provide a singular, definitive position. Unassociated traffic sign observations, on the other hand, are directly incorporated into the map without modification.

This strategy makes the assumption that subsequent observations of the same traffic sign entity are accurate and relevant. It should be noted, however, that in reality, traffic signs could be removed, replaced, or obscured, potentially leading to inconsistencies in the mapping process. These potential discrepancies highlight an important area for future development, where mechanisms to detect and handle such changes could be considered. By doing so, the system would be able to ensure the integrity of the map over time, maintaining its accuracy and reliability even in the face of changing real-world conditions.

3.3.1.2 Lanemarkers Update

In the process of updating lanemarkers, the focus is primarily on the supplementation and extension of lanemarker polylines that were not fully observed or captured by preceding vehicles. This strategy is based on a core assumption that a single vehicle, during its journey, may not be able to observe and capture all relevant details due to various factors. For instance, the line of sight of a vehicle's sensors might be obstructed by other vehicles or road conditions, resulting in some lanemarkers being only partially observed or entirely missed. Moreover, the sensor data from the vehicle can be influenced by noise, leading to incomplete or inaccurate capturing of lanemarker polylines.

Therefore, the strategy for updating the map focuses on supplementing and extending lanemarker polylines, rather than modifying them. This decision is driven by two main considerations. Firstly, a conservative approach is used in the data filtering stage by setting a high confidence threshold for visual data, which is feasible given that the data volume is more than sufficient for map creation. Secondly, we generally refrain from making positional corrections to the polylines constructed by preceding vehicles. This choice is influenced by the way the 'lanemarker factor' is defined, as alignment errors between vehicles can significantly alter the geometry of the existing map, leading to instability.

To counteract this, we actually set weights for the existing map and the newly input vehicle data during the optimization process. A high weight is assigned to the existing map to minimize the influence of new observations from subsequent vehicles. This weighting approach is a provisional measure, and future work may involve fine-tuning these weights based on factors such as the age of the data. For instance, if a subsequent vehicle's data is significantly newer than the existing map, it may be assigned a higher weight.

Given a polyline representing a lane line observed by a vehicle, four main scenarios are considered:

Head Extension of the Polyline: This scenario involves subsequent vehicles initiating their lanemarker observations earlier than preceding vehicles, thus extending

the start of the polyline. Specifically, for every observation point in Car 2, the corresponding lanemarker polyline segment in Car 1 is found, following the method used in the **Lanemarkers Alignment** step. By sorting all observation points in order, a list of associated segments is obtained. If there are at least m consecutive points not associated with any polyline segment prior to the first point associated with the starting segment of Car 1, the head of the polyline is extended.

Tail Extension of the Polyline: Mirroring the process of head extension, the tail extension involves identifying at least m consecutive points not associated with any polyline segment after the last point associated with the ending segment. In such cases, the tail end of the polyline is extended.

Middle Gap Filling of the Polyline: This involves detecting a gap between the ending of a polyline and the start of another, with at least m consecutive points not associated with any polyline segment. This gap in the polyline is then filled.

Addition of a New Polyline: There are cases where subsequent vehicles capture entirely new lanemarkers that went unnoticed by preceding ones. These instances, where no observation point from Car 2 associates with a polyline from Car 1, result in the addition of new polylines to the map.

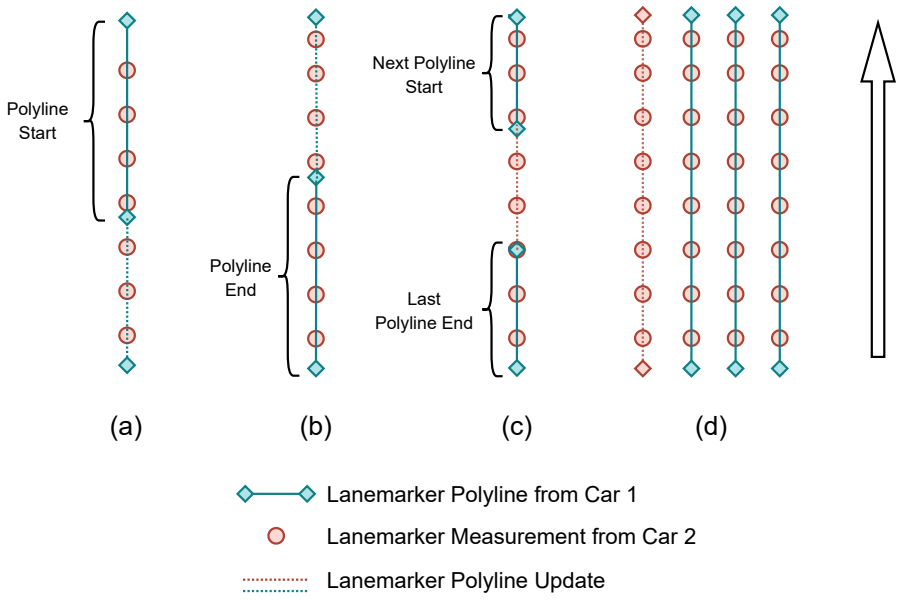

Figure 3.21: Scenarios when lanemarkers should be updated. (a) Head Extension (b) Tail Extension (c) Middle Gap Filling (d) Addition of New Polyline

These scenarios are visually depicted in Figure 3.21 with sub-figures (a), (b), (c), and (d) respectively representing the four cases described above.

3.4 Regional Map Stitching

A key step in the process of constructing a map at a global scale involves integrating multiple regional maps. In our methodology, the precise location information for

each regional map is calibrated through alignment with a globally unified SD Map, which helps to address positional offset issues that may occur across different regions. Nonetheless, the issue of gaps between regions remains a practical challenge that needs to be addressed.

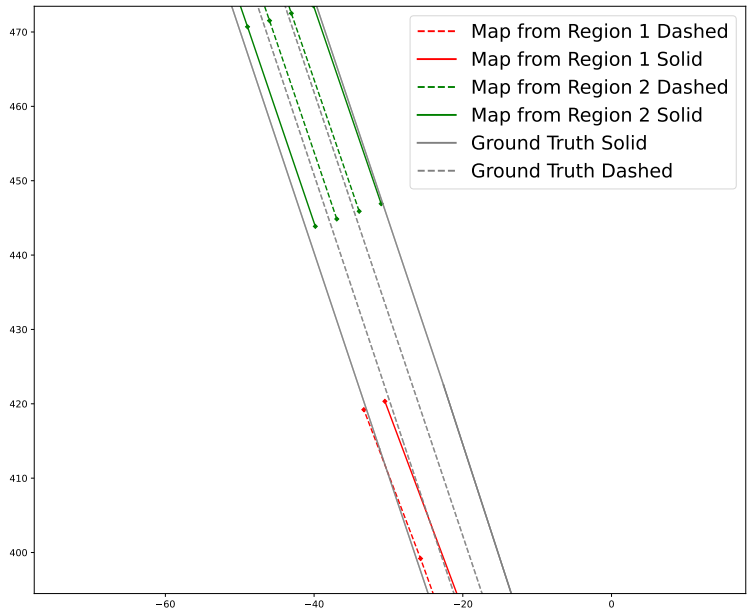

Figure 3.22: Schematic representation: A significant gap is observed between two regions when compared to the ground truth HD Map, indicating the presence of incomplete data.

As shown in Figure 3.22, the red and green lines represent lane markers from Regions 1 and 2, respectively, while the grey lines indicate the ground truth HD Map. Compared to the ground truth, a substantial gap can be observed at the junction between Regions 1 and 2.

These gaps are not caused by region division, but rather induced by errors in vehicle positional data. During the process of defining regional boundaries, we determine when a vehicle enters or exits a region based on its GNSS positional data. Errors in these data could potentially result in gaps between regions. Therefore, despite our method largely resolving map stitching problems, further research is needed to mitigate and manage these gaps caused by vehicle positional errors, to ensure a more seamless and comprehensive global map.

4

Neural Network Structure of Learning-based Method

To address the challenge of creating a unified, high-definition map from heterogeneous and decentralized sources, we introduce CrowdNet. This neural network leverages crowdsourced data from a fleet of vehicles to generate detailed road geometry. Each vehicle contributes its own observations of roads, which we call local maps, and is transformed into images. CrowdNet employs attention mechanisms to integrate these diverse local map images and decodes them to generate instance-level high-definition road elements, such as lane markers. Following this, the network will be composed of several stages: A set of local map images from crowdsourcing vehicles comes as input, together with a set of points and pseudo labels as connections from the previous frame, which, when inferencing, will be coming from the output of the previous frame. Every image in the set will be sliced into patches as its implementation in the ViT, and a zero-valued image, which serves as class tokens, will also be sliced into sequenced patches and concatenated to the beginning of the sequence of image patches. After these images go through a few transformer encoders, information from all the local maps will be aggregated into class tokens. The aggregated class tokens will go both ways, one of which is to go through a set of up-sampling layers, which restore its height and width same as the input image for the purpose of training for instance segmentation. It will also go through a transformer encoder-decoder structure, together with the query embedding, the decoder will be decoding a set of predictions, which will be later on sent to a few MLP layers for regression of geometry parameters and geometry attribution.

We denote number of patches from one image as N, number of drives that will be co-processed as N_d, for each timestamp. Overall network structure is shown in figure 4.1

4.1 Patches Projector

Each of N_d local map images in the set will be cropped to patches and projected to feature dimension through a linear projection layer

$$F_i = \left\{ \mathcal{F}_i \in \mathbb{R}^{(\frac{W_{img}}{W_{patch}} \times \frac{H_{img}}{H_{patch}}) \times C} \mid i = 1, 2, ..., N_d \right\}. \tag{4.1}$$

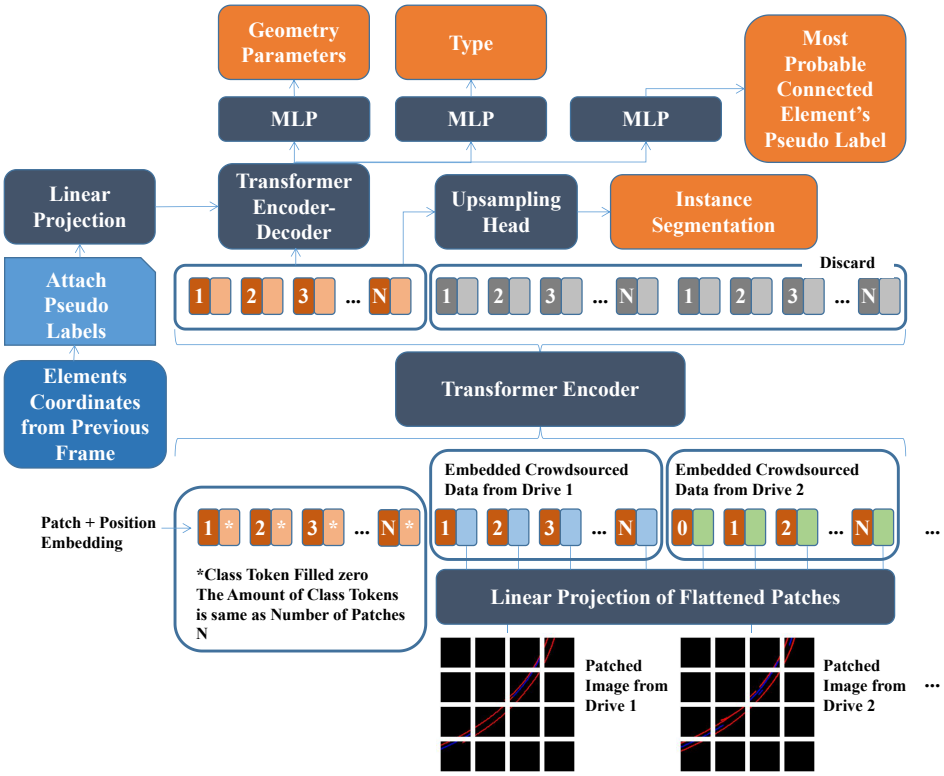

Figure 4.1: Network structure of the learning-based model.

