



Master's Thesis in Robotics, Cognition, Intelligence

# Automatic Calibration of Infrastructure LiDAR and Camera Sensors

Automatische Kalibrierung von Infrastruktur-LiDAR-  
und Kamerasensoren

<b>Supervisor</b>	Prof. Dr.-Ing. habil. Alois C. Knoll
<b>Advisor</b>	Walter Zimmer, M.Sc.
<b>Author</b>	Bohan Wang
<b>Date</b>	January 15, 2023 in Garching



# Disclaimer

I confirm that this Master's Thesis is my own work and I have documented all sources and material used.

Garching, January 15, 2023

---

(Bohan Wang)

## Abstract

In the field of autonomous driving, LiDAR plays an increasingly important role because of its ability to preserve rich scene information (such as 3D coordinates, reflectivity, and depth information). In order to obtain an accurate perception of the scene, the extrinsic calibration of LiDAR and camera is particularly important. As part of the *AUTOTech.agil* project, the main purpose of this work is to perform the calibration of LiDAR sensors and cameras. In this thesis, we divide the task into two parts. One is the calibration of the LiDAR and the HD map (in OpenDRIVE format). Once the position of the LiDAR in the HD map is obtained, the rough extrinsic of the LiDAR and the camera can be simply obtained using the known coordinates of the camera in the HD map. This will assist in the initial extrinsic settings for the automatic calibration between the LiDAR and the camera. In the second part, the contribution is mainly to improve the existing voxel-based automatic LiDAR-camera calibration method, since the original method is mainly used for the calibration of indoor scenes. Therefore, the original method has a series of problems when used in the outdoor scene of the *AUTOTech.agil* project. In order to analyze the source of the problem and improve it, the parameters of the model were optimized. In order to solve the interference of the camera's distant background on the calibration, monocular depth estimation was applied to automatically filter the background of the camera image. In addition, adaptive voxelization is introduced and multi-scene optimization is used to improve the generalization ability. Finally, the generalization ability and progress of the new model is tested and analyzed in a variety of different scenarios and sensor pairs, and the results are presented both quantitatively and qualitatively.

Keywords: Automatic calibration, LiDAR, camera, target-less calibration, HD map, adaptive voxelization, monocular depth estimation

## Zusammenfassung

Im Bereich des autonomen Fahrens spielt LiDAR aufgrund seiner Fähigkeit, reichhaltige Szeneninformationen (wie 3D-Koordinaten, Reflexionsvermögen und Tiefeninformationen) zu bewahren, eine immer wichtigere Rolle. Um eine genaue Wahrnehmung der Szene zu erhalten, ist die extrinsische Kalibrierung von LiDAR und Kamera besonders wichtig. Im Rahmen des AUTOTech.agil Projekts besteht der Kern dieser Arbeit darin, die Kalibrierung zwischen LiDAR und Kamerasensoren durchzuführen. In dieser Arbeit wird die Aufgabe in zwei Teile aufgeteilt. Ein Teil davon ist die Kalibrierung des LiDARs mit der HD-Karte (im OpenDRIVE Format). Sobald die Position des LiDARs in der HD-Karte bestimmt wurde, kann die grobe Extrinsic des LiDARs und der Kamera unter Verwendung der bekannten Koordinaten der Kamera in der HD-Karte berechnet werden. Dies hilft bei den initialen extrinsischen Einstellungen für die automatische Kalibrierung zwischen dem LiDAR und der Kamera. Im zweiten Teil besteht mein Beitrag hauptsächlich darin, das vorhandene voxelbasierte automatische LiDAR-Kamerakalibrierungsverfahren zu verbessern, da die ursprüngliche Methode hauptsächlich für die Kalibrierung von Sensoren in Innenräumen verwendet wird. Daher hat die ursprüngliche Methode eine Reihe von Problemen, wenn sie auf dem Testfeld des AUTOTech.agil Projekts im Freien zum Einsatz kommt. Um die Ursache des Problems zu analysieren und zu verbessern, wurden die Parameter des Modells optimiert. Um die Störung des entfernten Hintergrunds der Kamera bei der Kalibrierung herauszufiltern, wurde eine monokulare Tiefenschätzung umgesetzt, um den Hintergrund des Bildes automatisch zu entfernen. Zusätzlich wird eine adaptive Voxelisierung eingeführt und eine Mehrszenenoptimierung verwendet, um die Generalisierungsfähigkeit der Kalibrierung zu verbessern. Abschließend werden Generalisierungsfähigkeit und Fortschritt des neuen Modells in einer Vielzahl unterschiedlicher Szenarien und Sensorpaare evaluiert, analysiert und die Ergebnisse sowohl quantitativ als auch qualitativ dargestellt.

Schlüsselwörter: automatische Kalibrierung (ohne die Verwendung von Schachbrettmustern), LiDAR, Kamera, HD-Karte, adaptive Voxelisierung, monokulare Tiefenschätzung



# Abbreviations

**TOF** Time of Flight  
**LiDAR** Light Detection and Ranging  
**PC** Point Cloud  
**RGBD** Red Green Blue Depth  
**IMU** Inertial Measurement Unit  
**FoV** Field-of-View  
**HD map** High Definition map  
**SVD** Singular Value Decomposition  
**PnP** Perspective-n-Point  
**ICP** Iterative Closest Point  
**ROS** Robot Operating System  
**BA** Bundle Adjustment  
**CRF** Conditional Random Field  
**SfM** Structure from Motion  
**RANSAC** Random Sample Consensus  
**PBA** Plane-constrained Bundle Adjustment  
**MLE** Maximal Likelihood Estimation  
**GPS** Global Positioning System



# Contents

<b>Abbreviations</b>	<b>vii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Problem Statement . . . . .	4
1.3 Challenges . . . . .	4
1.4 Contribution . . . . .	5
1.5 Thesis Outline . . . . .	6
<b>2 Theoretical Background</b>	<b>7</b>
2.1 LiDAR Sensor and Point Cloud . . . . .	7
2.1.1 Introduction of LiDAR Sensor . . . . .	7
2.1.2 Properties of Point Cloud Data . . . . .	9
2.2 Point Cloud Registration . . . . .	11
2.3 LiDAR-Camera Calibration . . . . .	13
2.4 Search algorithm . . . . .	14
2.4.1 KNN . . . . .	15
2.4.2 KDTree . . . . .	15
2.4.3 Octree . . . . .	16
2.5 Evaluation Metrics . . . . .	17
<b>3 Related Work</b>	<b>19</b>
3.1 LiDAR-Camera Calibration Datasets . . . . .	19
3.1.1 Indoor Datasets . . . . .	19
3.1.2 Outdoor Datasets . . . . .	20
3.1.3 Outdoor infrastructure Dataset . . . . .	22
3.2 LiDAR-Camera Calibration Methods . . . . .	24
3.2.1 Target-based Methods . . . . .	24
3.2.2 Targetless Methods . . . . .	26
3.3 Point Cloud Upsampling . . . . .	31
3.3.1 Optimization based method . . . . .	31
3.3.2 Deep Learning-Based Methods . . . . .	33
<b>4 Manual Calibration</b>	<b>35</b>
4.1 LiDAR-HD Map Manual Calibration . . . . .	35
4.1.1 HD Map Structure . . . . .	35
4.1.2 OpenDRIVE . . . . .	37
4.1.3 Solution . . . . .	41
4.1.4 Results . . . . .	44
4.1.5 Short discussion . . . . .	48
4.2 LiDAR-Camera Manual Calibration . . . . .	49

4.2.1	Manual Calibration Pipeline . . . . .	49
4.2.2	Results . . . . .	49
4.2.3	Short discussion . . . . .	54
<b>5</b>	<b>Automatic Calibration</b>	<b>55</b>
5.1	Voxel-Based Automatic LiDAR-to-Camera Calibration . . . . .	55
5.1.1	Voxel-based Edge Extraction . . . . .	56
5.1.2	LiDAR-Camera Edge Pair Matching . . . . .	56
5.1.3	Calibration Formulation and Optimization . . . . .	57
5.2	Automatic Calibration Pipeline . . . . .	58
5.2.1	Single Frame Calibration . . . . .	58
5.2.2	Multiple Frame Calibration . . . . .	59
5.2.3	Offline Calibration . . . . .	59
5.3	Hyperparameter . . . . .	60
5.3.1	Key Principle of Calibration . . . . .	60
5.3.2	Canny Edge Extraction . . . . .	60
5.3.3	Voxel Cutting . . . . .	61
5.3.4	Plane Fitting . . . . .	62
5.3.5	Edge Cutting . . . . .	63
5.4	Background Filtering and Cropping . . . . .	64
5.4.1	Problem Statement . . . . .	65
5.4.2	Automatic Background Removal . . . . .	65
5.4.3	Template based Background Cropping . . . . .	67
5.4.4	Template based Background Cropping Results . . . . .	67
5.5	Dealing with Shadows . . . . .	69
5.5.1	Effect of Shadows . . . . .	69
5.5.2	Shadow filtering . . . . .	70
5.6	LiDAR preprocessing . . . . .	71
5.6.1	Dimension Reduction and Cropping . . . . .	71
5.6.2	Scattering . . . . .	72
5.6.3	Outlier Removal . . . . .	72
5.6.4	Point Upsampling . . . . .	74
5.7	Adaptive Voxelization . . . . .	74
5.7.1	Adaptive Voxelization . . . . .	75
5.7.2	Result of Adaptive Voxeliastion . . . . .	75
<b>6</b>	<b>Results</b>	<b>77</b>
6.1	Result South LiDAR-South2 Camera . . . . .	77
6.1.1	Scene 1 . . . . .	78
6.1.2	Scene 2 . . . . .	80
6.1.3	Scene 3 . . . . .	82
6.2	Result North LiDAR-South1 Camera . . . . .	84
6.2.1	Scene 1 . . . . .	85
6.2.2	Scene 2: Initial Extrinsic with Offset and Heavy Traffic . . . . .	86
<b>7</b>	<b>Evaluation and Discussion</b>	<b>89</b>
7.1	Limit Test - South LiDAR and South2 Camera . . . . .	89
7.1.1	Rotation Limit Test . . . . .	90
7.1.2	Translation Limit Test . . . . .	96
7.1.3	Short Summary . . . . .	100
7.2	Limit Test - North LiDAR and South1 Camera . . . . .	100

7.2.1	Rotation Limit Test . . . . .	101
7.2.2	Translation Limit Test . . . . .	105
7.2.3	Short Summary . . . . .	109
7.3	Runtime Measurements . . . . .	109
<b>8</b>	<b>Conclusion</b>	<b>111</b>
<b>9</b>	<b>Future Work</b>	<b>113</b>



# Chapter 1

## Introduction

Thanks to the development of artificial intelligence and sensor technology, autonomous driving technology is becoming a reality and showing strong competitiveness. The application of autonomous driving technology makes road traffic more comfortable, efficient and safe, and these advantages make autonomous driving technology one of the most important counterpoints for the upgrading of the automotive industry.

In this chapter, the motivation for automatic calibration will be presented within the AUTOTech.agil project in Section 1.1 In 1.2 the role of this work will be stated. In Section 1.3 the main challenges that were encountered will be listed. In Section 1.4 our main contributions will be stated. Finally, the structure of this thesis will be described briefly in Section 1.5.

### 1.1 Motivation

At present, autonomous driving technology is mainly used in highway scenarios. On the one hand, the application scenarios of highways have a greater demand for autonomous driving: long-distance driving is boring and tiring. The introduction of autonomous driving can significantly improve the comfort of long-distance road trips and significantly reduce accidents caused by drowsy driving. On the other hand, the existing autonomous driving technology is not enough to deal with the complex and changing urban environments. The highway has the advantages of wide field of vision, fewer types of obstacles that are easy to identify, fewer emergencies and a stable environment. This makes autonomous driving on the highway easier to achieve.

One of the key problems of autonomous driving is the perception of the environment, which relies on the application and fusion of many different sensors. Although current self-driving vehicles are equipped with various sensors including LiDAR, RGB camera, RGBD camera, radar, IMU, etc.. However, they are limited by the occlusion of the field of view (only the FoV from the ego vehicle) and the incomplete environment information they obtain. Complex traffic scenarios, such as intersections, roundabouts or exits on motorways are the biggest challenges for autonomous driving. The restricted field of vision of the vehicle is not sufficient to evaluate these situations. Therefore, within the context of *AUTOTech.agil*, Therefore, within the context of *AUTOTech.agil*, it is also necessary to equip the surrounding infrastructure with sensors.

With the project *AUTOTech.agil*, the Technical University of Munich has set the objective for the further development of autonomous driving in mixed traffic conditions. Autonomous driving can optimize the flow of traffic and reduce the number of accidents. In addition, the data collected by the technology of autonomous driving, offer a lot of opportunities for

commercial and scientific use.

In the project *AUTOTech.agil*, sensors installed on gantry bridges can provide additional information about the current traffic situation, which contains the status of all traffic participants with their position, velocity, type, size, etc.. Those data will be transported to vehicles through a 5G network. Due to the higher mounting position of sensors on the gantry bridge (about 7 m above the ground), it can greatly improve the environmental perception range of vehicles.

As Figure 1.1 shows, gantries with different sensors collect the real-time data of the traffic on the highway.



**Figure 1.1:** Gantry with sensors, the gantry contains multiple cameras ,LiDARs and Radars

The predecessor project *Providentia++* was subdivided in two phases:

In the first phase of the project (*Providentia 1*), high-resolution cameras and radar systems were mounted on two overhead gantry bridges on a test section of the A9 highway (no. 2 and 3 in Figure 1.2). The sensor data was transmitted wirelessly via 5G, 2D object detection was used to identify vehicle classes, and the radar and camera data was then fused – in other words, a digital twin was created. An autonomous vehicle (*Fortuna*) was used for example to incorporate the information from the digital twin to independently change lanes on the highway or to slow down to avoid traffic jams or accidents.

By the end of 2019, the infrastructure on the A9 highway had been set up and the digital twin of the highway traffic had been developed. In the second phase (*Providentia++*) the technology was fine-tuned, the infrastructure was expanded into urban space, and value-added services were developed. New transmitter and sensor towers were erected on the edge of a feeder road into a residential area. Intersections, roundabouts, bus stops, train stations, and park-and-ride parking lots can be surveyed anonymously by the sensor technology.

LiDAR systems, which generally cover a larger angle than radars, were used in the project for the first time. In addition, value-added services were created for drivers, highway operators, vehicle manufacturers, and academia. The follow-up project *Providentia++* was launched in early 2020 and lasted two and a half years. The *Providentia++* project was funded with 9.16 million euros from the Federal Ministry of Transport and Digital Infrastructure (BMVI) and industry partners. The industry partners are the Technical University of Munich as consortium leader, Cognition Factory, Elektrobit Automotive, Valeo, fortiss, and Intel Deutschland. Addi-



**Figure 1.2:** Overview of the whole project, it contains multiple sections in the A9 highway, here are 6 different sensor stations in the A9 highway.

tional associated partners are Huawei Technologies Deutschland, IBM Deutschland, brighter AI, and 3D Mapping Solutions [81].

The *AUTOTech.agil* researchers also have new services, application scenarios, and business models in mind. Drivers, highway operators, scientists, and car manufacturers all stand to benefit equally from these services in the future. For example:

- **Warning of risk of collision and looking ahead:** *AUTOTech.agil* extends the capability of in-vehicle sensors through external information from sensor stations equipped with cameras, radars, and, in the future, LiDARs [66]. This will enable drivers to react to traffic situations earlier than before.
- **Improving traffic flow:** Connected vehicles are able to “understand” the current traffic situation and anticipate how traffic will develop. This connectivity can lead to autonomous vehicles on the highway changing lanes earlier when there are indications of slower traffic, and beginning to slow down even before the traffic volume increases [66]. The autonomous vehicles will therefore contribute to better traffic flow.
- **Controlling traffic with autonomous vehicles:** The observation of traffic on the A9 generates a great deal of current movement and position data, which can be analyzed to find out how vehicles should be controlled in so-called mixed traffic [66].
- **Exchanging real-time data between vehicle and infrastructure:** Vehicles, cameras, and radar systems all collect data on the highway. Car manufacturers and suppliers can use the data from the digital twin in their vehicles to conduct so-called real-world tests [66].
- **Analyzing traffic with autonomous vehicles:** Because autonomous vehicles have an overview of the entire traffic situation and can react proactively. This leads other drivers to avoid frequent lane changes [66].

## 1.2 Problem Statement

As the most fundamental part of the new *AUTOTech.agil* project, the aim of this thesis is to calibrate LiDAR sensors to camera sensors with an automatic, targetless method. More specifically, the LiDAR point cloud data with depth information and 2D image data needs to be used to calculate the extrinsic between them. Since the model should be generalized and tested in real-world data, there will be no external target, e.g. checkerboard or *AprilTags*. The process will be fully targetless – only the natural edge features in the input data will be detected and processed automatically. The calibration task will be subdivided into two parts: On the one hand, a HD map will be used to assist the calibration of camera and LiDAR. A HD map is a high resolution map with rich and accurate information of the roads. It contains the location, size and other information of the roadmarks and the objects (e.g. gantry bridge) on the roads. Matching objects detected by LiDAR with HD map objects can get the extrinsic of LiDAR in the HD map frame. After calibrating the LiDAR to the HD map, using the result of camera-HD map calibration one can easily get the extrinsic of the LiDAR to the camera, which can serve as the initial extrinsic of LiDAR-camera automatic calibration. But due to the *OpenDRIVE* format and the variety of different objects, this calibration can only be done manual.

On the other hand, we will perform automatic LiDAR-Camera Calibration inspired by the voxel-based automatic LiDAR to camera calibration method [50] from Liu et al.. The original method which was developed by Hong Kong University is mainly used for the calibration of indoor scenes. Therefore, the original method has a series of problems when used in the outdoor scene of the *AUTOTech.agil* project. In order to analyze the source of the problem and improve it, the parameters of the model were optimized. In order to solve the interference of the camera's distant background on the calibration, monocular depth estimation was applied to automatically filter the background of the camera image. In addition, adaptive voxelization is introduced and multi-scene optimization is used to improve the generalization ability. Finally, the generalization ability and progress of the new model are tested and analyzed in a variety of different scenarios and sensor pairs, and the results are presented both quantitatively and qualitatively.

Our automatic calibration algorithm is based on the targetless calibration method published by Hong Kong University [91] by Liu et al.. In order to improve the robustness of the method under different external conditions, such as different scene complexities (traffic conditions, obstacles, background buildings), lighting conditions (weather, shadows), sensor conditions (density of point clouds in LiDAR, LiDAR range, the reflectivity of the point cloud, the range of the camera, the brightness of the camera image), the model needs to be improved. Therefore, we use various automatic preprocessing methods and modify the original model to make it adaptive.

## 1.3 Challenges

To summarize, the following two main challenges will be addressed in this thesis:

1. How to calibrate LiDAR sensors to a HD map with a diverse format of object data in the HD map?
2. How to improve the robustness of the automatic calibration of LiDAR and camera in the real-world complex environment under different light and weather conditions so that it can work automatically?

In order to solve the calibration problem between the LiDAR and the HD map under the *OpenDRIVE* standards 1.4 and 1.6, a manual calibration method based on SVD was adopted. The reason is that the objects in the HD map are marked with a limited number of key points and the data format is complex and changeable, thus the automatic calibration method cannot be adopted.

Our automatic calibration algorithm is based on the targetless calibration method published by Hong Kong University [91] by Liu et al.. In order to improve the robustness of the method under different external conditions, such as different scene complexities (traffic conditions, obstacles, background buildings), lighting conditions (weather, shadows), sensor conditions (density of point clouds in LiDAR, LiDAR range, the reflectivity of the point cloud, the range of the camera, the brightness of the camera image), the model needs to be improved. Therefore, we use various automatic preprocessing methods and modify the original model to make it adaptive.

## 1.4 Contribution

The thesis provides a complete analysis of automatic calibration methods between the LiDAR and the camera in targetless environments and solutions to improve the robustness and adaptability of the model proposed by Liu et al. from Hong Kong University [91]. In addition, we also provide solutions for HD map and LiDAR calibration. The main contributions in the thesis can be summarized as follows:

1. We provide a manual HD map to LIDAR calibration solution based on SVD, and calibrate two LiDARs to the HD map in the A9 dataset and visualize the results.
2. We have made a lot of improvements to the targetless calibration method introduced in [91] to enhance its robustness and adaptability under different conditions, and make it able to accept sensor data for calibration in real-time using the publish and subscribe mechanism of ROS(robot operating system [69]). The improvements we have made include:

- Conducting a hyperparameter search to find parameters suitable for outdoor scenes, introducing adaptive—voxelization [50] so that the model can automatically adapt to different scenarios.
- Introducing monocular depth estimation [68] to automatically filter the background to create a background template.
- Using template cropping of the original image to remove background objects outside the LiDAR range.
- Preprocessing the original image to remove the influence of shadows.
- Registering and preprocessing the point cloud from multiple sensors (data extraction, filtering, scattering and upsampling) makes it easy to be calibrated, and calibrates multiple frames of data at the same time to get more robust results.

3. We manually calibrate the LiDARs and cameras in the S110 intersection of the A9 dataset and use it as the ground truth to test our automatic calibration model. We test the model in different scenes and prove its effectiveness and robustness.

## 1.5 Thesis Outline

The thesis is structured as follows: Chapter 1 includes the background of the *Providentia 1*, *Providentia++* and *AUTOTech.agil* project, the problems that need to be addressed, and the main contribution of the thesis. Chapter 2 introduces some theoretical concepts involved in the thesis: the properties of point clouds, point cloud registration, LiDAR to camera calibration, data structure (k-d tree and Octree) and the evaluation metrics of calibration. Chapter 3 briefly introduces some well-known existing manual and automatic LiDAR to camera calibration methods. Chapter 4 introduces the solution we used for HD map to LiDAR calibration and some calibration results. Some results from manual calibration will also be presented. Chapter 5 introduces the solution we used for the automatic calibration [91] and lists contributions we have made to the method. Chapter 6 presents both quantitative and qualitative experiment results. Chapter 7 contains the evaluation of the performance of our model under different situation. We also provide a brief discussion about the experimental results in this chapter. In Chapter 8 we make a short conclusion and in Chapter 9 we give some suggestions for future work.

# Chapter 2

## Theoretical Background

This chapter presents the relevant theoretical knowledge that is a prerequisite to understand the contribution of this thesis. Section 2.1 introduces some typical LiDAR sensors and the main properties of point clouds. Section 2.2 gives a brief introduction of point cloud registration tasks. Section 2.3 gives a brief introduction of LiDAR to camera calibration task and basic knowledge about intrinsic and extrinsic parameters. Section 2.4 gives some basic knowledge of search methods (e.g. KNN) and storage methods of neighbouring data. Section 2.5 introduces the evaluation metrics used in LiDAR to HD Map calibration and LiDAR to camera calibration.

### 2.1 LiDAR Sensor and Point Cloud

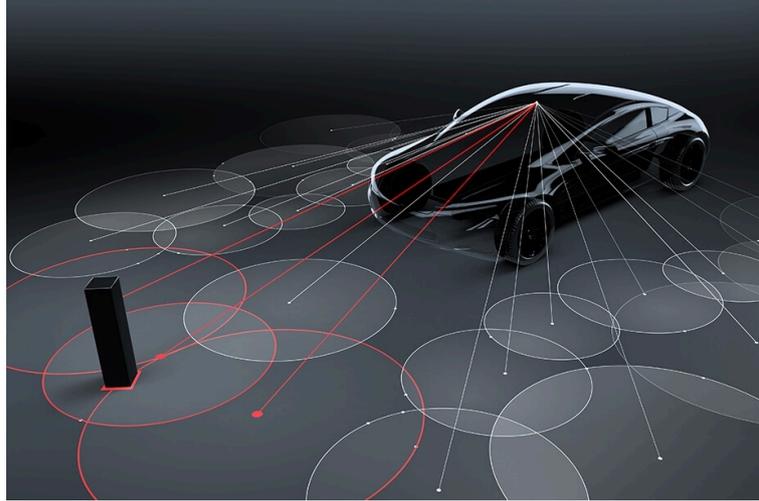
In autonomous driving, the ability to quickly obtain accurate 3D road information is critical for autonomous vehicles. LiDAR is an acronym for Light Detection and Ranging. LiDAR emits a laser beam from a light source (transmitter) and then acquires reflections from objects. The reflected light is received and processed by the receiver, and the time-of-flight (TOF) is used to map the distance [36]. Compared with 2D cameras, LiDARs completely retain accurate 3D information in the depth map, and have an irreplaceable role for autonomous driving. Therefore, it is necessary to introduce the basic principles and the basic characteristics of LiDARs and point cloud data.

#### 2.1.1 Introduction of LiDAR Sensor

LiDAR, also known as optical LiDAR (Light Detection And Ranging), is the abbreviation of laser detection and ranging system. The measurement accuracy of LiDAR is higher than that of traditional equipment. Different wavelengths from infrared (10 micrometers) to ultraviolet (250 nanometers) are used for the measurements [33]. Several different types of scattering are used for different applications, such as Raman scattering [33].

Accurate timing electronics Inertial Measurement Unit and GPS LiDARs mainly include laser emitter, receiver(scanner), timer, GPS(global positioning system) or IMU(Inertial Measurement Unit), and a signal processing circuit [36] [62]. There are two main types of laser emitting parts, one is laser diodes, which usually have two substrate materials, silicon and gallium arsenide [20], and the other is the very hot vertical cavity surface emission laser (VCSEL) such as the LiDAR on the iPhone. The advantages are design flexibility in addressable arrays, low temperature dependence, high reliability, Wafer-Level Manufacturing Process [3] [70].