We refer to each projected image patch as a token. A local map image contains $\frac{W_{\text{img}}}{W_{\text{patch}}} \times \frac{H_{\text{img}}}{H_{\text{patch}}} tokens. When N_d$ vehicles provide their data, the total number of tokens becomes $N_d \times \left(\frac{W_{\text{img}}}{W_{\text{patch}}} \times \frac{H_{\text{img}}}{H_{\text{patch}}}\right)$. These tokens are then stacked to produce a sequence of tokens, denoted as $\mathcal{F} \in \mathbb{R}^{(N_d \times \frac{W_{\text{img}}}{W_{\text{patch}}} \times \frac{H_{\text{img}}}{H_{\text{patch}}}) \times C}$.

4.2 Local-map Aggregator

The objective of crowdsourced data aggregator is to aggregate information from the feature sequence. The structure of this aggregator is a transformer encoder. Similar to the implementation in ViT, a set of class tokens $CLS \in \mathbb{R}^{(\frac{W_{img}}{W_{patch}} \times \frac{H_{img}}{H_{patch}}) \times C}$ is initiated with zero value and concatenated to the start of the sequence, which we would like the attention mechanism work to aggregate crucial information from different local map to class tokens. For each local map tokens and class tokens, the same positional encoding is added. This encoding provides information about the position of each token within the image. By using identical positional encodings across images, we ensure that the attention layers give equal consideration to each local map. The aggregated class tokens from this aggregator still have a long way to go before reaching the network's output, which makes it challenging to update the aggregator's model parameters using gradients. To address this issue, the class

tokens are not only passed to the subsequent stage of the network but are also processed through a set of upsampling and MLP (Multi-Layer Perceptron) layers. Up-sampled results will be compared with the ground-truth map, with a instance segmentation loss to assist the converge of local-map aggregator.

The reason we use a transformer encoder is that, considering the number of drives that are to be aggregated will always be different, if we follow some classical method, in which, we stack local maps in feature dimension brings a mutative size of input lacks flexibility. However, since transformer encoder layers can accept the input sequence at any length, if more drive from the fleet is coming, it only needs to be stacked to the end of the sequence. Moreover, we require a mechanism that aggregates the portions of the input information relevant to a specific position in the output. which was what the multi-head attention mechanism were designed for.

4.3 Connections Between Frames

Since we have sliced the local map into frames, every output from the whole network is only road elements in a small area, we would like to find the relationship of elements between neighboring frames. therefore, we design a special mechanism to solve this problem, First, when we were slicing frames, we would leave an overlap between neighboring frames, when generating maps, generated road elements from the previous frame falls into the overlapping area would be extracted by its coordinates, which is, in the application, the coordinate that is most close to the image border. Each selected road element will also be given a pseudo-label. Coordinates and pseudo labels would be embedded and sent into the road elements decoder, when it is decoding road elements, if it thinks a certain decoded element is connected to some element in the previous frame, it would output the same pseudo label as it was assigned to the element in the previous frame.

4.4 Road Elements Decoder

The network structure of the Road Elements Decoder is the same as a detection transformer without the CNN layers at the beginning, which, is composed of a set of transformer encoder and decoder layers, followed by MLPs for decoding results aiming at different tasks. However, In the original design of DETR, the output was structured as a set of bounding boxes, with each box represented by four values(x1, y1, x2, y2, constitute the bottom left and top right of the bounding box), while, our prediction will be modified to a set of six parameters which can be used to revert to a set of coordinates describing the road element's geometry. Similar to its implementation in DETR, the decoder would also output type information for each road element. At the same time, in the need to predict the relationship of road elements in neighboring frames, the decoder would also predict the pseudo labels introduced in the last part.

In the Local-map Aggregator (refer to section 4.2), a transformer encoder is already incorporated. However, in this particular segment of the network, we introduce

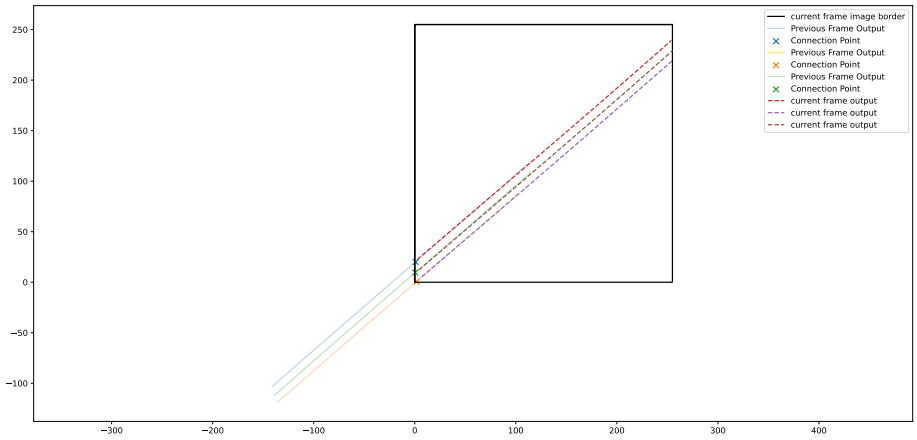

Figure 4.2: Illustration on how the connection points are selected from the output of the previous frame, the point that is closest to the image border from every predicted road element of the previous frame is chosen, and its coordinates together with a pseudo label will be input to the neural network

another transformer encoder, specifically designed to leverage its capacity to discern global relationships. This enables the separation of individual road elements, thereby creating a more refined input for the transformer decoder.

4.5 Discriminative Loss

In order to enhance the convergence of the crowdsourced data aggregator, relying solely on the loss obtained from the final output of the model (geometry, type, and connection) may lead to sluggish convergence due to the elongated path of back-propagation. To address this challenge, it is essential to introduce a loss function that aids in the convergence at the initial stages of training. To this end, we propose to upsample the output of the aggregator to match the original size of the image, subsequently utilizing the instance segmentation loss. This approach strategically assists the aggregator in optimization, providing a more efficient pathway to convergence. The discriminative loss function [41] was designed to resolve the problem of distinguishing individual instances within a class by simultaneously working on two facets: promoting intra-instance cohesion and enforcing inter-instance separation. These components are referred to as intra-cluster loss and inter-cluster loss, respectively.

The intra-cluster loss ensures that the pixels within a given instance are close together in the feature space by minimizing the distance between the feature embeddings of individual pixels and the centroid of the embeddings for the instance. On the other hand, the inter-cluster loss focuses on maximizing the distance between the centroids of different instances, thereby promoting distinctiveness between instances.

The application of discriminative loss involves the following steps:

- **Definitions**
  - N: Number of instances in an image.
  - M_i: Number of pixels in the i-th instance.
  - $\mathbf{f}_{ij}$: Feature embedding of the j-th pixel in the i-th instance.
- **Calculating Centroids**
  - **Step 1**: Calculate the centroid $\mathbf{c}_i$ of each instance i using the formula:

$$\mathbf{c}_i = \frac{1}{M_i} \sum_{j=1}^{M_i} \mathbf{f}_{ij}$$

- **Intra-Cluster Loss (Variance Term)**
  - **Step 2**: Compute the intra-cluster loss V_i for each instance i using:

$$V_i = \frac{1}{M_i} \sum_{j=1}^{M_i} \|\mathbf{f}_{ij} - \mathbf{c}_i\|_2^2$$

  - **Step 3**: Calculate the total variance term V for all instances:

$$V = \frac{1}{N} \sum_{i=1}^{N} V_i$$

- **Inter-Cluster Loss (Distance Term)**
  - **Step 4**: Compute the inter-cluster loss D_{ik} between instances i and k using:

$$D_{ik} = \max(0, 2\delta - \|\mathbf{c}_i - \mathbf{c}_k\|_2)$$

    where δ is a margin.
  - **Step 5**: Calculate the total distance term D:

$$D = \frac{1}{N(N-1)} \sum_{i=1}^{N} \sum_{k \neq i}^{N} D_{ik}$$

- **Regularization Term**
  - **Step 6**: Compute the regularization term R to control the magnitude of the centroids:

$$R = \frac{1}{N} \sum_{i=1}^{N} \|\mathbf{c}_i\|_2^2$$

- **Total Discriminative Loss**
  - **Step 7**: Calculate the total discriminative loss L as a weighted sum of V, D, and R:

$$L = \alpha V + \beta D + \gamma R$$

    where α, β, and γ are hyper-parameters.

The objective of the training process is to minimize this loss through optimization techniques, which leads to better separation and more accurate segmentation of instances.

In conclusion, the discriminative loss function presents an advanced method for enhancing instance segmentation by facilitating precise differentiation between instances within the same class. Its balanced focus on both intra-instance cohesiveness and inter-instance separation caters to applications that require stringent segmentation accuracy.

4.6 Discrete Geometric Loss

Similar to the implementation in [42], the intended output of our neural network will be in the form of a set of least square parameters, which we will introduce in detail in section 5.4.1. In [42], the geometric loss function is proposed to achieve the intended output. It is designed to minimize the squared area between the predicted curve and the ground truth curve in the image plane, up to a point t. This loss function has a geometric interpretation and is formulated as:

$$L = \int_0^t (y_{\text{pred}}(x) - y_{\text{gt}}(x))^2 \, dx \tag{4.2}$$

where $y_{\text{pred}}(x)$ is the predicted curve and $y_{\text{gt}}(x)$ is the ground truth curve.

However, our experiments reveal a potential limitation with geometric loss, as illustrated in figure 4.3. Two distinct outputs from the model may yield the same geometric loss value for a given ground truth, even though we would prefer model output (prediction at the right side) that more closely resemble the shape of the ground truth to have a lower loss value. Therefore, we discretized the loss function, As is shown in figure 4.4, areas of squares between model output and ground truth will be summed up as the loss value. At the same time, a variance value reflecting the difference in the areas of those squares is also incorporated into the loss value. This addition serves to assign a lower loss value to predictions that closely resemble the shape of the ground truth but do not completely coincide with it.

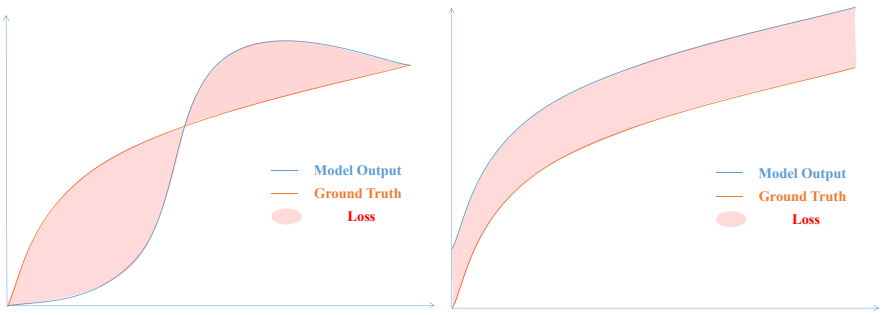

Figure 4.3: Two cases where geometry loss have a close value

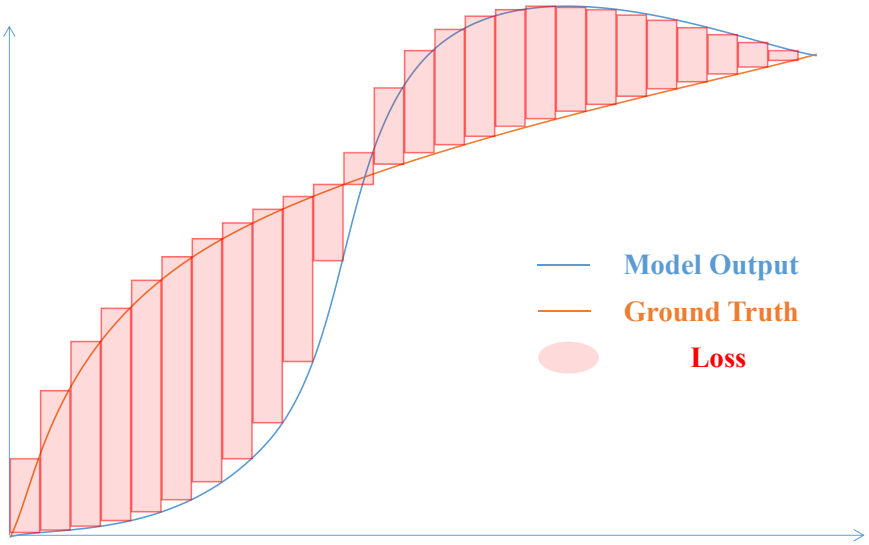

Figure 4.4: Discretized Geometry Loss

4.7 Generalized IOU Loss

The Generalized Intersection over Union (GIOU) loss [43] is a refined metric introduced to overcome the limitations of the traditional Intersection over Union (IOU) measure in object detection and segmentation tasks. Specifically, GIOU loss considers not only the overlap but also the geometric relationship between the predicted and ground-truth bounding boxes. Defined as $\text{GIOU} = \text{IOU} - \frac{C-U}{C}$, where C is the area of the smallest enclosing box and U is the union of the two boxes. $\text{IOU} = \frac{I}{U}$, where I is the intersection area of two boxes. GIOU provides additional geometric cues that enhance the model's understanding of bounding box configurations. By incorporating spatial geometry, GIOU contributes to more stable and robust training, effectively handling cases where traditional IOU may fail to calculate gradients between various non-overlapping box configurations.