LiDAR has two main scanning strategies. One is to acquire the entire FoV (field of view) at one time, which is the so-called flash system. Due to its short time consumption and large range, it is mainly used in short-distance scenes less than 100m and has advantages in observing dynamic scenes [2]. The other is to scan only a subset of FoV at a time, and obtain the complete scene through multiple scans, which is the so-called scanning (Figure 2.1), and its advantage is that it has a longer perception distance [2].



**Figure 2.1:** Complete scenes are built by combining subsets of FoV. This method can obtain point clouds with more details, but it also takes more time. Usually scanning is realized by a mechanical drive unit, but more and more people in the industry have questioned the robustness of mechanical components, and solid-state LiDAR is the future development direction [2].

The technical parameters of the two LiDARs we used in our experiments are shown in the table 2.1. For convenience, we use the company name as the abbreviation of the names of these two LiDARs. In our calibration task, there are three LiDARs available, two of them (S110 north and S110 south) are Ouster LiDARs, and the other (S110 north west) is a Valeo LiDAR.

Number of Channels is the most important attribute for the calibration task, LiDAR with more channels will generate a more dense point cloud in one scan. Ouster has 64 channels but Valeo only contains 16. Because Ouster has more laser beams, it reaches a higher accuracy (1.5-10 cm) than Valeo (10 cm).

Another important attribute is the Range of LiDAR, both of them has enough range for our calibration task.

The quality of point clouds also depends on the rotation rate, the faster, the better.

The bandwidth is also important for the multi sensor calibration and data fusion, because we need synchronized sensor data, higher bandwidth means lower latency.

<i>Company</i>	<b>Ouster</b>	<b>Valeo</b>
<i>Picture</i>		
<i>Type/Model</i>	OS1-64 multi-beam flash LiDAR Gen 2	SCALA 2 (Gen. 2)
<i>Num. Channels</i>	64	16
<i>Range</i>	120 m	150 m
<i>Angular resolution Horiz.</i>	0.18 (512, 1024, 2048) deg	0.25 deg
<i>Angular resolution Ver.</i>	0.5 (0.35 – 2.8) deg	0.6 deg
<i>Precision/Accuracy</i>	1.5-10 cm	10 cm
<i>FOV Horiz.</i>	360°	133°
<i>FOV Ver.</i>	33.1°(-16 to + 16)	10°
<i>Rotation rate (frame rate FPS)</i>	10 or 20 Hz	23 FPS
<i>bandwidth</i>	130 Mbps	9 Mbps
<i>Price</i>	\$12, 000	EUR 3, 990

**Table 2.1:** Two types of LiDARs used in our experiment. Attributes of LiDARs are listed on the left. Ver. means vertical and horiz. means horizontal. It should be pointed out that the performance of Ouster is significantly better than that of Valeo. The data is provided by Walter Zimmer in the AUTOTech.agil project [100].

### 2.1.2 Properties of Point Cloud Data

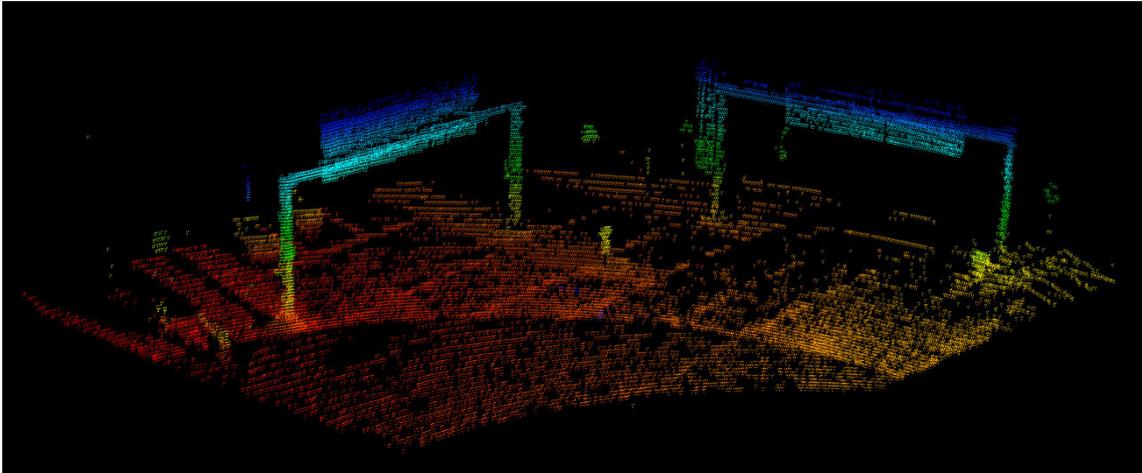
A point cloud is a set of points in a certain coordinate system. A point contains a wealth of information, including three-dimensional coordinate  $[x, y, z]$ , color, reflectivity value, intensity value, time of flight, and more. Figure 2.2 shows an example point cloud.

Typically, point clouds are stored in PCD file format (.pcd). The PCD format has a file header that describes the overall information of the point cloud: a readable header that defines the number, size, dimension and data type of the point cloud; a data segment that can be ASCII or binary [47]. The data body part is composed of Cartesian coordinates of points, and spaces are used as separators in ASCII mode [47]. Figure 2.3 shows an example of .pcd file.

The .pcd storage format is the official format specified by the PCL library, a typical format tailored for point clouds. The advantage is that it supports the n-dimensional point type extension mechanism, which can better utilize the point cloud processing performance of the PCL library. There are two content formats, ASCII and binary.

Raw point clouds generated by sensors usually have the following disadvantages:

1. High dimensionality: Generally, the point cloud generated by the sensor contains very rich attributes in addition to Cartesian coordinates, such as reflectivity, intensity, time, etc. Many dimensions are unnecessary for our task, but they will significantly increase the size of the PCD file and occupy memory, which will slow down the processing time and increase the computational burden.



**Figure 2.2:** Point cloud of S110 intersection on the A9 highway. This point cloud is registered from all 3 LiDARs in S110 intersection, cropped, 100 times scattered, filtered and only reserve 4 dimension (x, y, z, intensity), here the intensity is visualized.

```

# .PCD v.7 - Point Cloud Data file format
VERSION .7
FIELDS x y z rgb
SIZE 4 4 4 4
TYPE F FFF
COUNT 1 1 1 1
WIDTH 213
HEIGHT 1
VIEWPOINT 0 0 0 1 0 0 0
POINTS 213
DATA ascii
0.93773 0.33763 0 4.2108e+06
0.90805 0.35641 0 4.2108e+06

```

**Figure 2.3:** An example PCD file. Actually .pcd is a .txt file, including header and data body. This PCD file is stored in ASCII format and only contains four dimensions.

2. Ineffectiveness: Usually the original point cloud will contain many useless coordinate points. Some are caused by stains on the LiDAR, while others are points outside the task's area of interest.

3. Sparsity: Since the density of the laser beam is limited, the point cloud obtained by a single scan is usually too sparse to fully capture the features of the target. Such a point cloud cannot be used directly for calibration.

Due to the characteristics of the point cloud listed above, the point cloud acquired from the sensor needs to be preprocessed before it can be used for calibration.

## 2.2 Point Cloud Registration

Point cloud stitching and registration refer to the same concept, which is to transform point clouds at different positions to the same position through the information of overlapping parts [38]. Below we use the term registration to describe this process. Registration is generally divided into three categories: coarse registration, fine registration and global registration.

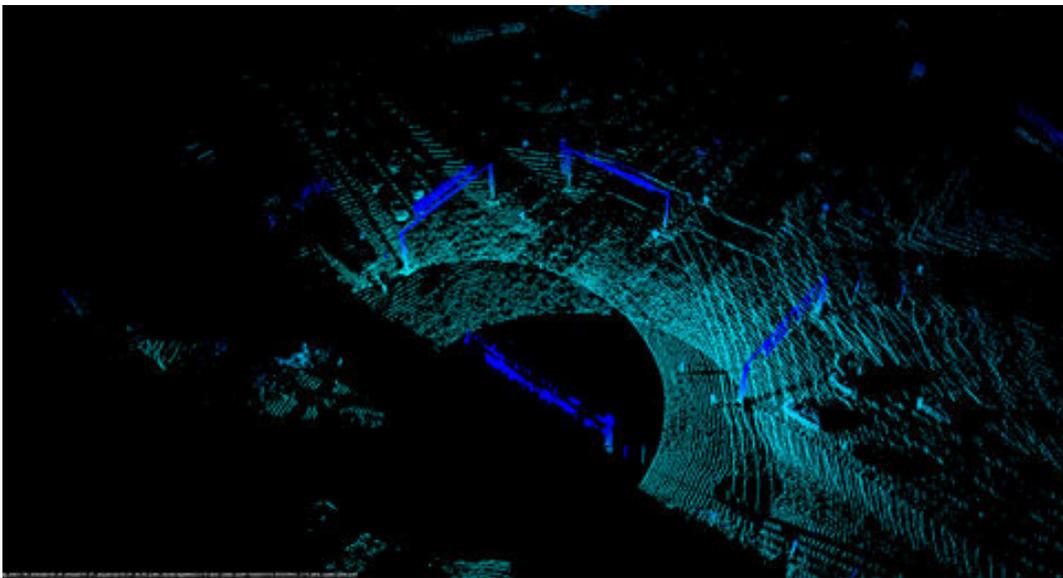
**Coarse registration:** It is generally used to register two point clouds with very different positions, such as two frames of point clouds in the camera coordinate system. Coarse registration methods are roughly divided into two categories: coarse registration with markers and coarse registration without markers [11].

**Fine registration:** This generally refers to the ICP registration method. Besl and McKay published this method in 1992 [8]. Nowadays it has become the most fundamental method for registration. It is mainly used for the point cloud that has been roughly registered and needs to improve the registration accuracy.

**Global registration:** The point cloud data registered frame by frame often has cumulative errors. Global registration can spread the accumulated error to each frame, thereby reducing the overall registration error [58].

Since the point cloud information obtained from a single perspective can only reflect a part of the object, there will be dead spots when performing LiDAR to camera calibration. Through point cloud calibration, point clouds from different perspectives can be fused into one coordinate system to obtain more complete scene information, which can effectively avoid the mismatch between LiDAR and camera data.

Figure 2.4 shows an example of registered point cloud.



**Figure 2.4:** registered point cloud. The point cloud in the figure is obtained by registering point clouds from three different LiDARs into the coordinate system of S110 south2. This can be seen from the ring scan traces on the ground. By registering, we can not only obtain the information of the whole scene in a LiDAR coordinate system, but also take advantage of the details of the overlapping parts of the LiDAR view - which will help the calibration task.

## 2.3 LiDAR-Camera Calibration

Generally, there are two types of methods for calibrating the external parameters between the LiDAR and the camera: one is to use the 3D-3D constraint, that is, the three-dimensional laser point (3D) measured by the laser and the three-dimensional coordinates (3D) of the calibration plate measured by the camera [21]. Both 3D points are used to construct constraints. The other one is to use 3D-2D constraints, that is, to use laser-measured three-dimensional laser points (3D) and image two-dimensional features (2D point features, line features) to build constraints [45].

This thesis mainly explains the method of using 3D-2D constraints to calibrate external parameters. This method is essentially similar to the PnP [28] and PnL [52] problems.

Perspective-n-Point(PnP) [28] is to estimate the pose of a camera given a set of 3D points in the world coordinate and their 2D projections in the image frame. It has many variants:

- P3P only use 3 point only use 3 point pairs. It is the oldest version of PnP. It can be even dated back to 1841 [67].
- EPnP (efficient PnP) solves the general problem of PnP for  $n \geq 4$ . It was published by Lepetit, et al.in 2008 [44].
- SQPnP is a non-minimal, non-polynomial solver. It transforms PnP as a non-linear quadratic program. SQPnP was introduced by Terzakis and Lourakis in 2022 [79].

The Perspective-n-Line (PnL) problem is to estimate the pose of a camera from N 2D/3D line correspondences [96]:

- P3L use 3 lines to estimate the pose. Many P3P method has been published: Dhome et al., for example, published their P3P method in 1989 [23], Chen in 1990 [15], and Xu et al. in 2017 [87].
- The DLT method only need 6 pairs of 3D points and 2D points [1], is the most popular PnL nowadays.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K[R|t]_c^L \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}^L \quad (2.1)$$

As shown in equation 2.1, the aim of calibration is to calculate extrinsic parameters of a rigid-body transformation based on the matched point pair between LiDAR and camera. Where K contains the intrinsic parameters of the camera,  $[R|t]_c^L$  is the six-degree-of-freedom (6-DOF) rigid-body transformation,  $t = (t_x, t_y, t_z)$  is the translation vector, and  $R = f(\theta_x, \theta_y, \theta_z)$  is the rotation matrix. Besides that, since the camera and LiDAR are fixed in the infrastructure of the intersection, we have the initial extrinsic  $R_0$  and  $t_0$ , so there is no need to consider the coarse calibration.

After that we use the initial extrinsic  $R_0$  and  $t_0$  and equation 2.1 to project the LiDAR points onto the camera image. Then we will calculate the reprojection error (i represent i-th. point):

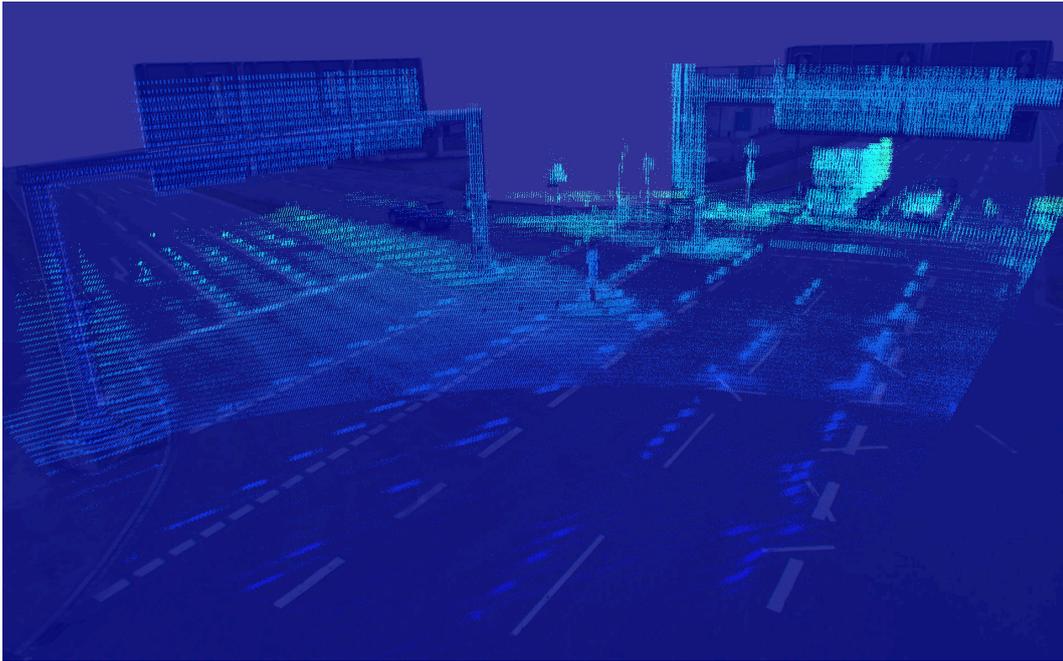
$$\epsilon_i = \begin{bmatrix} u_i \\ v_i \end{bmatrix} - K[R|t]_c^L \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix}^L \quad (2.2)$$

We use *bundle adjustment* (BA) [80] to find the optimal extrinsic. Bundle adjustment amounts to jointly refining a set of initial camera and structure parameter estimates for finding the set of parameters that most accurately predict the locations of the observed points in the set of available images. More formally, assume that  $n$  3D points are seen in  $m$  views and let  $x_{ij}$  be the projection of the  $i$ th point on image  $j$ . Let  $v_{ij}$  denote the binary variables that equal 1 if point  $i$  is visible in image  $j$  and 0 otherwise. Assume also that each camera  $j$  is parameterized by a vector  $a_j$  and each 3D point  $i$  by a vector  $b_i$ . Bundle adjustment minimizes the total reprojection error with respect to all 3D point and camera parameters, we have

$$\min_{a_j, b_i} \sum_i \sum_j (v_{ij} d(Q(a_j, b_i), x_{ij}))^2.$$

where  $Q(a_j, b_i)$  is the predicted projection of point  $i$  on image  $j$  and  $d(x, y)$  denotes the Euclidean distance between the image points represented by vectors  $x$  and  $y$ .

Optimizing the reprojection error, BA will get a optimal extrinsic. After obtaining the optimal extrinsic, we project the PC onto a 2D image to gain an intuitive view of the calibration results. A calibration result of early work is shown in fig 2.5.



**Figure 2.5:** Calibration result from our early work, the lane marks on the ground is not converged and we will improve the algorithm to get better calibration result in chapter 6.

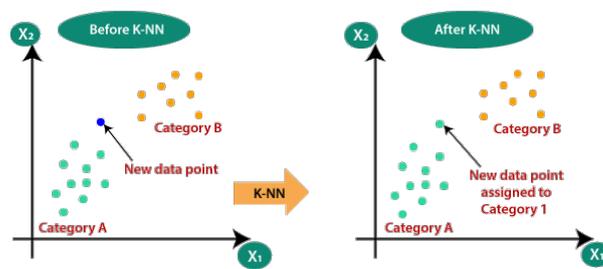
## 2.4 Search algorithm

The LiDAR-camera automatic calibration method used in this work first needs to project the 3D point cloud into the 2D image through the initial extrinsic parameter, and then establish the LiDAR-camera point pair (edge) through the nearest neighbor search. After that, iteratively reduce the distance of all point pairs. Therefore, it is necessary to introduce the nearest

neighbor search method and the storage method of the data.

### 2.4.1 KNN

The KNN (K-Nearest Neighbor) algorithm is one of the most basic and simplest algorithms in machine learning algorithms. It can be used for both classification and regression. KNN performs classification by measuring the distance between different feature values. The idea of the KNN algorithm is very simple: for any  $n$ -dimensional input vector, each corresponds to a point in the feature space, and the output is the category label or predicted value corresponding to the feature vector. We search the  $k$  nearest neighbours of input points in the feature space, and classify the input point as the the same category of the neighbours with largest number [30]. For example, an input vector  $x$  that needs to be predicted, we only need to find a set of  $k$  vectors closest to the vector  $x$  in the training data set, and then predict the category of  $x$  as  $k$ . Figure 2.6 shows how a input point is classified by KNN.



**Figure 2.6:** The input point is assigned to category 1 since it has more points in the neighbourhood of input point. [41]

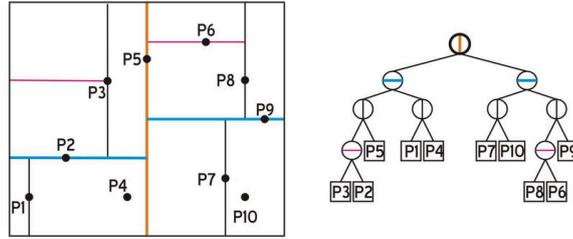
### 2.4.2 KDTree

Usually, when the pixel and point cloud density is small, the nearest neighbor can be found by directly calculating the distance from the projected point to each pixel one by one. However, point cloud data usually contains tens of thousands to millions of points, and the number of pixels of the camera image are usually large, so the cost of calculating the Euclidean distance one by one is unacceptable.

In order to quickly find  $k$  neighbors, we can consider using a special data structure to store training data to reduce the number of searches. Among them, *KDTree* is the most famous one.

A *KDTree* (K-dimension Tree)(figure 2.7) is a binary tree data structure that stores instance points in  $k$ -dimensional space for fast retrieval. This algorithm is equivalent to continuously dividing the  $k$ -dimensional space by using a hyperplane perpendicular to the coordinate axis to form a series of  $k$ -dimensional super-rectangular regions [7]. Each node of a *KDTree* corresponds to a  $k$ -dimensional hyper rectangular region. The use of a the tree can avoid most of the data point search, thereby reducing the amount of calculation of the search.

We use a recursive method to construct a *KDTree*: (1) Construct the root node so that the root node corresponds to the super-rectangular area of all points contained in the  $k$ -dimensional



**Figure 2.7:** KDTree example: The 2D space is subdivided by hyper plane, the subdivision is performed recursively (left) and the corresponding binary tree structure of the subdivided data (right). [78]

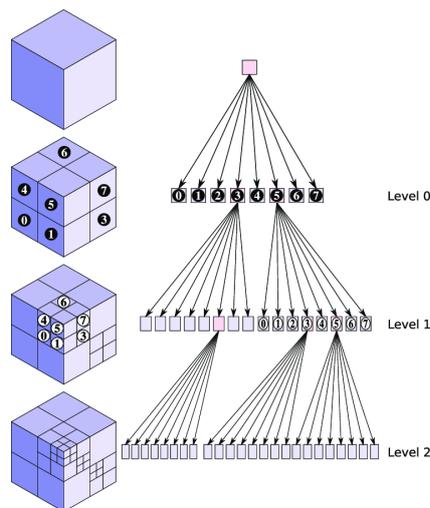
space; (2) continuously segment the  $k$ -dimensional space to generate child nodes. Usually, we select the coordinate axis cyclically to divide the space. When a dimension coordinate is selected, we select the median of all training instances on the coordinate axis as the segmentation point. At this point, the constructed tree is a balanced binary tree, like Fig. 2.7 shows.

Using a *KDTree*, we can quickly find the nearest neighbor pixels of a large number of projected points.

### 2.4.3 Octree

KDTree is usually used for the division and storage of two-dimensional data, but our calibration also needs to voxelize [32] the 3D point cloud to extract the edges, which requires a storage method in three-dimensional space. Usually Octree are used for space division. An octree (figure 2.8) is a tree data structure that subdivided in 8 nodes. Each internal node has exactly eight children node. *Octrees* are usually used to partition a three-dimensional space by recursively subdividing it into eight octants [57].

In our task, Octree will be adopted to store the voxels of point cloud in 3D spaces.



**Figure 2.8:** Topology of octree. The space is subdivided into 8 sub spaces (left), each node also has 8 children (right). The partition will be conducted recursively [43].

## 2.5 Evaluation Metrics

For LiDAR to HD map calibration, we choose the RMSE loss as the metric.

RMSE (Root Mean Square Error) is the square root of the ratio of the square of the deviation between the predicted value and the true value to the number of observations  $n$  [14].

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n \left( \frac{d_i - f_i}{\sigma_i} \right)^2} \quad (2.3)$$

For automatic LiDAR to camera calibration, we use the normalized optimization cost  $\frac{1}{N_{match}} \mathbf{r}^T (\mathbf{J}_w \Sigma \mathbf{J}_w)^{-1} \mathbf{r}$  [50] to represent the convergence of the extrinsic parameter. It was derived from the MLE loss in the paper published by Liu et al.. The underlying principle is introduced in 5.1.

We also adopt benchmark to evaluate the performance of calibration in Chapter 7. We directly use difference of the PRY (pitch, roll, yaw) in degree and XYZ difference (in meter) between calibration result and the benchmark (manual calibration result) as the metrics.

$$Loss = Extrinsic_{result} - Extrinsic_{benchmark} \quad (2.4)$$



# Chapter 3

## Related Work

This chapter gives an overview of the related work in the automatic sensor calibration domain. Section 3.1 introduces some well-known datasets in the field of LiDAR to camera calibration. Section 3.2 describes several state-of-the-art target based LiDAR to camera calibration methods and targetless methods. In Section 3.3 we will introduce some upsampling methods.

### 3.1 LiDAR-Camera Calibration Datasets

This section mainly introduces the commonly used datasets for LiDAR-camera calibration tasks. Typically, LiDAR-based 3D object detection datasets and autonomous driving datasets are adopted for calibration tasks. So the dataset presented here is actually mainly used for 3D object detection. This section contains new datasets, from the highly cited KITTI dataset to the current research frontier of multi-modality, time-series fusion, etc.. According to the scene, the datasets can be divided into indoor and outdoor datasets. We will also highlight the outdoor dataset for infrastructure, which will be directly related to our experiments—the A9 dataset.

#### 3.1.1 Indoor Datasets

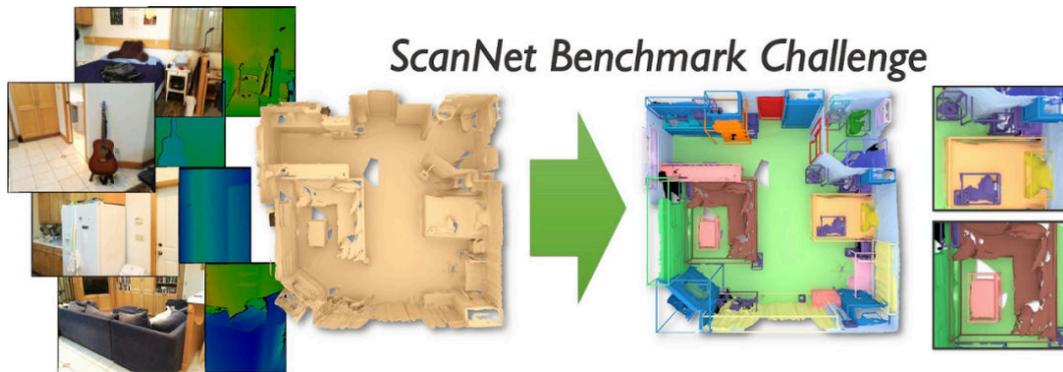
The indoor 3D object detection is a relatively a new research field. It mainly include ScanNetV2 and SUN RGB-D.