In our implementation, the GIOU loss is specifically designed to ensure the accurate positioning of the starting and ending points of the road geometry. These points are critical as they help minimize accumulated errors when stitching together road elements across different frames. It's important to note that the starting point doesn't always have to be at the bottom left, and the ending point doesn't always have to be at the top right. We adjust these coordinates to meet the calculation requirements of the GIOU loss, thereby ensuring the integrity of the loss computation process.

However, it's crucial to understand that the GIOU loss is not responsible for the geometry between the starting and ending points. That aspect is managed by discrete geometric loss introduced in 4.6, which works in tandem with the GIOU loss to provide a comprehensive solution for accurate road geometry representation.

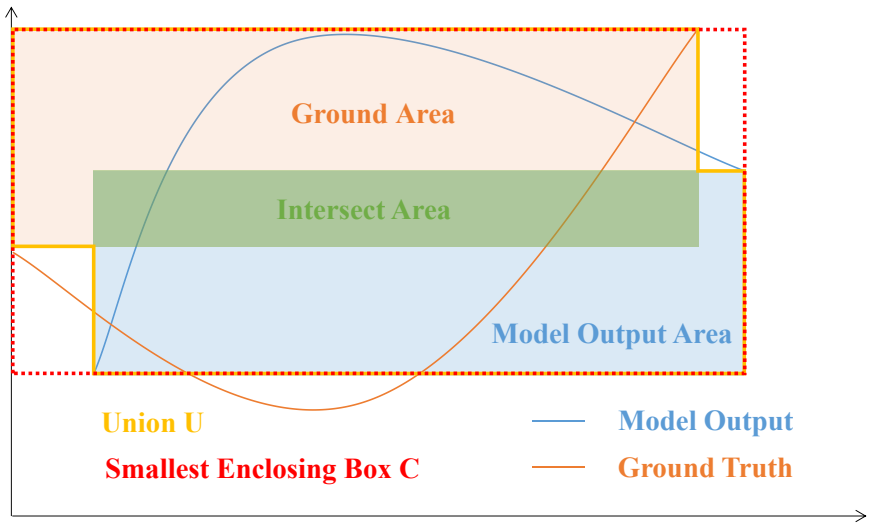

Figure 4.5: Generalized IOU Loss, when we are calculating, we would use the intersect area divided by union area as IOU, and using the smallest enclosing box area C to calculate GIOU with equation GIOU = IOU $- \frac{C-U}{C}$

5

Implementation

The implementation phase is essential for transitioning the proposed lane marking mapping methodologies from theoretical constructs to a functional system. This phase encompasses various processes, including data preprocessing, model implementation, parameter tuning, and performance optimization. Attention to detail in each of these aspects is necessary for ensuring the model's capability to handle real-world data reliably and efficiently. Subsequent subsections will detail these processes and discuss the specific considerations and decisions made during implementation.

5.1 Preprocessing of Crowdsourced Data

Our crowdsourced mapping process involves considerable preprocessing of data collected from vehicles. This data encompasses vehicle pose measurements, detected lanemarkers, and identified traffic signs.

5.1.1 Vehicle Pose Data

Vehicle pose data is derived from GNSS measurements and odometry readings. While GNSS measurements offer position data at the meter-level accuracy, they are susceptible to significant variances. Generally, GNSS isn't recognized as a source of high-precision pose data, prompting the incorporation of additional sensors, such as cameras, for positioning. Variabilities in GNSS data can introduce consequential inaccuracies or errors. Incorporating these measurements into map generation could result in discrepancies in the map, misalignments of roads, signs, and lane markings. Moreover, this may compromise vehicle position estimations, subsequently affecting processes like alignment with high-definition maps. Fortunately, the challenge is counteracted by odometry readings, which are typically more consistent and less affected by environmental factors, ensuring reliable positioning over short durations.

To ensure data consistency and reliability, each GNSS measurement's variance, including lateral, longitudinal, and yaw, is analyzed against a predefined threshold. Only data with all three variances below these thresholds is retained. These thresholds, set empirically, are 4 m^2 for lateral and longitudinal variance and 0.06 rad^2 for yaw variance.

The GNSS data is then interpolated to align with the timestamps of the odometry data. This process leverages linear interpolation, considering that the frequency of

the GNSS data is higher than that of the odometry data, coupled with the smooth continuity of GNSS signals in most areas.

5.1.2 Lanemarker Detections

Each detected lane line is categorized into four groups relative to the vehicle's x-axis position: left, right, second left, and second right. Each lane line is represented as a third-degree polynomial of the form $y = ax^3 + bx^2 + cx + d$, where (x, y) are positions relative to the vehicle center in the vehicle's coordinate system. An associated start and end distance, along with a validity flag, are also provided.

Firstly, lane line data with a validity of zero is discarded. Secondly, we sample x coordinates along the lane line polynomial to generate corresponding y coordinates. The range of x is defined by the start and end distances, within which the polynomial is valid. Our sampling strategy begins at the start distance, typically zero, and samples every 4 meters along the x-axis, limiting to a maximum of five sample points per lane line. This approach is a trade-off between preserving the shape of the lane line and computational efficiency. Closer detection points are prioritized, offering better accuracy and robustness.

5.1.3 Traffic Sign Data

Detected traffic signs are characterized by their position, type, size, tracking ID, and confidence. Due to potential false positives in traffic sign detection, we set a threshold that requires traffic signs to appear in at least 10 frames with a confidence level above 0.99 in each frame. This condition enhances the reliability of traffic sign detection, reduces the likelihood of false detections, and improves map accuracy. Traffic signs that do not meet these conditions are discarded.

5.2 Model Implementation

This project leveraged a combination of Python and C++ programming languages. Python played a pivotal role in data processing, association, and visualization of results. In contrast, the core solution for the SLAM optimization problem was formulated in C++.

5.2.1 Ceres Solver

Ceres Solver was adopted as the primary optimization tool for the SLAM task, driven by several pivotal factors:

- **Automatic Differentiation:** Manual computations of Jacobian matrices for cost functions can be intricate and susceptible to errors. The automatic differentiation capabilities of Ceres Solver alleviate these challenges, facilitating the optimization problem's construction.
- **Robust Kernel Functions:** The extensive suite of robust kernel functions offered by Ceres Solver ensures efficient handling of noise and outliers, pivotal

for achieving an accurate map.

- **Parallel Processing and CUDA Integration:** With an escalating complexity accompanying the increasing map data, the optimization problem demands more computational resources. Ceres Solver, equipped with parallel processing capabilities, efficiently utilizes multiple threads to expedite computations. Moreover, its CUDA support provides the opportunity to harness GPU's parallel computation power, contingent on the available hardware configuration.

With these merits, Ceres Solver was identified as the optimal tool for swiftly and efficiently constructing HD maps.

5.2.2 Optimization Algorithm and Solver Configuration

Within the Ceres Solver framework, the Levenberg-Marquardt algorithm was selected for optimization. The `SPARSE_NORMAL_CHOLESKY` linear solver was employed, which is adept at handling large-scale sparse systems. This choice is grounded in the following reasons:

- **Large-Scale:** The optimization variables mainly consist of vehicle poses and map elements, such as traffic signs and lane markings. With accumulating data, the optimization can encompass anywhere from hundreds to thousands of pose nodes, classifying the problem as large-scale.
- **Sparsity:** Many constraints in the optimization problem specifically target a subset of variables. For example, odometry constraints are limited to connecting adjacent pose nodes, GNSS constraints are exclusive to certain pose nodes, and traffic sign constraints are relevant only to the poses that detect the sign. A significant portion of the Jacobian matrix remains unpopulated, bestowing the problem with its sparse nature.

5.2.3 Selection of Robust Kernel

To manage outliers, we employed not only stringent data filtering strategies but also incorporated robust kernel functions. The choice of kernels was guided by empirical observations and the known error distributions of the data sources. Given the potential for substantial outliers in GNSS data, the Cauchy kernel was selected for the GNSS Factor. For other data sources, where the error characteristics differed, the Huber kernel was deemed appropriate.

5.3 Performance Optimizations

In the construction of high-precision maps, efficiency is paramount due to the sheer volume of data and the computational complexity involved. To ensure the system's robustness and speed, several optimization strategies were employed throughout the workflow.

5.3.1 Optimization Acceleration with Ceres Solver

First and foremost, the Ceres Solver's ability to leverage both multi-threading and CUDA considerably accelerated the solving of the optimization problem. This integration allowed the system to benefit from multi-core CPU architectures and, when available, GPU resources. Although real-time processing is not a primary requirement for our task, the enhanced computational efficiency is clearly beneficial. Faster computation enables more rapid iterations, leading to quicker results, which is advantageous for both development teams and clients.

5.3.2 Efficient Nearest Neighbor Search with K-D Tree

A recurring task in our pipeline was the search for the nearest neighbor given a point. This process was fundamental in various phases, from data association to error correction. For these searches, the K-D Tree (K-Dimensional Tree) data structure was the tool of choice. The strength of K-D Tree lies in its efficiency in handling multidimensional search queries. Instead of brute-forcing a search through all points to find the nearest neighbor, a task with linear complexity, K-D Tree typically narrows this down to logarithmic time complexity. Given that our project often involved searching among thousands of 2D or 3D points, the efficiency gains from K-D Tree were substantial.

5.3.3 Phased Optimization Strategy

Another noteworthy aspect of our approach was the phased nature of our optimization workflow. The whole process was designed in stages, wherein the optimized results of one phase served as initial values for the subsequent phase. This strategy ensured that each subsequent optimization task commenced with a promising starting point, thereby increasing the likelihood of convergence to a high-quality solution and reducing the overall computation time.

By weaving these optimization strategies into our system's fabric, we not only improved its computational efficiency but also enhanced the accuracy and robustness of our high-precision maps.

5.4 Dataset Preparation for Learning Based Approach

In the design of the neural network, the system takes a set of images of local maps from the fleet and outputs a set of parameters, with each parameter describing a road element's geometry in the area. In this section, we will detail the precise form of ground truth, explain how the training dataset and simulated evaluation dataset are generated, and describe the preprocessing of the data from the fleet, which is also used for evaluation. Additionally, we will discuss the methods and reasoning behind our handling of data when its geometry complicates the fitting process using the least square method.