**ScanNetV2** is an indoor dataset proposed by Stanford University, Princeton University and Technical University of Munich in CVPR18SH (Fig. 3.1). ScanNet is an RGB-D video dataset, which can be used for semantic segmentation and target detection tasks. It contains a total of 1513 collected scenes<sup>1</sup>. ScanNet contains a total of 21 categories of objects, of which 1201 scenes are used for training and 312 scenes are used for testing [75]. The dataset contains 2D and 3D data. The 2D data includes N frames in each scene; in order to avoid overlapping information between frames, every 50th frame is taken [18]. 2D labels and instance data are provided as .png image files. Color images are provided in 8-bit RGB .jpg files and depth images as 16-bit .png files [18]. The information contained in each frame is color, depth,

---

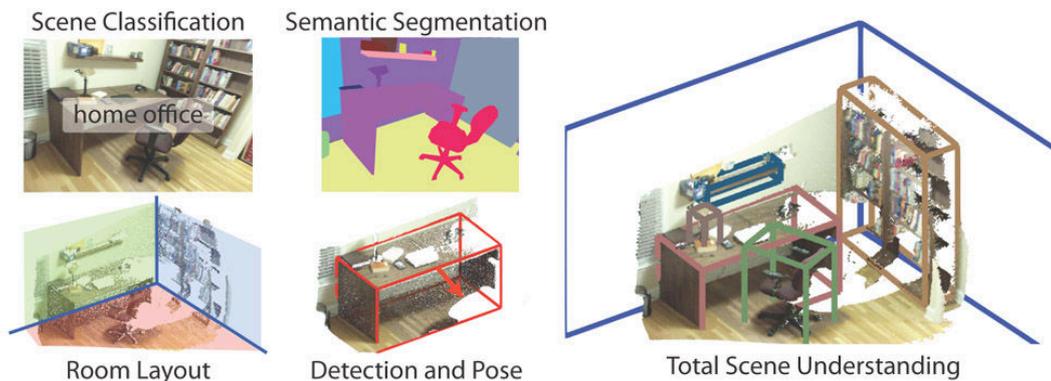
<sup>1</sup>The number of point clouds in each scene is different[75]. If you want to use end-to-end, you may need to sample (using e.g. FPS sampling), so that the points of each scene are the same.

instance-label, label, and corresponding pose. The 3D data is a series of .ply files.



**Figure 3.1:** ScanNetV2, contain a total of 1513 collected scenes. The figure is captured from the official website of ScanNet Benchmark [74].

**SUN RGB-D** is an indoor dataset proposed by Princeton University for segmentation and detection tasks (Fig 3.2). This dataset contains 10335 RGB-D images and is similar in scale to Pascal VOC [82]. The entire dataset is densely annotated, including 146,617 2D polygon annotations and 64,595 3D bounding boxes with precise object orientations, as well as 3D room layouts and scene categories for each image [73]. The data set is the union of the three datasets: NYU depth v2, Berkeley B3DO, and SUN3D [73].

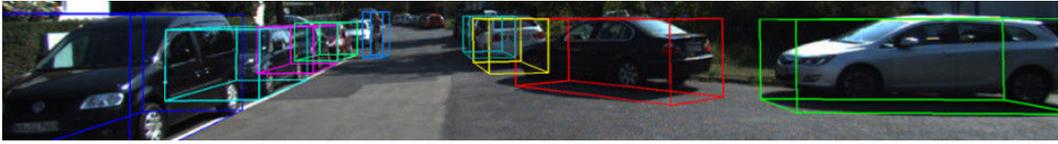


**Figure 3.2:** SUN RGB-D. The figure is accessed from the official website of SUN RGB-D [82].

### 3.1.2 Outdoor Datasets

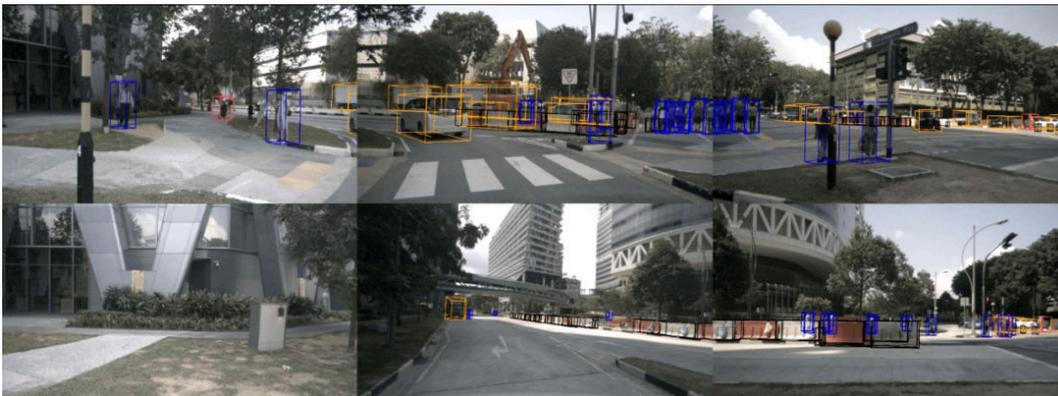
The **KITTI** dataset is jointly established by the Karlsruhe Institute of Technology in Germany and the Toyota American Institute of Technology (Fig. 3.3). It is currently the largest computer vision algorithm evaluation dataset in the field of autonomous driving scenario over the world [46]. This data set is used to evaluate the performance of computer vision technologies such as stereo, optical flow, visual odometry, 3D object detection and 3D tracking in the vehicle environment. KITTI contains real image data collected from scenes such as urban areas, rural areas, and highways [59][77]. There are up to 15 vehicles and 30 pedestrians in each image, as well as various degrees of occlusion and truncation [77]. The entire dataset

consists of 389 pairs of stereo images and optical flow maps, a 39.2 km visual odometry sequence, and images of more than 200k 3D labeled objects, sampled and synchronized at a frequency of 10 Hz [31]. Overall, the original dataset is categorized as 'Road', 'City', 'Residential', 'Campus' and 'Person' [31]. For 3D object detection, the label is subdivided into car, van, truck, pedestrian, pedestrian (sitting), cyclist, tram and misc [76].



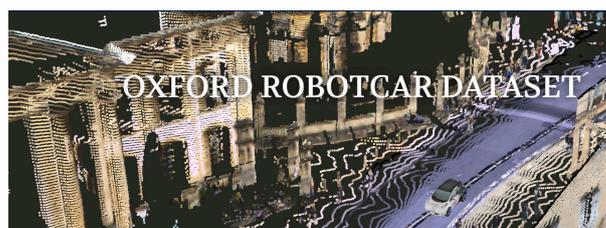
**Figure 3.3:** KITTI, figure was accessed from the official website of KITTI[76]

The **nuScene** dataset (Fig 3.4) consists of 1,000 scenes, each scene is 20 seconds long, and contains a variety of scenarios; in each scene, there are 40 key frames, in another word, there are 2 key frames per second [12]. The key frames are manually marked, and there are several annotations in each frame, the form of the mark is a bounding box. Not only the size, range, but also category, visibility, etc. are marked. This dataset released a teaser version (containing 100 scenes) in 2018, and the official version (1,000 scenes) was released in 2019 [12].



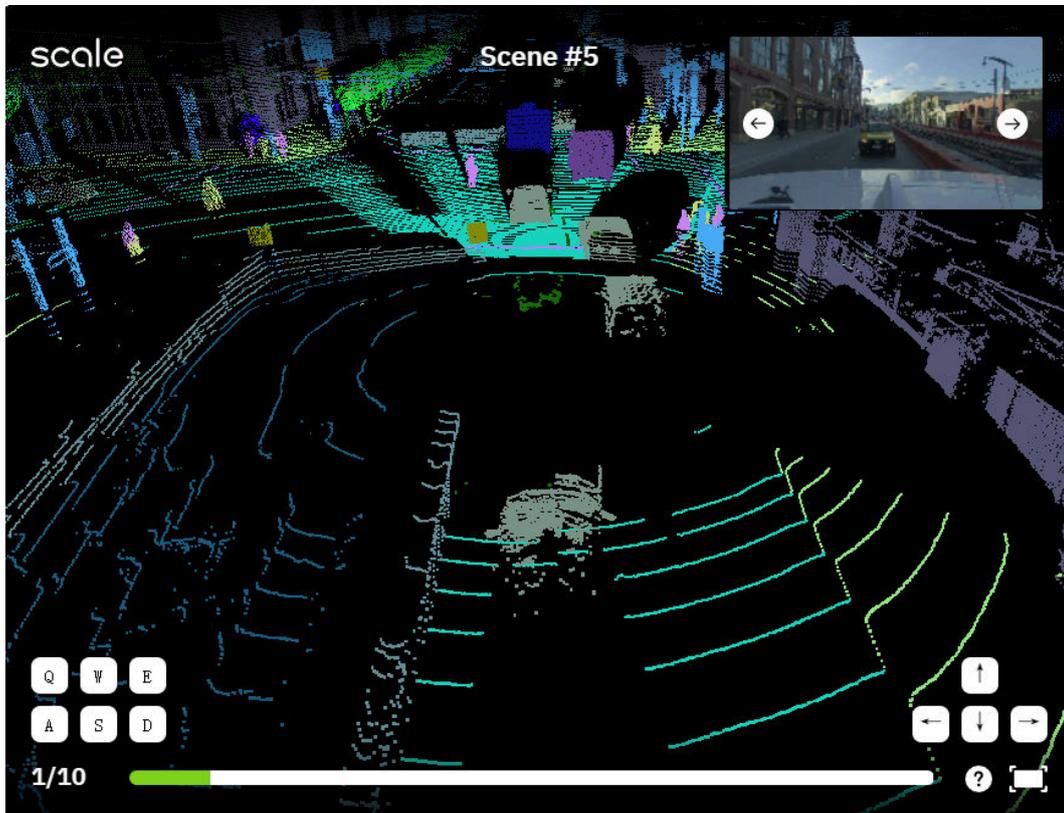
**Figure 3.4:** Nuscence dataset. The picture was accessed from the official website [39]

The **Oxford Robotcar** (Fig 3.5) was proposed by the Robotics Laboratory at the University of Oxford. Its radar is a Navtech CTS350-X frequency-modulated continuous wave (FMCW) scanning radar. It provides a range resolution of 4.38 cm, a rotational resolution of 0.9 degrees, and a maximum range of 163 m[55].



**Figure 3.5:** Oxford Robotcar, the picture was accessed from the official website [40]

**PandaSet**(Fig 3.6) was collected in San Francisco. It contains a total of 48,000 camera images, 16,000 LiDAR scans, 100+ scenes, each scene is 8 seconds long and contains a total of 28 annotation classes and 37 semantic segmentation labels[71]. It is an object detection dataset for autonomous driving scenes that integrates industry and academia.



**Figure 3.6:** PandaSet, the picture was accessed from the official website of PandaSet [71]

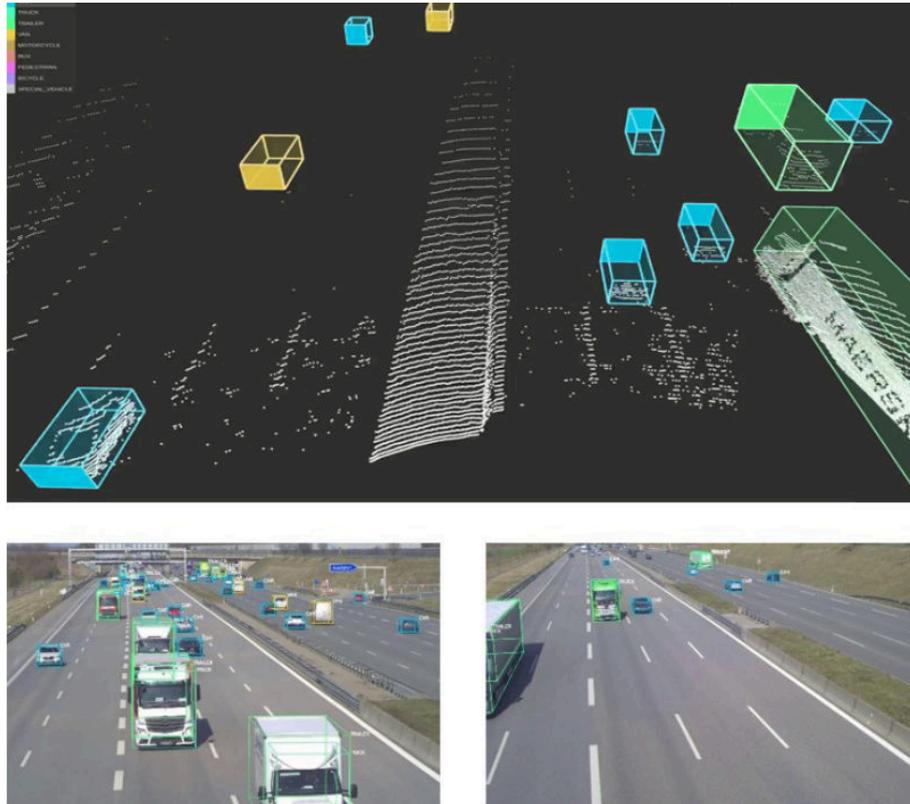
**Waymo**, the self-driving company owned by Google's parent company Alphabet, announced the Waymo Open Dataset project on its blog on August 21, 2019[86]. As far as data is concerned, Waymo contains 3,000 driving records, with a total duration of 16.7 hours and an average length of about 20 seconds; 600,000 frames, a total of about 25 million 3D bounding boxes, 22 million 2D bounding boxes, and a variety of autonomous driving scenes[53].