5.4.1 Representation of Road Geometry in Ground Truth

Our work focuses on mapping highways. Road elements' geometry on the highway are quite different from urban areas, which have smoother curves and lacks sharp turns. If we are to use polylines with key points to represent the road geometries, we would need numerous key points to retain the geometry information. As stated in MapTR [17], decoding the long sequence of key points using the neural network brings great accumulated error. However, we can use the least square method to fit a set of parameters for representing road elements. The least squares method is a standard mathematical procedure for finding the best-fitting line or curve to a given set of data points by minimizing the sum of the squares of the offsets. Given a set of data points $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, the goal is to find parameters that fit a model such as $y = ax + b$ by minimizing the sum of squared differences $S = \sum_{i=1}^{n}(y_i - (ax_i + b))^2$. The optimal values of a and b are obtained by taking the partial derivatives of S with respect to a and b, setting them to zero, and solving the resulting equations.

Our intended output of neural network representing geometry of road elements will be represented by 6 values, $a, b, c, d, x_{start}, x_{end}$, from our investigation, curve calculated from these parameters can roughly describe the shapes of road elements that will appear on the highway, and the coordinates of road geometry can be restored from this parameter using:

$$x_{geometry} = x_{start} + i \cdot \frac{x_{end} - x_{start}}{step - 1} \quad | \; i = 0, 1, 2, \ldots, step - 1 \tag{5.1}$$

$$y_{geometry} = a \cdot x_{geometry}^3 + b \cdot x_{geometry}^2 + c \cdot x_{geometry} + d \tag{5.2}$$

The *step* is manually selected, the more *step* is, the closer these geometry points to actual curve represented by parameters, in our case, $step = 256$.

5.4.2 Processing Road Elements in North-South Direction

Representing road elements using parameters offers convenience but also poses challenges. One such challenge arises when dealing with roads that run almost parallel to the vertical axis, particularly those oriented in the north-south direction. In such cases, using the least-square method for fitting becomes problematic.

The issue stems from the fact that when a road's orientation has a significant angle difference from the x-axis, the parameters required to represent this road can become extremely large. These large parameters are difficult for the model to accurately regress, leading to potential inaccuracies in the representation of the road geometry.

To address this, we calculate the angle (θ) of the vector that starts from the first point and ends at the last point of each sliced trajectory. If this angle falls within certain ranges ($\frac{7\pi}{16} < \theta < \frac{9\pi}{16}$ or $\frac{-9\pi}{16} < \theta < \frac{-7\pi}{16}$) that make it difficult for the model to regress the parameters, we rotate the local map of the vehicle to bring the road's

orientation outside of these problematic ranges. We would also record the angle so that when the model outputs the road geometry, we can rotate the map elements back to the original direction.

5.4.3 Training and Evaluation Dataset Generation

To the best of our knowledge, there hasn't been any open dataset about crowdsourced mapping, which requires us to make a dataset of our own.

When we are making crowdsourced data, we generate synthetic shapes as found in a typical HD map in the image coordinate system. We would perform a least-square calculation to fit a set of parameters that best describe the geometry to serve as part of the ground truth. Also, their attributes like type information and connected elements' coordinate transferred to the image coordinates will also be generated.

We would also create input to the neural network through adding noise to the ground truth. In order to simulate the bias brought by GNSS, which was around meter-level, we would add noise to the data both along the driving direction and vertical direction. Also, we would randomly erase part of the road geometry to simulate occlusions, sometimes, a whole lane marking can be erased on the training data. We would also draw randomly generated curves to simulate the case that sometimes the vehicle's camera algorithm produces hallucinations on lane markings.

These data consist of around 350000 samples, each containing road elements of a fragment of the road with a length from 50-200 meters. The simulated evaluation dataset will be selected from the whole dataset and the rest constitute of the training dataset.

5.4.4 Preprocessing the Crowdsourced Data from Real Vehicles for Inferencing

Crowdsourced data from real fleeting running is crucial for our evaluation of the performance of the neural network. The crowdsourced data needs to be preprocessed since they are stored in the form of the vehicle's location(latitude and longitude), pose(Euler angle) and the detection points of the lane markings from the vehicle's mounted camera (in the vehicle's ego coordinate system, converted to BEV plane) at each timestamp. We need to first rotate the detection of lane markings according to the vehicle's pose, and then add them with the vehicle's own position, so that we can get detection points of lane markers in world coordinates along the trajectory of which the vehicle has passed by during its journey.

5.4.5 Slice and Projected to Image Coordinate System

The neural network accepts as input a sequence of images, with each image containing a canvas that illustrates key points of road elements. Whether derived from actual vehicles or simulations, these images represent an area spanning several kilometers in both length and width. If we were to correspond one pixel in the image to a real-world grid size of $1\,\mathrm{m} \times 1\,\mathrm{m}$, the resulting image size would become unman-

ageable for the neural network, not to mention the accompanying information loss that could occur with an increase in grid size. To tackle this challenge, we divide the entire area into patches.

More specifically, we begin by extracting a vehicle's trajectory and segmenting it into individual pieces. For each of these sliced trajectories, we identify the bottom-left and top-right coordinates, which together define a bounding box. This box is then used to extract key points of lane markers from all vehicles' local maps within the specified area. Additionally, we subtract the bottom-left coordinates of the bounding box from the lane marker coordinates. This adjustment enables us to represent the lane markers within the image coordinate system, thereby translating real-world spatial data into a format that our neural network can process effectively.

In addition, it is necessary for the images within the list to have consistent dimensions, yet the sizes of the trajectory bounding boxes often differ. To address this discrepancy, we compute scaling factors for height and width as $S_{\text{height}} = \frac{H_{\text{bbox}}}{H_{\text{img}}}$ and $S_{\text{width}} = \frac{W_{\text{bbox}}}{W_{\text{img}}}$, among the list of images, respectively. These factors are then utilized to scale the lane marker key points in such a way that they occupy the entire image. Along with these scaling factors, we also store the coordinates of the bounding box's bottom-left corner. Together, these elements enable an effective conversion between the image coordinate system and the world coordinate system, ensuring a coherent representation across different scales and contexts.

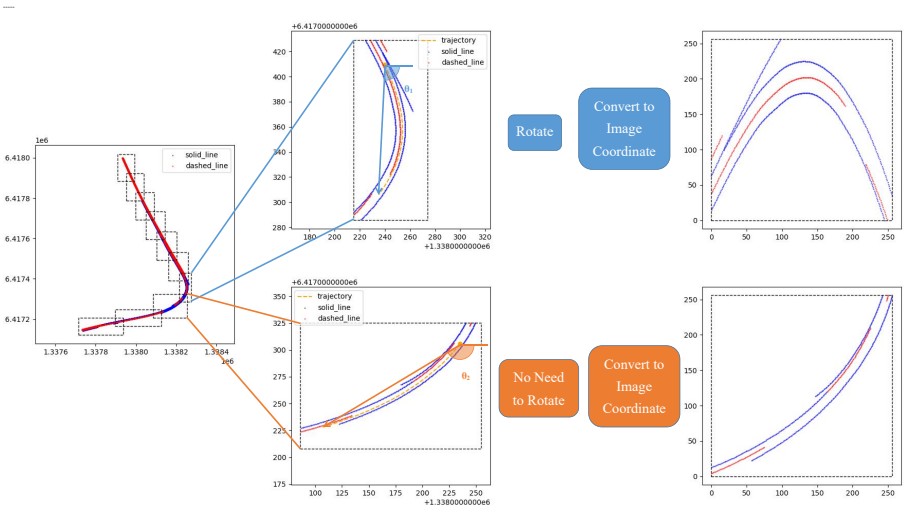

Figure 5.1: Data pre-processing procedures.

5.5 Learning Based Neural Network Training Details

The neural network and training script are written in PyTorch and the network is trained parallel on multiple GPUs. PyTorch provides the DistributedDataParallel module which uses collective communications in the torch.distributed package to synchronize gradients and buffers. It parallelizes the model by replicating it on each

GPU, and each replica is responsible for computing the gradients for a subset of the data. Each GPU is assigned batches in the size of 24. The model will be trained for 100 epochs.

The optimizer of the neural network will be the AdamW optimizer, a variant of the Adam optimizer that includes weight decay, making it suitable for various deep learning tasks. The learning rate, set at 1×10^{-4}, controls how much the model's weights should be updated during training; a smaller learning rate often leads to more precise, though slower, convergence to a minimum loss. In addition to the AdamW optimizer, we will implement a cosine annealing warm restart learning rate scheduler to control the learning rate during training. Cosine annealing adjusts the learning rate according to a cosine function over a predefined number of epochs, while 'warm restarts' periodically reset the learning rate to its initial value. This combination allows for potentially faster convergence and the ability to escape local minima in the loss landscape, enhancing the training process.

6

Evaluation & Results

This chapter assesses the methodologies and results of both Optimization-Based and Learning-Based Approaches to high-precision mapping. The evaluation focuses on two main criteria: scalability, which is the capacity to amalgamate data from multiple vehicles into a unified model, and accuracy, measured by the deviation between the generated maps and a pre-defined ground truth. Each approach employs a unique methodology for constructing maps, requiring distinct evaluation metrics. While this makes direct comparisons challenging, both methodologies are systematically evaluated against an established ground truth. This allows for an objective analysis of the advantages and limitations of each method, contributing to future improvements in high-precision mapping for autonomous vehicles.

6.1 Evaluation for Optimization-Based Approach

6.1.1 Evaluation Method

In this section, we aim to evaluate the performance of the autonomous vehicle mapping system based on two main criteria: accuracy and completeness. These criteria help determine the system's ability to represent the real-world environment faithfully.

For accuracy, our primary concern is the spatial positioning of lane markers. To evaluate this, we use the dataset provided by Zenseact, which comprises crowdsourced data, SD, and HD Maps. The HD Map from Zenseact serves as our ground truth. When comparing the lanemarker polylines from our map to the ground truth, the metric we use is the average Euclidean distance. Given the limited number of points on our generated polylines, we implemented an upsampling approach. This involves adding points at 1m intervals to the polyline to provide a denser representation. This densification ensures a thorough comparison by measuring the distance of each upsampled point to the ground truth. The process also facilitates a detailed observation of error distribution across the entire map.

For completeness, our focus is on ensuring that our generated map covers the entirety of the real-world environment. The generated map should capture all lanes and their characteristics. Additionally, while assessing traffic signs, our system seems to identify a greater number of signs than the ground truth. This discrepancy indicates that the ground truth may not have documented every traffic sign, so we will visually compare the results to assess the system's performance in this regard.

The evaluation area consists of a rectangular region with dimensions approximately 3464m by 2910m. This area was selected due to its diverse range of road conditions, including straight paths, curves, off-ramps, on-ramps, overpasses, and underpasses. For evaluation purposes, this rectangular region is divided longitudinally into two equal-sized sub-regions, subsequently referred to as 'Region 01' and 'Region 02.' These divisions enable a focused evaluation of the system's capacity to manage varying scenarios.

The map generation process unfolds incrementally, initiating with data from the earliest available vehicle and being updated as new data is collected. This incremental approach ensures both continuity and the enhancement of map accuracy. Following each update, the map undergoes a rigorous comparison with the established ground truth to identify and rectify discrepancies, offering a dynamic representation of how the map evolves over time.

6.1.2 Evaluation Results

Before presenting the evaluation results, it is necessary to clarify the choice of data representation methods. Box plots and histograms are utilized to display the positional errors for individual drives in Regions 01 and 02. The box plot serves as a compact summary of central tendency and data variability, capturing essential statistical metrics like the median, quartiles, and outliers. In contrast, histograms are employed to show the frequency distribution of the data, giving insight into the distribution of error values.

Figure 6.1 and Figure 6.2 together provide a comprehensive view of the positional errors associated with individual vehicles in Region 01. The box plot in Figure 6.1 shows that these errors generally hover around a 1-meter range, with each box representing the statistical distribution of errors for a single vehicle. This includes critical indicators like the median, quartiles, and average error. These discrepancies indicate inherent variations in map construction among different vehicles, primarily due to sensor inaccuracies and observational errors. Further nuanced details are revealed by the histogram in Figure 6.2, which shows that the errors are most frequently concentrated within the 0-2 meter range, specifically peaking around 1 meter. Additionally, a limited number of errors exceed 4 meters, possibly due to association errors and serving as points for further investigation.

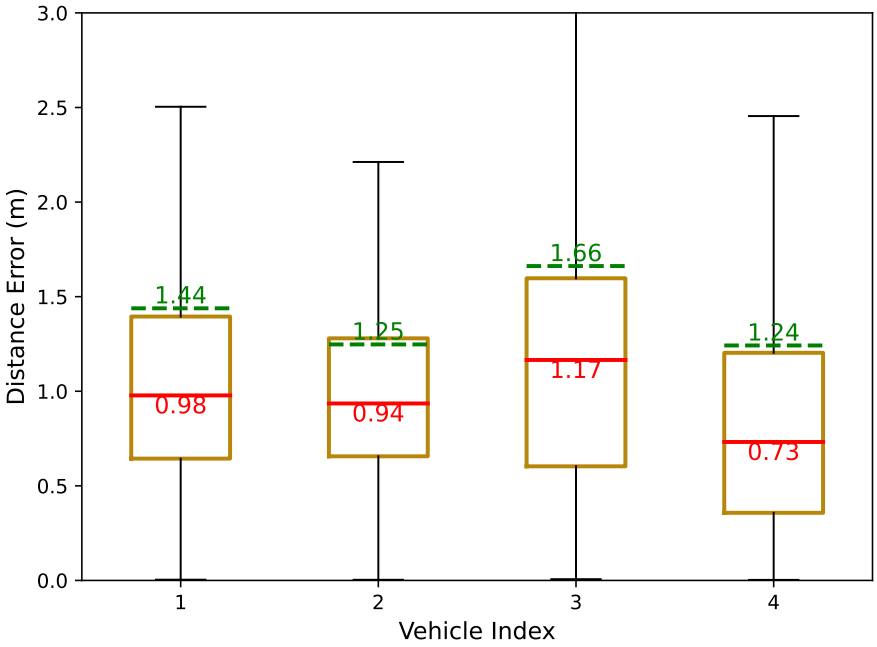

Figure 6.1: Box Plot: positional errors of four individual drives in Region 01

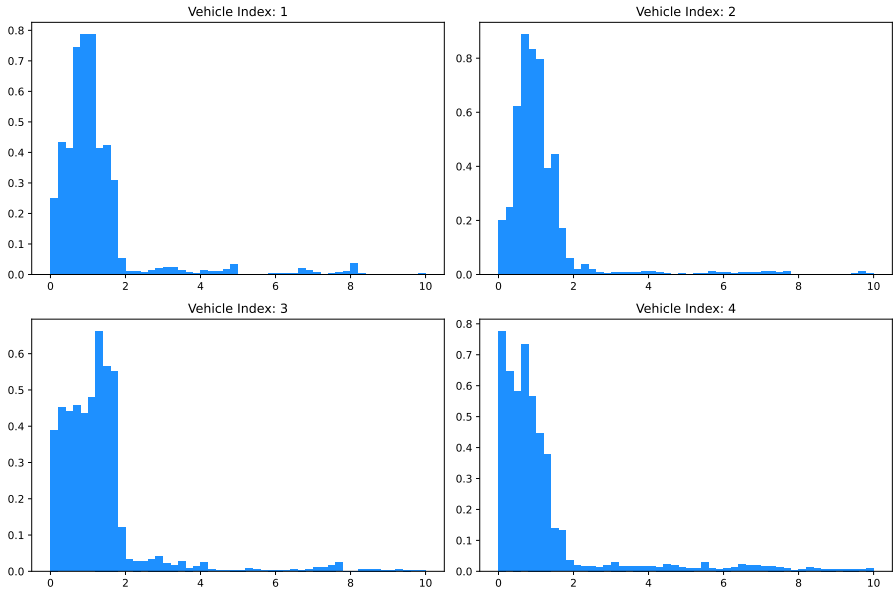

Figure 6.2: Histogram: distribution of positional errors of four individual drives in Region 01

Figures 6.3 and 6.4 collectively show the changes in map accuracy as data from one to four vehicles are successively incorporated into the model. The box plot in Figure 6.3 manifests that the aggregate positional errors largely correspond with those of the first vehicle's data, which serves as the initial reference. This suggests that the model's overall accuracy is robust to subsequent data inputs, maintaining stability in the presence of incremental data sets. This observation is corroborated by the

histogram in Figure 6.4, where the error distribution exhibits a higher concentration within the 0-2 meter range as more data is added. This consolidation in the histogram serves as further empirical evidence of the model's robustness in sustaining map precision.

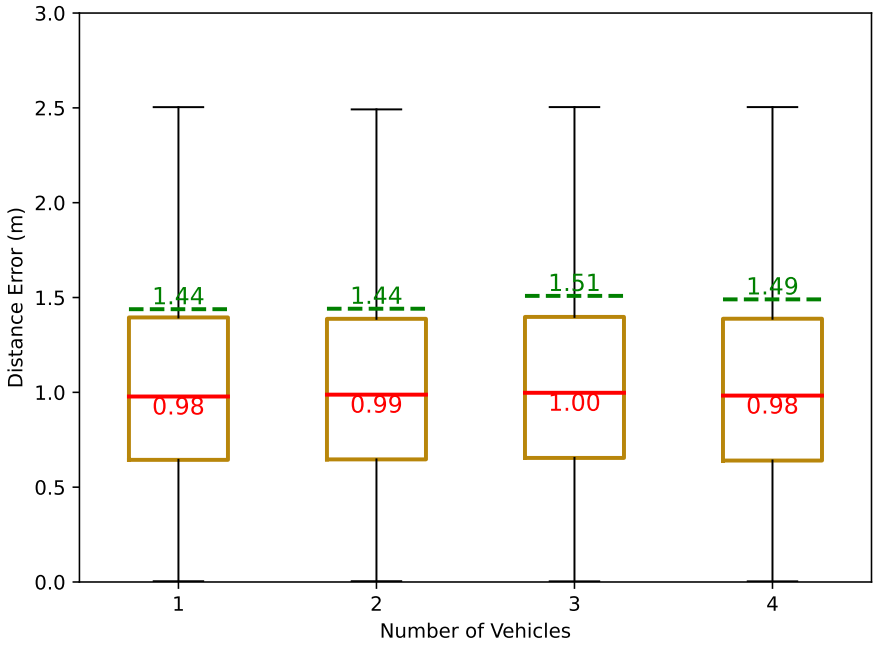

Figure 6.3: Box Plot: accumulated positional errors with more individual drives' data in Region 01 input to the model

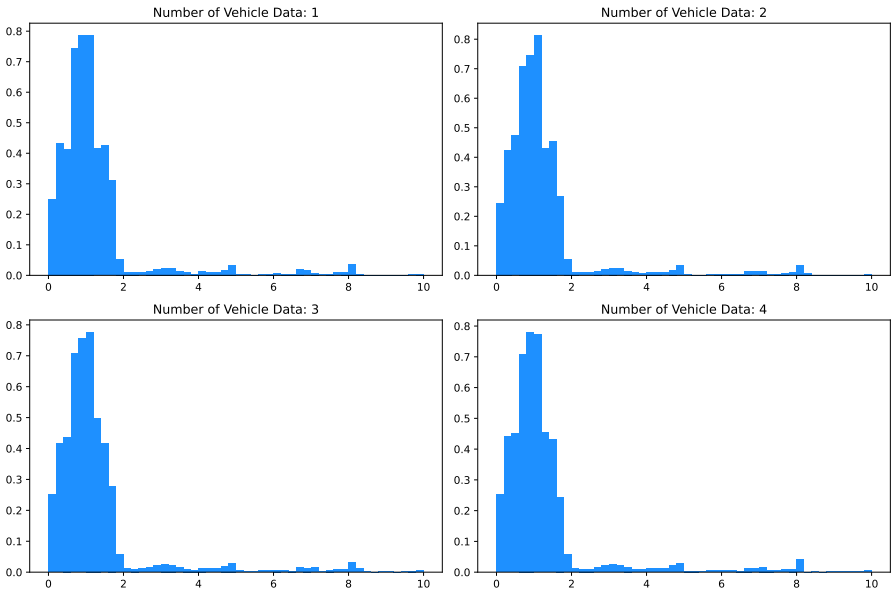

Figure 6.4: Histogram: Distributions of accumulated positional errors with more individual drives' data in Region 01 input to the model

Figures 6.5 and 6.6 analyze the positional errors for four individual vehicles in Region 02. The box plot in Figure 6.5 reveals that three of the four vehicles have positional errors with mean and median values near 0.5 meters, exhibiting greater accuracy than vehicles in Region 01. However, Vehicle 3 presents an exception with notably higher errors. This inconsistency is corroborated by the histogram in Figure 6.6, which shows a frequency distribution for Vehicle 3 that diverges significantly from the others, indicating substantial discrepancies relative to the ground truth. Such large deviations in individual vehicle data pose challenges to the integrity and robustness of the overall mapping model.

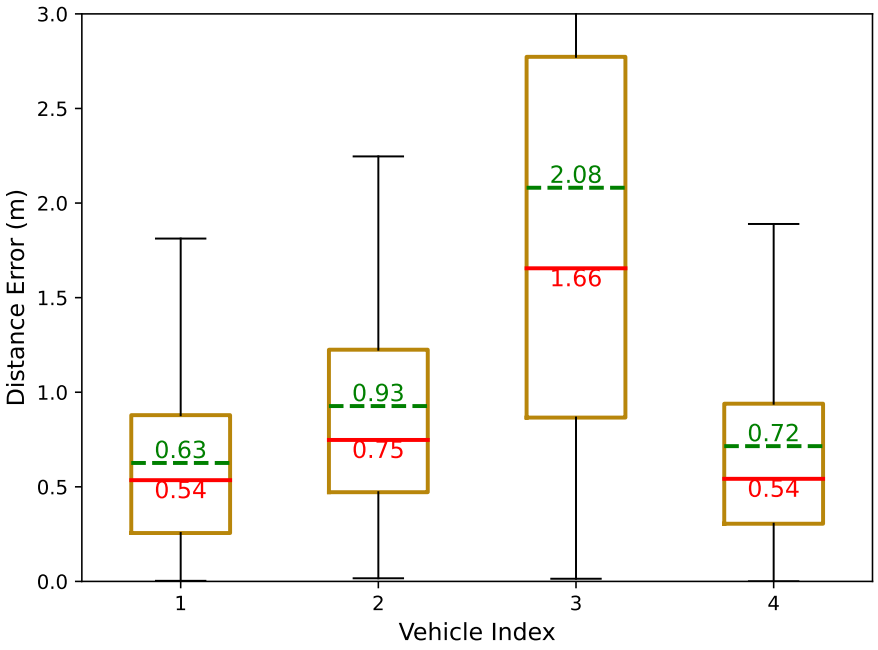

Figure 6.5: Box Plot: Positional errors of four individual drives in Region 02

Figures 6.7 and 6.8 illustrate the accumulated positional errors as data from the four individual vehicles in Region 02 are sequentially fed into the model. The box plot in Figure 6.7 demonstrates that the model's accuracy remains largely unaffected by the integration of Vehicle 3's higher-error data. The positional errors' distribution remains stable, suggesting that the overall mapping accuracy is robust to inconsistent data inputs. This observation is further substantiated by the histogram in Figure 6.8, which shows that even with the incorporation of more error-prone data, the frequency distribution of errors remains tightly clustered. These findings highlight the robustness of our model in maintaining high mapping accuracy even when faced with varying data quality.