### 3.1.3 Outdoor infrastructure Dataset

The infrastructure dataset is a subset of the outdoor dataset. Since our calibration task is based on the infrastructure sensor in the S110 intersection, we will introduce the dataset directly related to our calibration task—the A9 dataset.

The importance of autonomous driving technology based on data-intensive machine learning has become increasingly prominent, and high-quality real-world data is an important prerequisite for this technology[16]. In order to collect and study the detailed traffic data for developing further value added services, project Providentia++ came into being; and has been changed the name to Providentia++ with the Chair of Robotics, Artificial Intelligence and Real-time Systems at the Technical University of Munich's Department of Informatics serving as consortium leader[81].

A9-Dataset was released by project Providentia++ based on roadside sensor infrastructure from the 3 km long Providentia++ test field near Munich in Germany. "The dataset includes anonymized and precision-timestamped multi-modal sensor and object data in high resolution, covering a variety of traffic situations; the first set includes in total more than 1000 sensor frames and 14000 traffic objects [16]." It contains LiDAR frames from two gantry bridges on the A9 highway in Munich with the corresponding objects labeled with 3D bounding boxes [81]. Fig 3.7 shows an example of labeled data in A9 dataset.

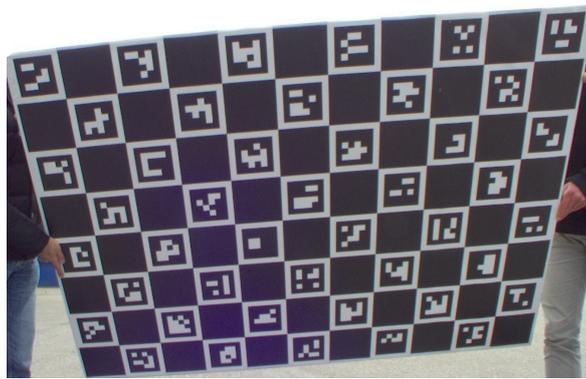


**Figure 3.7:** "Labeled frames from LiDAR and cameras on two measurement stations on the freeway [16]"

## 3.2 LiDAR-Camera Calibration Methods

Extrinsic calibration is a well-studied field in robotics and is mainly divided into two categories: target-based and targetless. The primary distinction between them is how they define and extract features from the sensors.

Geometric solids, AprilTag (Fig. 3.8) and chessboards have been widely used in object-based methods. It is mainly due to their explicit constraints on plane normals and the simplicity of problem formulation. With these easily identifiable objects, the correspondence between point clouds and images can be easily established and high accuracy can be achieved. However, due to the need to place targets in scenes, this method cannot be used in real-world autonomous driving scenarios at all.



**Figure 3.8:** An example of target for calibration, here the AprilTag was used in earlier manual calibration task of Providentia++ project [99].

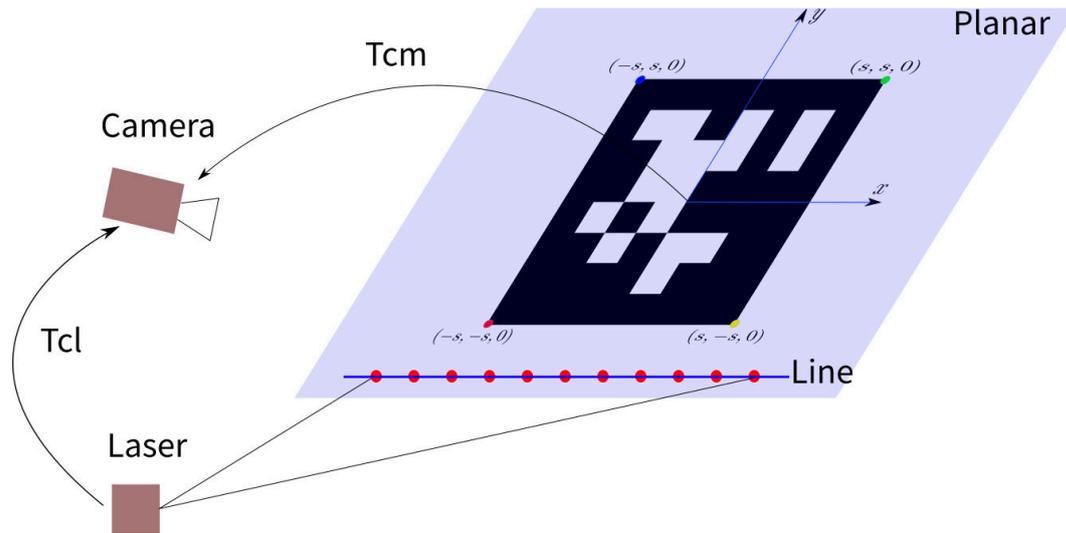
Targetless methods do not detect explicit geometric shapes from known targets. Instead, they use the plane and edge features that exist in nature.

### 3.2.1 Target-based Methods

This section will introduce some target-based calibration methods.

The **Camera Laser Calibration Tool** is a ROS-based automatic calibration toolbox for single-line laser and camera extrinsic calibration [93]. The calibration principle is shown in Fig. 3.9. The camera estimates the plane of the board in the camera coordinate system through the AprilTag. Since the point cloud falling on the TAG has easily identifiable geometric features, these features (such as straight lines) can be used to establish the correspondence between point cloud and pixel. Once the correspondences are built, the algorithm calculates the distance from the point to the plane as the error, and solves it using the nonlinear least squares method.

In the paper "**LiDAR-Camera Calibration using 3D-3D Point correspondences**" introduced by India IIT Robotics Research Lab, they implemented two methods. The first method is based on 2D-3D correspondences, using a hollow rectangular cardboard as the target,

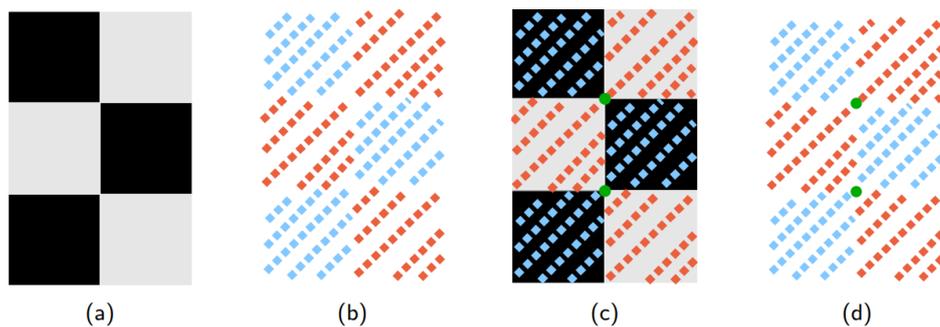


**Figure 3.9:** Camera Laser Calibration Tool, using AprilTag on the plane as the target for detection. Correspondences will be established between AprilTag in the image and the LiDAR point clouds on the plane (the line in the figure) [93].

manually marking 2D corner pixels on the image; manually selecting line intersection in the point cloud to solve 3D corner points, and then using PnP and RANSAC to get the extrinsics [22]. The second method is based on 3D-3D correspondences, the main difference from the first method is the extraction of features in the image. By using the two-dimensional ArUco markers, the 3D coordinates of the corner points in the camera coordinate system can be directly calculated, and then the extrinsic parameters can be solved by ICP [22].

The ILCC method from Nagoya University, Japan, is a unique 3D corner extraction method. Based on the correlation between the point cloud reflection intensity and the color mode, a chessboard is used to estimate the corners in the point cloud. The principle shows in Fig. 3.10.

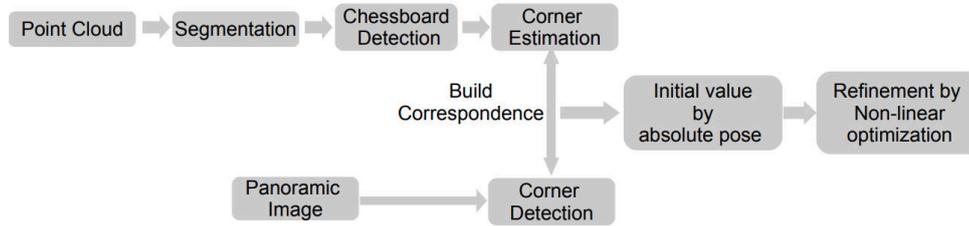
The overview of the proposed method is illustrated in the Fig. 3.11). First, the point cloud



**Figure 3.10:** (a) chessboard; (b) point cloud of the chessboard. Colors indicate the intensity (blue for low and red for high reflectance intensity); (c) Find a matrix that translates the most 3D points on the corresponding patterns on the board and estimate corners (Green points); (d) Consider the corners of the chessboard as the that of point cloud [85].

obtained from the LiDAR is segmented into multiple parts. The corners of the chessboard in the point cloud are estimated by minimizing a defined cost function. On the other hand, corners of the chessboard in the image are detected. A correspondence of the corners is built

based on the predefined counting order. The corresponding pairs are then used to estimate an initial value of the transformation matrix by solving an absolute pose problem. Finally, the value is refined by optimizing a nonlinear cost function.



**Figure 3.11:** Overview of the ILCC pipeline. Detecting corners in 3-D point cloud and camera image, then use chessboard to build the correspondence and use it for calibration [85].

Fang,C. et al. proposed an extrinsic calibration method using multiple cameras and 3D LiDARs. It is a **single-shot solution for calibrating extrinsic transformations among multiple cameras and 3D LiDARs**. They establish a panoramic infrastructure, in which a camera or LiDAR can be robustly localized using data from a single frame [27]. Experiments are conducted on three devices with different Camera-LiDAR configurations, showing that their approach achieved good calibration accuracy. The result shows in table 3.1

Method	Frames	Baseline (cm)	Reprojection Error (px)	Standard Deviation (cm)
Kalibr	150	12.09	0.18	0.01
AprilTags	1	12.01	0.62	0.13
CCTags	1	12.11	0.31	0.04

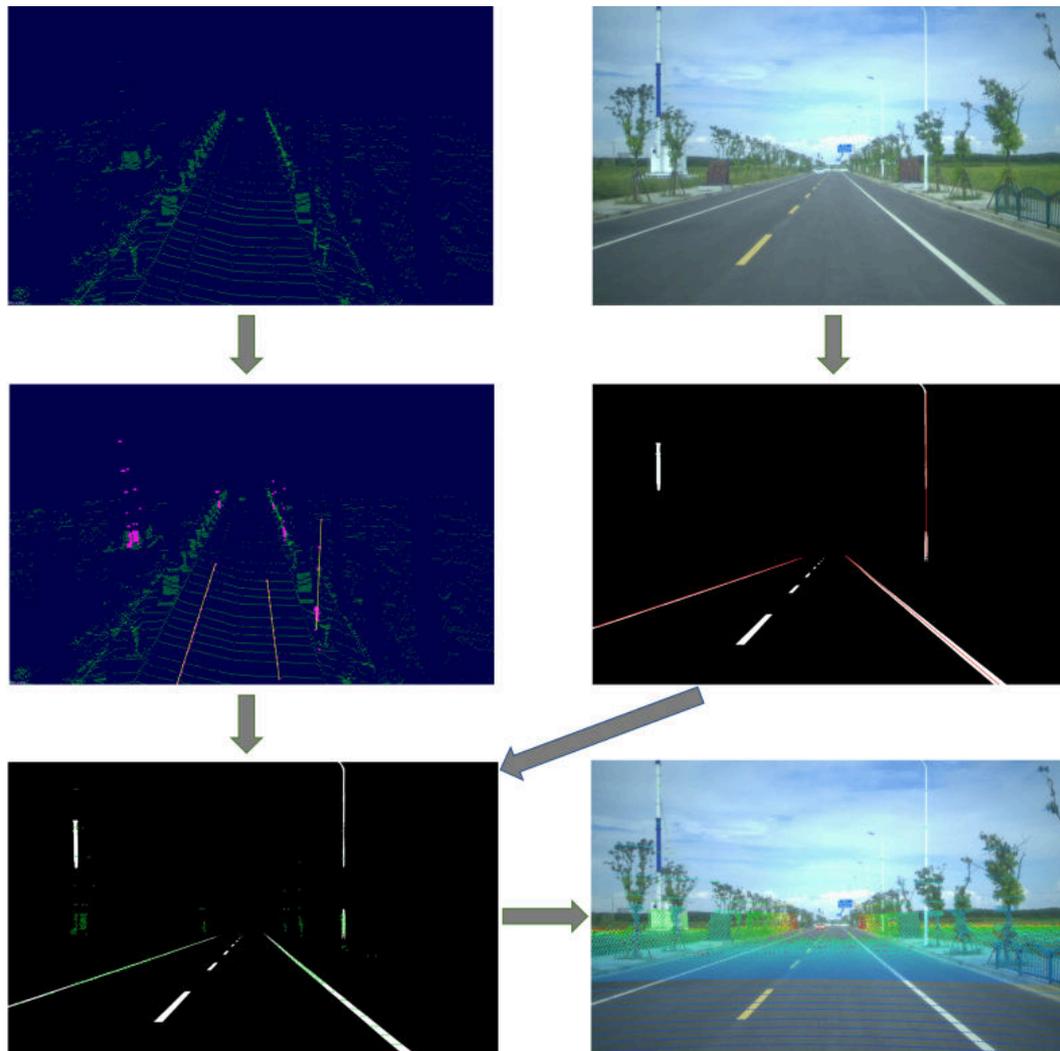
**Table 3.1:** Comparison results of the stereo camera, the reprojection error is within 1 pixel for all 3 cases [27]

### 3.2.2 Targetless Methods

This section will introduce targetless methods. Compared with target-based methods, targetless methods don't require a pre-deployed target in the scene and are capable to extract features (points, edges, lines and planes etc.) from raw data. So they have better prospects in the field of autonomous driving and our automatic calibration method is also based on targetless methods.