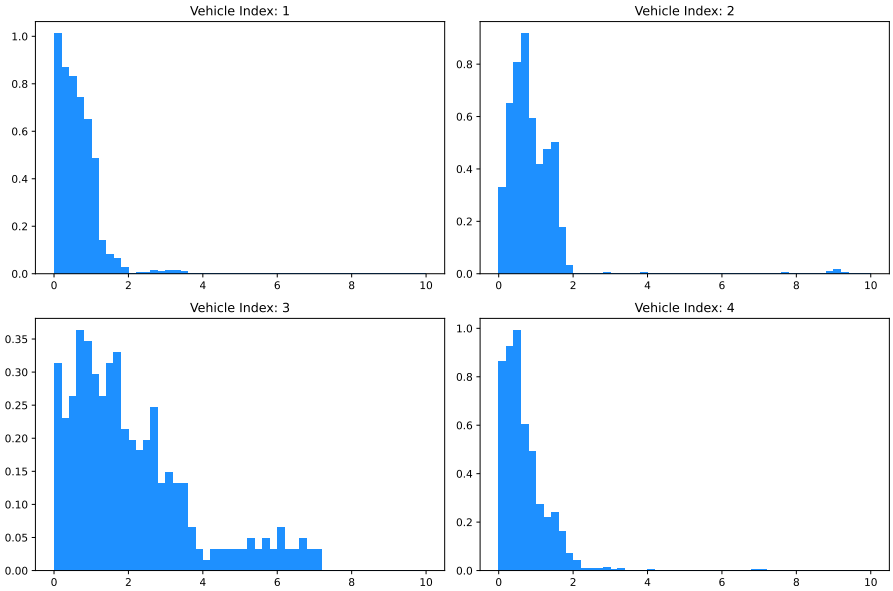

Figure 6.6: Histogram: Distribution of positional errors of four individual drives in Region 02

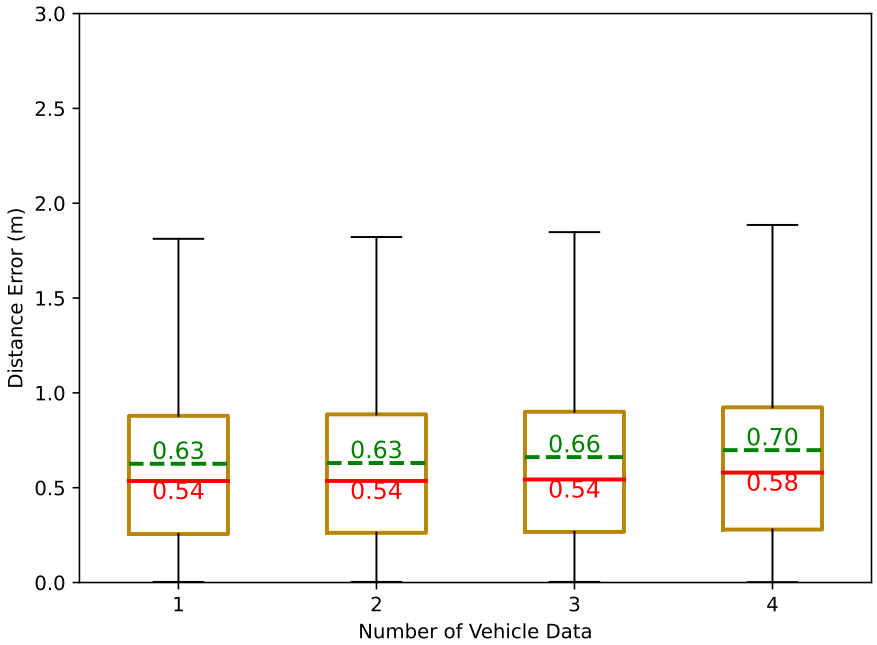

Figure 6.7: Box Plot: Accumulated positional errors with more individual drives' data in Region 02 input to the model

Following the statistical analysis, we present a visual comparison to provide a more intuitive understanding of the updates and improvements on the generated maps with the incorporation of new vehicle data.

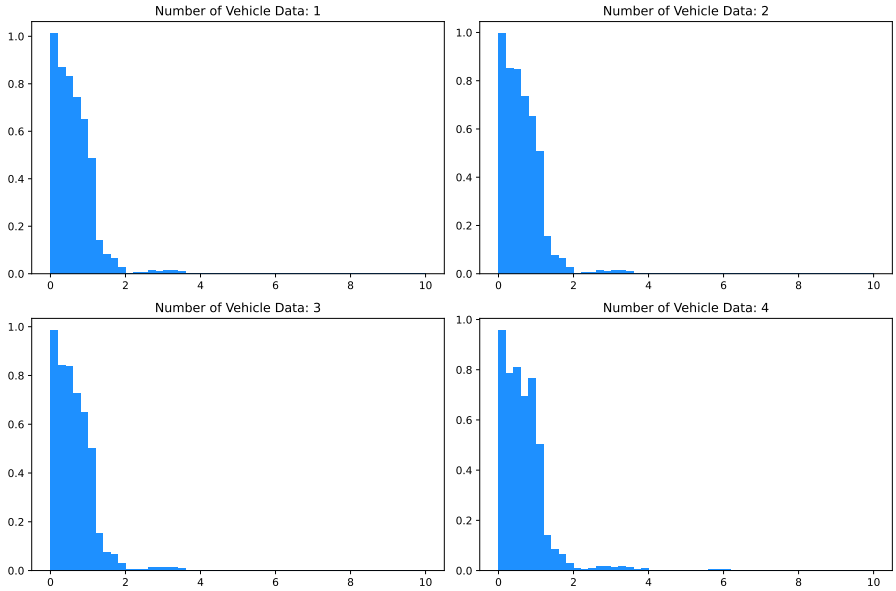

Figure 6.8: Histogram: Distributions of accumulated positional errors with more individual drives' data in Region 02 input to the model

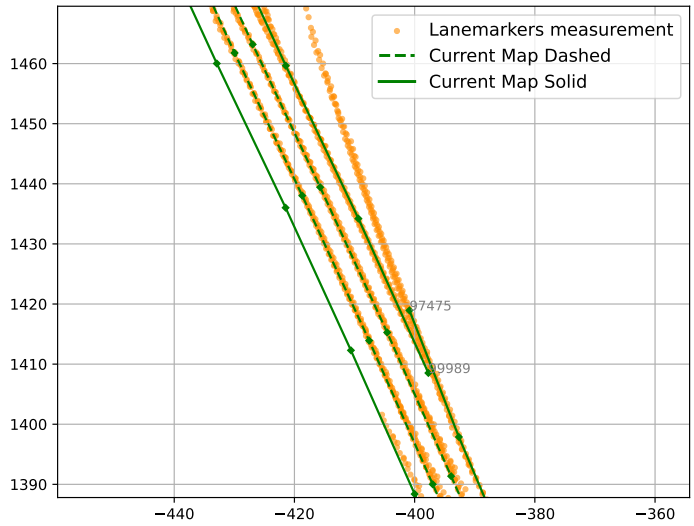

Figure 6.9: Current map and measurement: the lanemarker polyline of current map can be supplemented from tail.

Figures 6.9 and 6.10 show the map's refinement at the tail end of the lane markers. The before-and-after comparison distinctly illustrates how the integration of new vehicle data can enhance the accuracy of the map, filling in gaps and refining the tail section of the lane markers.

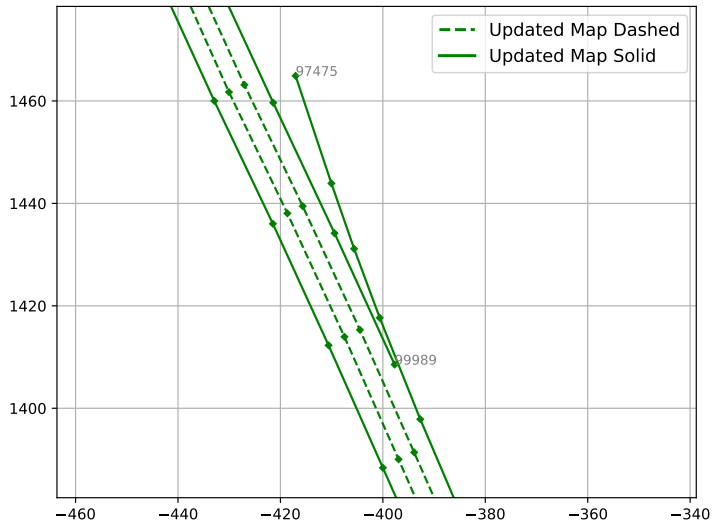

Figure 6.10: Updated map: the lanemarker polyline is supplemented from tail using measurements.

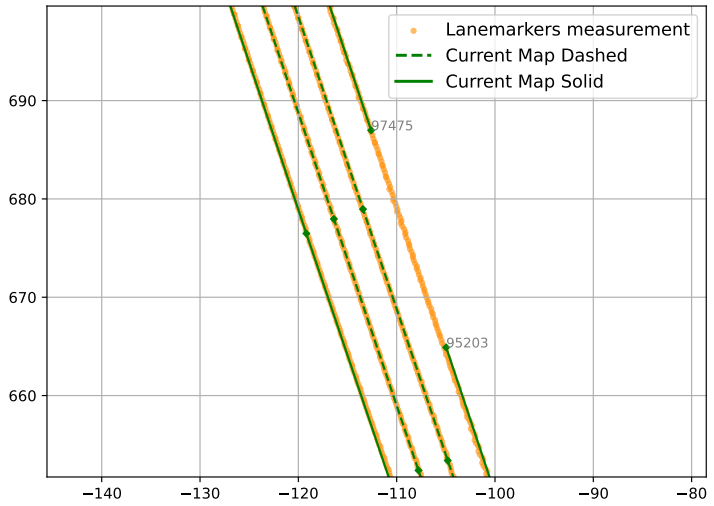

Figure 6.11: Current map and measurement: the lanemarker polyline of current map can be supplemented from head.

Similarly, Figures 6.11 and 6.12 focus on the head section of the lane markers. The before-and-after contrast here also underscores the improvements made, showcasing the ability of new measurements to provide a more comprehensive representation at the beginning of the lanes.

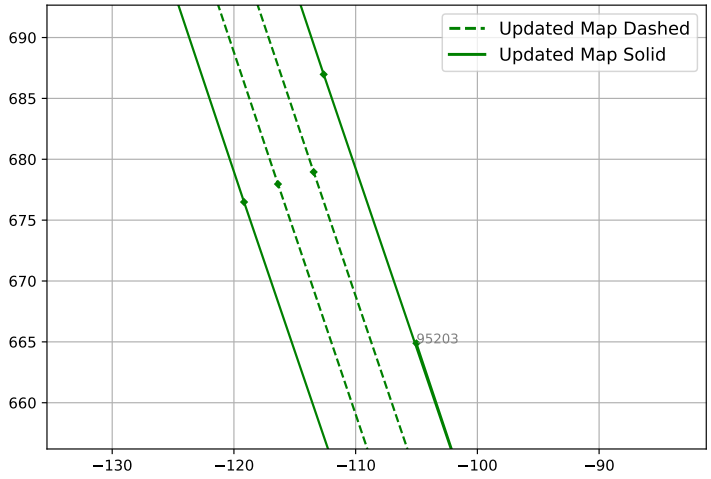

Figure 6.12: Updated map: the lanemarker polyline is supplemented from head using measurements.

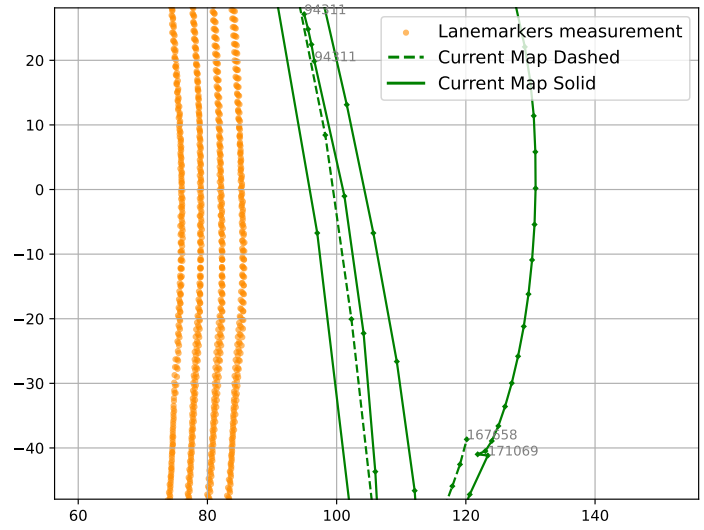

Figure 6.13: Current map and measurement: the lanemarker polyline of current map can be supplemented from tail.

Finally, Figures 6.13 and 6.14 highlight the capability of the map generation system to create entirely new lane markers when the data suggests it. This is especially significant as it showcases the system's adaptability and responsiveness to fresh data inputs.

The visual representations in the figures underscore the efficacy of integrating new

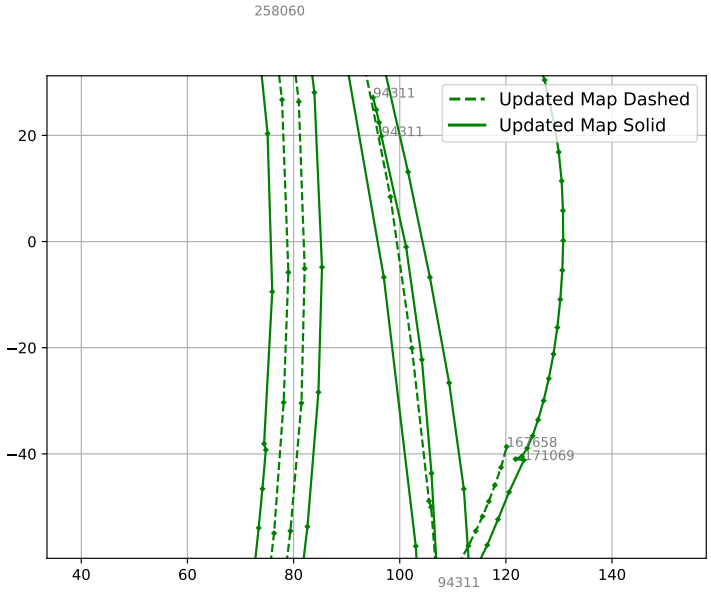

Figure 6.14: Updated map: the lanemarker polyline is supplemented from tail using measurements.

vehicle data into the map generation process. Refinements at both the tail and head sections of lane markers clearly demonstrate the system's ability to enhance granularity and accuracy. Most notably, the introduction of entirely new lane markers epitomizes the system's adaptability, attesting to its capability to continuously evolve and optimize in response to updated data inputs.

Comprehensive analysis reveals that the precision of this model in map generation is significant. Most positional errors remain within a range of 1 meter, and in specific areas like Region 02, its accuracy even approaches half a meter. Moreover, regardless of the fundamental quality of the data, the model exhibits robustness, ensuring overall performance stability. It's crucial to emphasize that the system can effectively integrate data from different vehicles, addressing gaps in the existing map and subsequently constructing a comprehensive and detailed map. Importantly, these results were achieved despite limitations in testing resources, underscoring the efficacy and potential value of the model.

6.2 Evaluation for Learning Based Approach

6.2.1 Evaluation Method

In this section, we focus on the performance of the learning-based approach, namely the correctness of the road element's geometry, location, type and whether it has correctly connected to road elements in the previous frame. We use Chamfer Distance as the evaluation metric when calculating geometry and location.

The Chamfer Distance serves as a vital metric to gauge the dissimilarity between

two point sets A and B, and is widely employed in computer vision and geometric modeling for tasks such as shape matching and point cloud registration. Formally defined, the Chamfer Distance is computed by

$$CD(A, B) = \sum_{\mathbf{a} \in A} \min_{\mathbf{b} \in B} \|\mathbf{a} - \mathbf{b}\|_2^2 \tag{6.1}$$

This distance is computed by determining the squared Euclidean distance from every point in one set to its nearest counterpart in the other set. However, as for the need of having comparable results to the model-based approach, we would replace the squared Euclidean distance with Euclidean distance. We also use Chamfer Distance to align elements, for every predicted road element, we use the distance to find the according element in the ground-truth set according to closest Chamfer Distance. Marker type (solid line, dashed line) between aligned elements will also be compared, we will calculate the true positive (TP) and False Positive (FP) of the predictions, precision score will be calculated by Precision $= \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$.

It is worth noting that, comparing to the optimization-based approach, the network takes no prior information (SD Map information, traffic sign, etc.). It is only taking the crowd-sourced data, which has been made into images, and interprets them to vectorized road elements geometries. Sometimes, the crowdsourced data we use for training significantly deviates from the HD map that serves as our reference. For instance, we may observe differences in road width or find that the locations of elements on the HD map have an overall bias compared to the crowdsourced data. This discrepancy poses a challenge: even when the network's output appears accurate to the human eye, it may perform poorly according to our evaluation metrics. To mitigate this issue, we also evaluate the network's performance using a simulated dataset. The advantage of using simulated data is that it allows us to control for the inconsistencies and biases present in crowdsourced data. By doing so, we obtain a more reliable and comprehensive assessment of the network's capabilities, supplementing the evaluations based on real-world data.

When we are evaluating, the function of connecting elements in the neighboring frames could also affect results, error accumulated from previous frames, though can be mitigated by the model, still somehow affecting the output. Therefore, we would evaluate the same data with the function turned on and off separately.

The crowdsourced data is derived from the travels of five real drivers around Gothenburg, spanning a total distance of 25 kilometers. The neural network will perform inference on this data, and evaluation metrics will be computed by comparing the network's output with the reference map. Additionally, our simulated evaluation set, comprising approximately 15 kilometers, will be fed into the network for inference. The resulting output will be contrasted with the ground truth to assess the numerical performance of the neural network. We will conduct two tests on every datasets, with and without connecting the previous frame elements, to determine the impact of this function.

6.2.2 Evaluation Results

We assessed the performance of the neural network on both real-world and simulated datasets. To investigate the influence of connecting elements, we conducted experiments with this function both enabled and disabled. A box plot showing the error distribution of different methods and dataset is shown in 6.16. Additionally, histograms depicting the distribution of errors across different cases have been plotted and can be found in Figure 6.15.

By examining the results in the boxplot 6.16, which illustrates the performance of the neural network tested under different scenarios, we can observe that although the HD map and input data are not perfectly aligned in real-world scenarios, the model still strives to output the map's geometric shape according to the given input data. Surprisingly, the error in real-world data does not differ significantly from that of simulated data. However, the precision is notably lower in real-world instances, a discrepancy that may be attributed to a misalignment between the output and the high-definition map. Performance on simulated data is superior to real-world data, with accurate predictions made for the types of various elements. Additionally, our results indicate that disabling the connection functionality between frames can prevent error accumulation, thus enhancing the model's performance. Conversely, enabling this feature does not significantly impair the model's performance, demonstrating a balanced trade-off between functionality and accuracy.

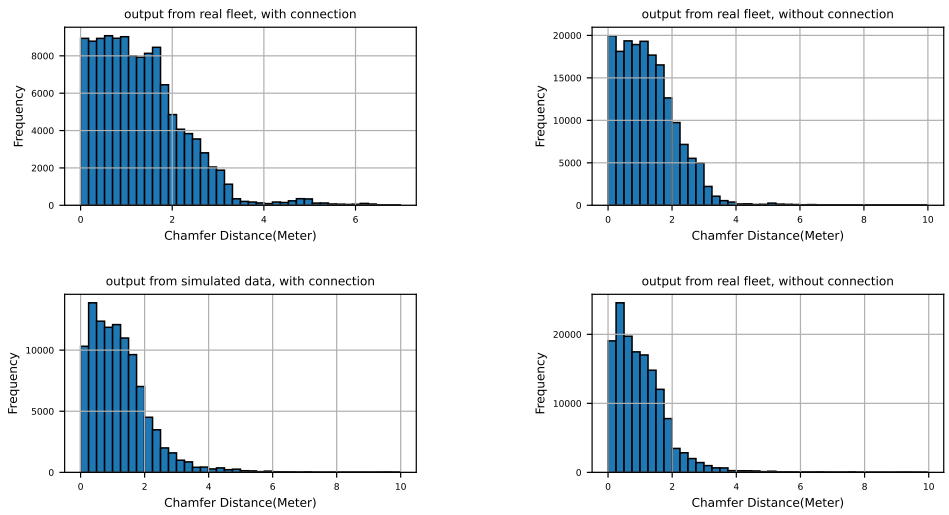

Figure 6.15: Histogram of Euclidean Distance from different evaluation case.

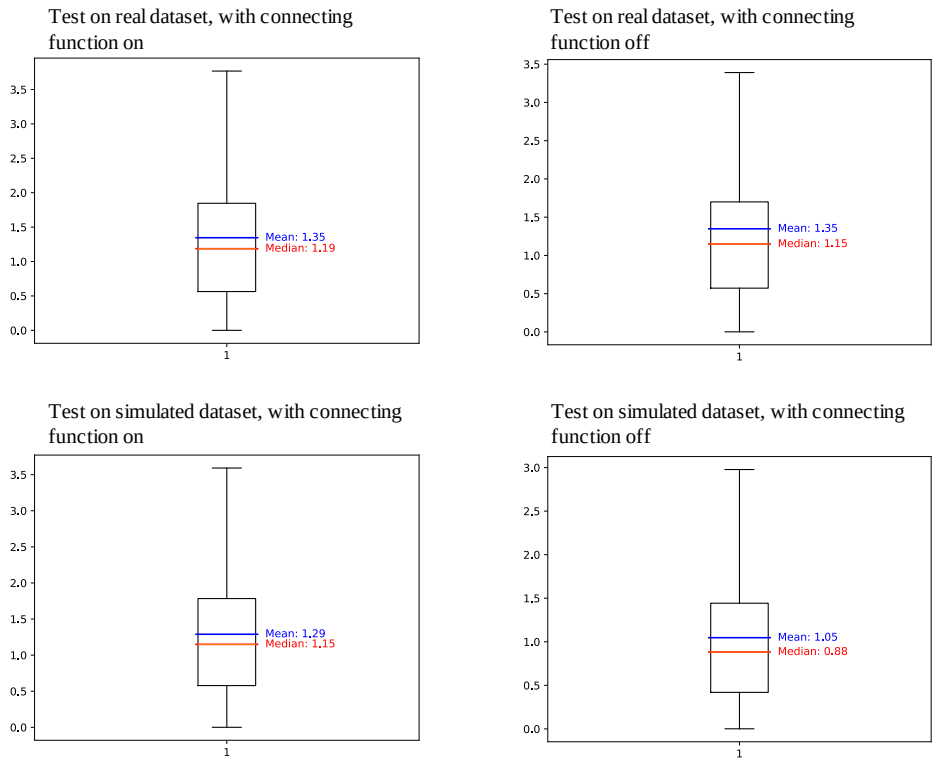

Figure 6.16: Box plot of error distributions from different evaluation case.

7

Conclusion & Discussion

This final chapter offers a comprehensive review of the two principal methodologies explored in this research for HD map construction: optimization-based and learning-based approaches. The following sections will delve into each method's relative advantages and challenges based on the data and findings from this study. Additionally, the respective applicability of these methods in different scenarios will be discussed, along with the potential for integrating both approaches to enhance map accuracy and efficiency.

7.1 Optimization-based Method: An Overview

The methodology for High-Definition (HD) map construction is grounded in nonlinear optimization theory and the solving of nonlinear optimization problems. This approach has proven to be consistently reliable across extensive multi-vehicle datasets. It maintains an error margin predominantly within 1 meter and even narrows it down to approximately 0.5 meters in specific regions, thus highlighting the model's robustness against variations in data quality among different vehicles.

Despite its promising attributes, the method has its challenges. First, the computational demand can be high, especially when dealing with large datasets. Second, within the scope of our project, the precision of the map generated by the initial vehicle significantly impacts the model's overall accuracy. Moreover, the model is less effective when confronted with significant discrepancies between local maps from different vehicles due to sensor instability. This shortfall is partly attributable to the model's simplistic lane-line update strategy, which is predicated on the assumption of reasonable alignment between the data from preceding vehicles.