**CRLF** [54] extract line features and use P3L to calculate the extrinsic. For line feature extraction in LiDARs, they separate the point cloud into ground and background(objects). They use the intensity of the reflection to extract the lane on the ground; furthermore, a grid is used to separate the background point cloud. And they select the points with sufficient height in the background, then using RANSAC to fit the line in point cloud. For the camera image they use *BiSeNET-V2* [89] to make semantic segmentation, and use *Dense CRF* [35] for refinement.

Fig.3.12 shows how line features are extracted.



**Figure 3.12:** CRLF extracts straight line features from both the image (white region) and point cloud (pink points), and estimates the coarse calibration by solving the P3L problem under line feature constraints [54].

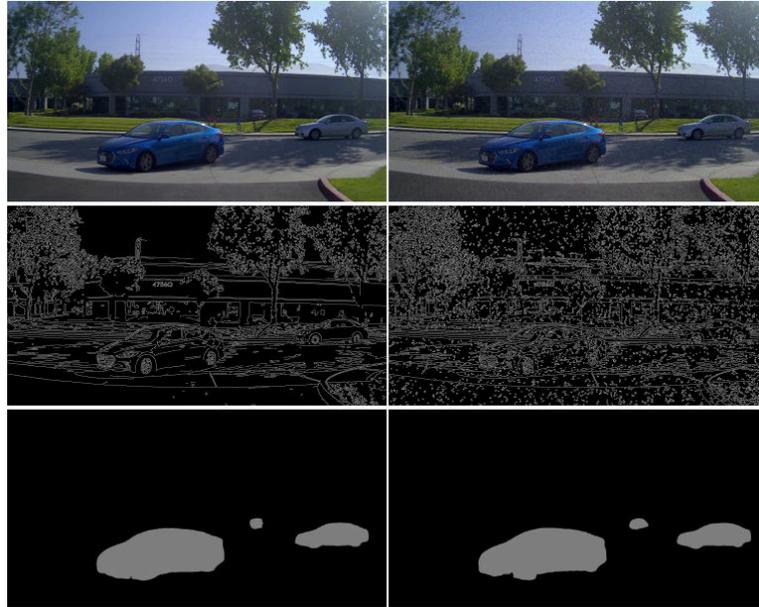
For the calibration they do coarse calibration first (P3L problem). Then they minimize the cost function of the correlation. The experiment is based on *KITTI* [77] dataset.

The next method (**Online Camera-LiDAR Calibration with Sensor Semantic Information**) introduced an online calibration method that can automatically computes the optimal rigid motion transformation between the aforementioned two sensors. And maximizes their mutual information of perceived data, without the need of tuning environment settings [97].

By formulating the calibration as an optimization problem with a novel calibration quality metric based on semantic features, they successfully and robustly align pairs of temporally synchronized camera and LiDAR frames [97].

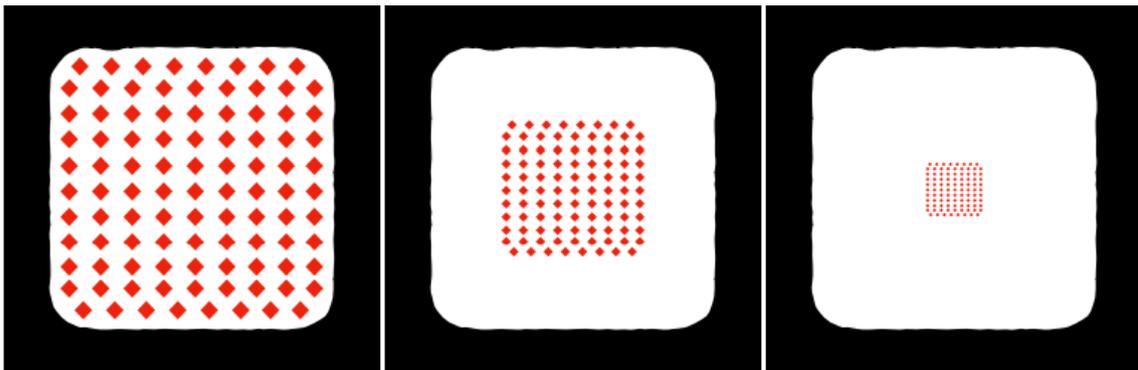
As the Fig.3.13 shows, they extract semantic objects in the images instead of edge features like most line-based targetless calibration methods. The main disadvantage of the Canny edge

detector is the robustness in complex environments, when there is shadow on the ground or the surface texture of the objects are too complex, Canny edge detector[13] will extract many useless edges, which will build wrong correspondences between LiDAR edges and image edges and influence the quality of the calibration.



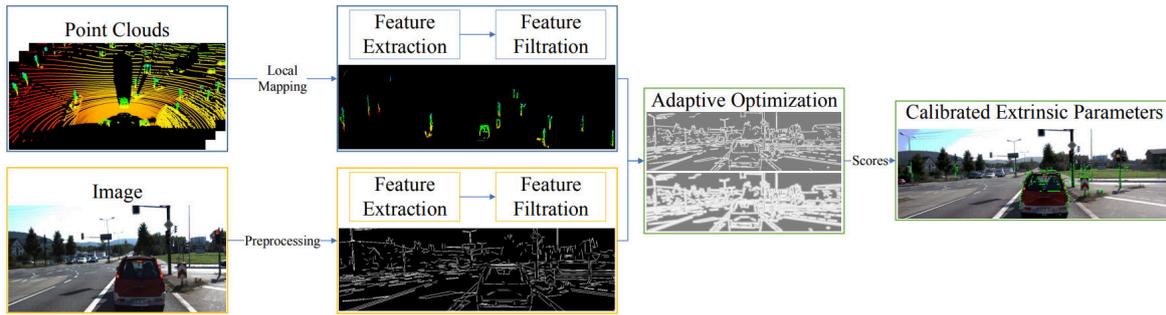
**Figure 3.13:** Semantic segmentation(bottom) instead of Canny edge detection(middle) is used to extract features from images.The Canny's edge detector extract too much "fake" edges(for example:shadow), but semantic based avoid this problem [97].

In order to optimize the extrinsic, a cost function is established to encourage proper projection of the LiDAR points onto the semantic objects. As Fig. 3.14 shows, the cost function will give the highest score of the projection on the left, which indicates the optimal extrinsic.



**Figure 3.14:** Projection of point cloud on the zone of interest. Good projection(left), bad projection(others). The projection on the left will get the highest score [97].

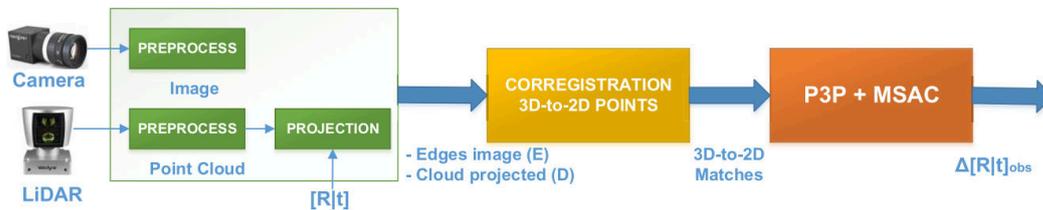
In [94], the authors proposed a **line-based automatic extrinsic calibration method** between the LiDAR and camera, which can be applied in real-world scenes. Initially, the line features are extracted and filtered from point clouds and images. Afterwards, an adaptive optimization is utilized to provide accurate extrinsic parameters. Fig.3.15 shows the calibration pipeline.



**Figure 3.15:** Pipeline for the line-based automatic extrinsic calibration. Preprocessing of the input data(left). Feature extraction(middle). Build correspondence and optimize the extrinsic(right two images) [94].

Another targetless LiDAR to camera calibration method was proposed in [60]. This method consists of a novel co-registration approach: using local edge features in arbitrary environments to get 3D-to-2D errors between the data of camera and LiDAR [60]. Based on the 3D-to-2D errors, the relative transformation (i.e. extrinsic parameters) can be estimated. In order to find the best transform solution, they use the perspective-three-point (P3P[67]) algorithm (Fig. 3.16).

To refine the final calibration, they use a *Kalman Filter* [42], which gives the system high stability against noise disturbances(Fig. 3.17). The presented method does not require any target, or a structured environment. Therefore, it is a target-less calibration approach and robust to changing environments.

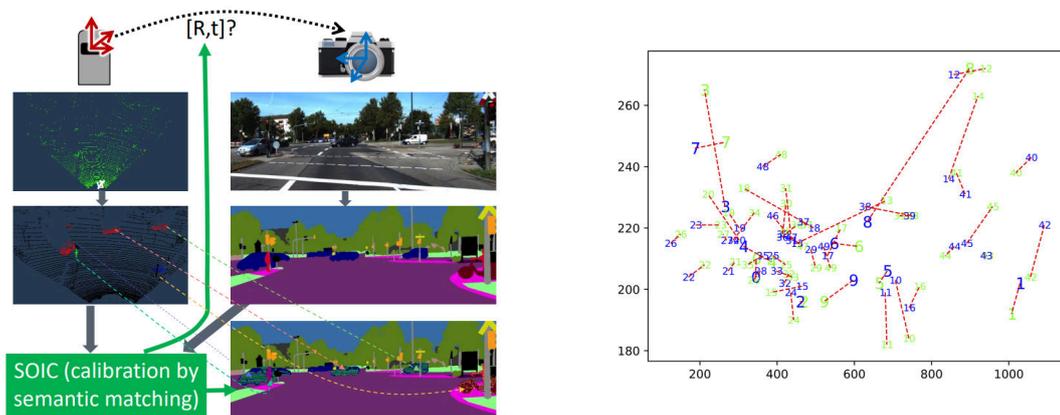


**Figure 3.16:** Calibration pipeline. Preprocessing and the projection of the LiDAR points in the image plane(left). co-register the feature points in LiDAR with points in the image, obtaining the 3D-to-2D matched pair and its errors(middle). Estimating extrinsic parameters using the P3P algorithm and perform inlier detection using RANSAC(right) [29] [60].

In **SOIC** [84](Semantic Online Initialization and Calibration) the authors introduce a semantic based method. They get rid of the dependence on the initial value of the targetless calibration method. As Figure 3.18 shows, *SOIC* removes this limitation by introducing *Semantic Centroids* (SC) to transform the initialization problem into a perspective-n-point (PnP) problem.