For future enhancements, more emphasis should be put on high-quality data to improve overall accuracy. The model could also benefit from integrating additional information like lane occupation to refine the map-building process further. Development of a more adaptive update strategy can help the model better cope with various scenarios and reconcile discrepancies between data from different vehicles. Additionally, assigning weighted contributions to individual vehicles based on factors like data quality and freshness could provide an additional layer of refinement to the optimization process, thereby enhancing the model's reliability.

7.2 Learning-based Method: An Overview

The deep learning model demonstrates the capability to interpret road geometric parameters based on input images and connects road elements across adjacent frames to create a vectorized map. It exhibits strong performance on a simulated dataset. However, there are noticeable discrepancies when inferring on real-world data. If equipped with higher-quality crowd-sourced datasets (including real vehicle data and aligned high-definition map data), the training could be more effective. The model's main strengths include its ability to decipher road geometric parameters from input images, the creation of a vectorized map by connecting road elements derived from neighboring frames, and its promising performance on simulated data.

Several drawbacks hinder the model's performance. It cannot correct crowd-sourced data based on prior knowledge, and despite best efforts, there remains a discernible gap between simulated and real-world data. Additionally, the model lacks the capacity to determine the confidence level of each road element output, leading to potential redundancy in predictions.

Potential areas for improvement in the learning-based method encompass several key aspects. These include the procurement of a refined dataset to reconcile the gap between the simulated and real-world data, thereby enhancing model authenticity. Additionally, the development of mechanisms to ascribe varied semantic values to the maps input into the model could be explored. For instance, introducing a special embedding for high-definition maps could enable the model to discern their unique characteristics. Furthermore, incorporating the capacity to determine confidence scores for each road element could allow for the elimination of superfluous predictions. These enhancements collectively hold the promise of increasing the model's overall efficiency and reliability.

7.3 Applicability in Different Scenarios

The optimization-based method demonstrates stable performance, especially in structured environments like highways or roads with well-defined patterns. This stability stems from the model's focus on precision and its adeptness at aligning map elements. Importantly, our evaluation results suggest that the model is robust to variations in data quality, demonstrating a notable degree of fault tolerance. Even when faced with low-quality or inconsistent data, the model maintains its performance. This makes it a reliable tool for initial map creation, particularly when the maps require relatively high precision. The method's robustness is particularly beneficial when integrating data from a variety of vehicles, as it minimizes the impact of poor-quality or inconsistent data on the overall map accuracy.

On the other hand, the learning-based method offers a significant potential ceiling. With larger models, higher quality data, and extended training durations, the performance and robustness of the model can be substantially enhanced. This method remains relatively stable in big data scenarios, even when the data contains noise or discrepancies. Trained learning models possess the ability to process imperfect

data and extract critical information from it. Given its inherent characteristics, the learning-based method is apt for scenarios that demand frequent updates or real-time feedback. It can swiftly adapt to fresh data inputs, facilitating real-time map adjustments.

7.4 Integration Potential

Based on the observed advantages of the optimization-based method in earlier phases of map construction, one feasible strategy could be a two-phase integration approach. In the initial phase, the optimization-based method may be favoured for creating a foundational map. It can employ nonlinear optimization theories and iterative solutions to achieve an early reasonable accuracy level. Once the foundational map is in place, the system can transition to the second phase, employing a learning-based method. This second phase would focus on leveraging machine learning algorithms to adaptively refine map features, update road conditions, or even identify and correct anomalies. This sequential approach aims to capitalize on the quick accuracy gains provided by optimization, followed by the adaptive, self-learning features of the learning-based method.

A more complex, concurrent approach could be considered in environments with abundant data. Both optimization and learning-based methods would operate in parallel, processing incoming data in real time. Given that each method may have unique strengths and weaknesses, a weighted merging algorithm could synthesize the results. For instance, optimization-based outputs could be given more weight in areas requiring high positional accuracy, while learning-based outputs could be prioritized for rapid feature updates or in highly variable conditions. While theoretically appealing, this parallel strategy would require additional research to validate its effectiveness.

In summary, the potential for methodological integration provides intriguing avenues for further study. The approaches suggested here, either sequential or concurrent, would require rigorous validation to substantiate their utility in the future.

Bibliography

[1] *Highly Automated Driving (HAD): The future of driverless road freight*. Here Technologies, 2022. [Online]. Available: https://www.here.com/solutions/automated-driving/the-future-of-driverless-road-freight-ebook.

[2] Z. Bao, S. Hossain, H. Lang, and X. Lin, "A review of high-definition map creation methods for autonomous driving," *Engineering Applications of Artificial Intelligence*, vol. 122, p. 106 125, 2023, ISSN: 0952-1976. DOI: https://doi.org/10.1016/j.engappai.2023.106125. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0952197623003093.

[3] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós, "Orb-slam: A versatile and accurate monocular slam system," *IEEE Transactions on Robotics*, 2015.

[4] R. Mur-Artal and J. D. Tardós, "Orb-slam2: An open-source slam system for monocular, stereo and rgb-d cameras," *IEEE Transactions on Robotics*, 2016.

[5] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. M. Montiel, and J. D. Tardós, "Orb-slam3: An accurate open-source library for visual, visual-inertial and multi-map slam," *arXiv: Robotics*, 2020.

[6] S. Cao, X. Lu, and S. Shen, "Gvins: Tightly coupled gnss-visual-inertial fusion for smooth and consistent state estimation.," *arXiv: Robotics*, 2021.

[7] G. Máttyus, W. Luo, and R. Urtasun, "Deeproadmapper: Extracting road topology from aerial images," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 3438–3446.

[8] S. M. Azimi, P. Fischer, M. Körner, and P. Reinartz, "Aerial lanenet: Lane-marking semantic segmentation in aerial imagery using wavelet-enhanced cost-sensitive symmetric fully convolutional neural networks," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 57, no. 5, pp. 2920–2938, 2018.

[9] P. Fischer, S. M. Azimi, R. Roschlaub, and T. KrauSS, "Towards hd maps from aerial imagery: Robust lane marking segmentation using country-scale imagery," *ISPRS International Journal of Geo-Information*, vol. 7, no. 12, p. 458, 2018.

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

[11] H. Gao, Y. Yuan, and X. Zheng, "Remote sensing road extraction by refining road topology," in *Proceedings of the 6th China High Resolution Earth Observation Conference (CHREOC 2019)*, Springer, 2020, pp. 187–197.

[12] Y. Wei, K. Zhang, and S. Ji, "Simultaneous road surface and centerline extraction from large-scale remote sensing images using cnn-based segmentation

and tracing," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 58, no. 12, pp. 8919–8931, 2020.

[13] A. Batra, S. Singh, G. Pang, S. Basu, C. Jawahar, and M. Paluri, "Improved road connectivity by joint learning of orientation and segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 10 385–10 393.

[14] T. Sämann, K. Amende, S. Milz, C. Witt, M. Simon, and J. Petzold, "Efficient semantic segmentation for visual birds-eye view interpretation," in *Intelligent Autonomous Systems 15: Proceedings of the 15th International Conference IAS-15*, Springer, 2019, pp. 679–688.

[15] Q. Li, Y. Wang, Y. Wang, and H. Zhao, "Hdmapnet: An online hd map construction and evaluation framework," in *2022 International Conference on Robotics and Automation (ICRA)*, IEEE, 2022, pp. 4628–4634.

[16] Y. Liu, T. Yuan, Y. Wang, Y. Wang, and H. Zhao, "Vectormapnet: End-to-end vectorized hd map learning," in *International Conference on Machine Learning*, PMLR, 2023, pp. 22 352–22 369.

[17] B. Liao, S. Chen, X. Wang, *et al.*, "Maptr: Structured modeling and learning for online vectorized hd map construction," *arXiv preprint arXiv:2208.14437*, 2022.

[18] J. Shin, F. Rameau, H. Jeong, and D. Kum, "Instagram: Instance-level graph modeling for vectorized hd map learning," *arXiv preprint arXiv:2301.04470*, 2023.

[19] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A survey of autonomous driving: Common practices and emerging technologies," *IEEE Access*, vol. 8, pp. 58 443–58 469, 2020. DOI: 10.1109/ACCESS.2020.2983149.

[20] Lantmäteriet. "SWEREF 99," Lantmäteriet. (N/A), [Online]. Available: https://www.lantmateriet.se/en/geodata/gps-geodesi-och-swepos/reference-systems/three-dimensional-systems/SWEREF-99/ (visited on 05/28/2023).

[21] C. Cadena, L. Carlone, H. Carrillo, *et al.*, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1309–1332, 2016. DOI: 10.1109/TRO.2016.2624754.

[22] G. Grisetti, R. Kummerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based slam," en-US, *IEEE Intelligent Transportation Systems Magazine*, pp. 31–43, Jan. 2011. DOI: 10.1109/mits.2010.939925. [Online]. Available: http://dx.doi.org/10.1109/mits.2010.939925.

[23] X. Gao, T. Zhang, Y. Liu, and Q. Yan, *14 Lectures on Visual SLAM: From Theory to Practice*. Publishing House of Electronics Industry, 2017.

[24] F. Dellaert, M. Kaess, *et al.*, "Factor graphs for robot perception," *Foundations and Trendső in Robotics*, vol. 6, no. 1-2, pp. 1–139, 2017.

[25] N. Andréasson, A. Evgrafov, E. Gustavsson, *et al.*, *An Introduction to Continuous Optimization*. Studentlitteratur, 2016.

[26] S. Agarwal, K. Mierle, and T. C. S. Team, *Ceres Solver*, version 2.1, Mar. 2022. [Online]. Available: https://github.com/ceres-solver/ceres-solver.

[27] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "G 2 o: A general framework for graph optimization," in *2011 IEEE International Conference on Robotics and Automation*, IEEE, 2011, pp. 3607–3613.

[28] N. Chebrolu, T. Läbe, O. Vysotska, J. Behley, and C. Stachniss, *Adaptive robust kernels for non-linear least squares problems*, 2021. arXiv: 2004.14938 [cs.RO].

[29] K. MacTavish and T. D. Barfoot, "At all costs: A comparison of robust cost functions for camera correspondence outliers," in *2015 12th Conference on Computer and Robot Vision*, 2015, pp. 62–69. DOI: 10.1109/CRV.2015.52.

[30] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[31] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[32] A. Dosovitskiy, L. Beyer, A. Kolesnikov, *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[33] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *European conference on computer vision*, Springer, 2020, pp. 213–229.

[34] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[35] J. Munkres, "Algorithms for the assignment and transportation problems," *Journal of the Society for Industrial and Applied Mathematics*, vol. 5, no. 1, pp. 32–38, 1957.

[36] R. E. Burkard, M. Dell'Amico, and S. Martello, *Assignment problems*. SIAM, 2012.

[37] W. Cook, L. Lovász, and A. Schrijver, *Combinatorial optimization*. John Wiley & Sons, 2004.

[38] A. Schrijver, *Combinatorial Optimization: Polyhedra and Efficiency*. Springer Science & Business Media, 2003, vol. 24.

[39] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, 1982.

[40] OpenStreetMap contributors, *Planet dump retrieved from https://planet.osm.org*, https://www.openstreetmap.org, 2017.

[41] B. De Brabandere, D. Neven, and L. Van Gool, "Semantic instance segmentation with a discriminative loss function," *arXiv preprint arXiv:1708.02551*, 2017.

[42] W. Van Gansbeke, B. De Brabandere, D. Neven, M. Proesmans, and L. Van Gool, "End-to-end lane detection through differentiable least-squares fitting," in *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, 2019, pp. 0–0.

[43] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese, "Generalized intersection over union," Jun. 2019.

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

CHALMERS
UNIVERSITY OF TECHNOLOGY