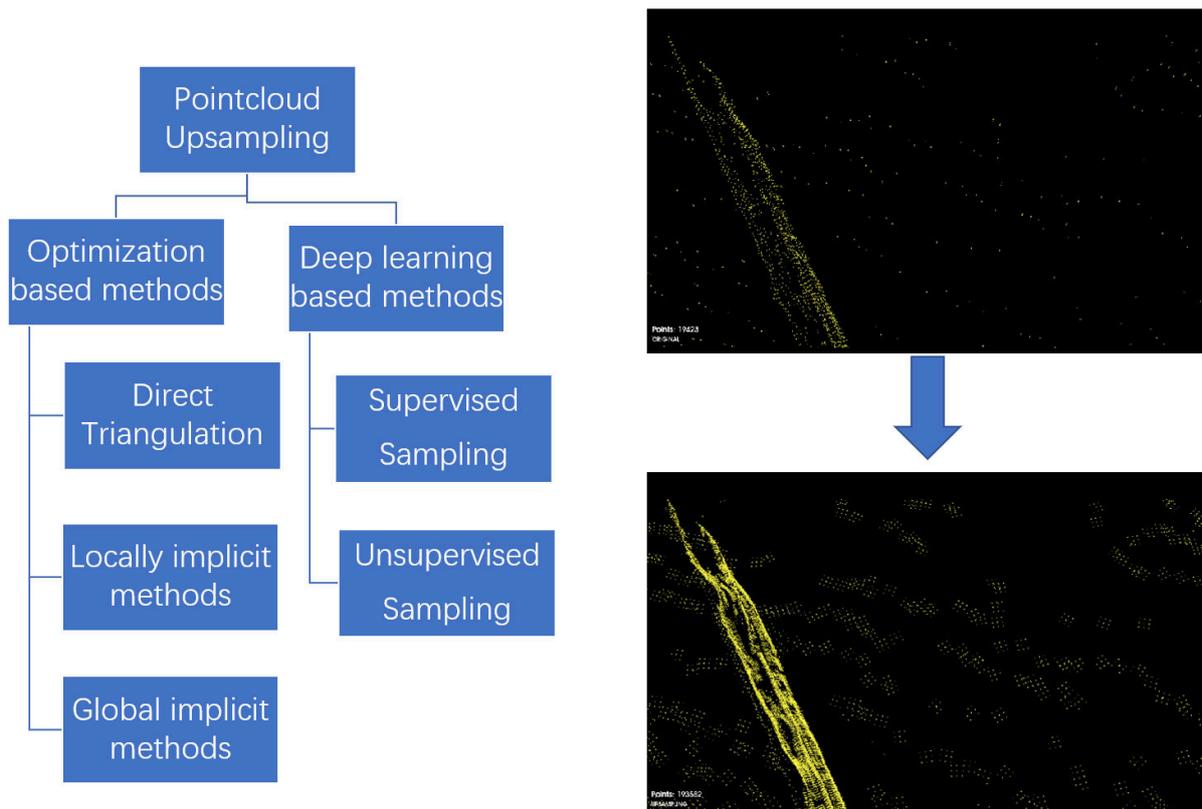
**Figure 3.17:** Whole pipeline including Kalman filter. The blue block stands for the calibration methods in Fig.3.16. Output of this block is taken as observation in a dynamic system and filtered with a Kalman filter (orange block). The yellow block represents the observation’s noise model. The initial guess is the calibration with which the whole process starts [60].



**Figure 3.18:** SOIC estimates the extrinsic between LiDAR and camera based on the semantic matching of point cloud and image(left). Correspondence of semantic centroids with initial parameters(right). Green numbers indicate semantic centroids from images and blue numbers are from semantic centroids of projected point clouds. Red line shows the correspondences [84].

### 3.3 Point Cloud Upsampling

In real-world LiDAR scanning, the state of the point cloud is not always ideal due to the performance limitations of the hardware, the influence of the environment, and the occlusion of the viewing angle. The sparsity of the point cloud and the occlusion of the objects in the scene will lead to the lack of texture, which may cause the mismatch between LiDAR and camera data. Scattering can increase the density of the point cloud to a certain extent to improve the sparsity of the point cloud, but it cannot generate the point cloud of the occluded part. Therefore, we adopt the technique of point cloud upsampling. Fig 3.19 shows the classification of upsampling techniques.

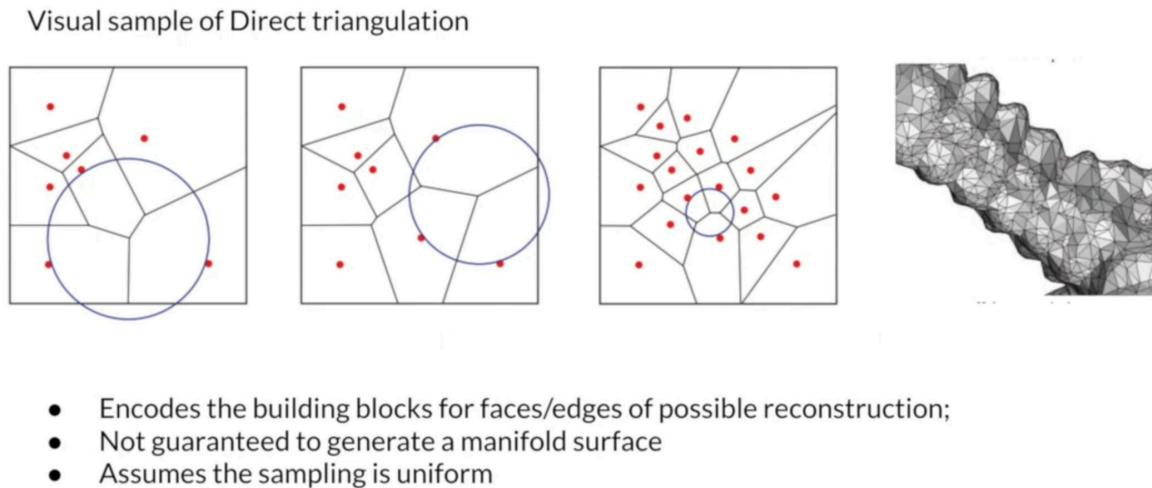


**Figure 3.19:** Upsampling technique. Upsampling can be divided into optimization based methods and deep learning based methods. Optimization based methods can be subdivided into 3 main categories, and deep learning base method can be distinguished by whether it is supervised(left). Figure on the right shows the point cloud before(top) and after(bottom) upsampling. We can see the texture is refined by upsampling.

#### 3.3.1 Optimization based method

Optimization based methods are early approaches for point cloud upsampling. They are mainly used for 3D texture reconstruction. Most classic 3D reconstruction methods are based on direct triangulation. Robert Sitnik et al. [72] introduced a *Mesh3D* algorithm used for triangulation of point clouds obtained from real 3D objects via optical scanning systems. The algorithm is characterized by a high level of automation. The implementation of this kind of a complex assessment system enables us to improve the smoothness and continuity of the mesh, which allows us to obtain more realistic and visually better reconstructions of

objects[72].



**Figure 3.20:** Take any three points on the plane and draw the Voronoi diagram. After obtaining the Voronoi diagram, select the center of the largest circle and project it onto the mean least square surface to obtain an additional point. Repeat the process until the radius of the largest circle is less than the specified threshold (This picture is a screenshot from a youtube video) [4].

The first algorithm for point cloud upsampling was proposed in 2003 [5]. They defined a smooth manifold surface from a set of points close to the surface, upsampling the point set by interpolating the points as vertices of a Voronoi diagram [64] (see Fig. 1.20) on a moving least squares (MLS) surface [5]. Constructing the Voronoi diagram recursively until the threshold is reached.

Subsequently, Lipman et al. introduced the locally optimal projection operator (LOP) for surface approximation from point set data [48]. The operator is parametric free in that it does not rely on estimating local parameters, fitting local planes, or using any other local parametric representation [48]. Therefore, it can handle noisy data that confuses the orientation of points. LOP is proved to be robust to noise and outliers. There are also modifications of LOP. Huang et al. [37] published a weighted locally optimal projection (WLOP), which adds local adaptive density weights to LOP. With the purpose to make the original point cloud distribution more even. Preiner et al. [65] established a WLOP operator based on a Gaussian mixture which describes the input point density, called Continuous LOP (CLOP). Compared with WLOP, CLOP adopts more particles instead of input points, generating better upsampling results [65].

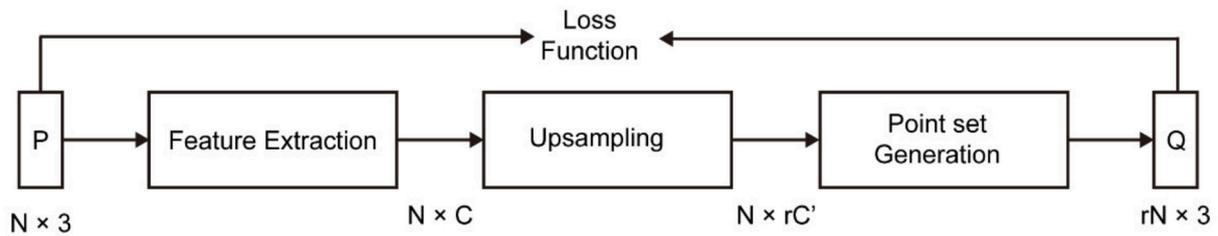
Different from LOP, Dinesh et al. [24] proposed a 3D point cloud super-resolution local algorithm based on the so-called GTV (graph total variation). First, they initialize new points at centroids of local triangles formed using the low-resolution point cloud, and connect all points using a  $k$ -nearest-neighbor graph; then the bipartite graph approximation is conducted to divide all nodes into two detached subsets [24]. They will be optimized until convergence to build correspondences. This algorithm is verified by Stanford 3D scanning repository data. In general, although the optimization-based method can achieve the purpose of upsampling the point cloud to a certain extent, the convergence speed is also relatively fast. But they rely on prior data and thus have significant limitations.

### 3.3.2 Deep Learning-Based Methods

The application of deep learning to point cloud upsampling has gradually become a popular research topic with various achievements in the past decade. Deep learning based point cloud upsampling can be divided into supervised point cloud upsampling and unsupervised point cloud upsampling.

#### Supervised Point Cloud Upsampling

Supervised point cloud upsampling is trained with both low-resolution point clouds (as training set) and corresponding high-resolution point clouds (as label). They are mainly composed of following components, namely feature extraction, upsampling, point set generation and a loss function [95]. A schematic diagram of the network model is shown in Fig. 3.21:



**Figure 3.21:** General workflow for supervised upsampling. Firstly extract feature from input point sets, then do upsampling and generate the new points, finally use the generate point set and the initial input points to get a loss function, continuing optimize the function and upsample the point set until convergence [95].

Yu et al. [90] published the first deep learning model for point cloud upsampling, *PU-Net*. They adopted two feature learning strategies: hierarchical feature learning and multi-level feature aggregation [90].

Zeng et al. [92] proposed a spatial feature extractor block to extract local features. Wang et al. [88] proposed a multi-step point cloud upsampling network (MPU). Point features are extracted from local neighborhoods through a KNN algorithm based on feature similarity [88].

#### Unsupervised Point Cloud Upsampling

Training upsampling networks using supervised learning methods has been widely adopted, but there is a fatal absence in practical applications. Generally speaking, few people in the business use both low-resolution LiDAR and high-resolution LiDAR to collect the same point cloud. Because such behavior is uneconomical. This makes it impossible to obtain downsampled data when using live data for training. This makes unsupervised upsampling methods gradually receive attention.

Liu et al. [61] proposed a new autoencoder, local to global autoencoder (*L2G-AE*). It can be applied to the application of unsupervised point cloud upsampling [61]. But the performance is still not satisfying, because this unsupervised method do not have ground truth label for training.

Even unsupervised methods have a better prospect, but its' performance is still limited.



# Chapter 4

## Manual Calibration

In this chapter, we will introduce the solution and experimental details for manual calibration (including manual calibration of LiDAR to HD map and manual calibration of LiDAR to camera), and show some quantitative results (extrinsic) and qualitative results (visualized results). In Section 4.1 we will introduce the calibration of LiDAR to HD map. In Section 4.2 we will introduce how we manually calibrate a LiDAR to a camera.

### 4.1 LiDAR-HD Map Manual Calibration

In this section we will introduce the calibration of LiDAR to HD map. Section 4.1.1 will briefly introduce the concept of a HD map. Section 4.1.2 will introduce the *OpenDRIVE* [25] (1.4 and 1.6) standard we will adopt for the calibration. Section 4.1.3 will introduce our HD map calibration solution in detail. Section 4.1.4 will show some of our calibration results. In Section 4.1.5 we will discuss the process and results of HD map calibration.

#### 4.1.1 HD Map Structure

The current high-definition map usually refers to the map used for automatic driving assistance, and needs to be compared with traditional navigation maps.

A high-definition map can reach centimeter level accuracy (Atila team achieved less than 3 cm deviation in their test) [17], provide more detailed elements, richer attributes, higher dimensions, and a fast update frequency with high precision [83]. This makes it an electronic map that can assist in achieving more reliable fusion of high-precision positioning functions, providing beyond-the-horizon environment perception capabilities, and providing optimal path planning at the lane level.

#### Comparison With Navigation Maps

The main differences between HD maps and traditional navigation maps are the following:

- Usage: Navigation maps are usually used for human drivers, whereas high-definition maps are used for machines.

- Accuracy: The accuracy of the navigation map has an error above the meter level (5-10 meters) [17], whereas the accuracy of high-definition maps is at the centimeter level.
- Map elements: In high-definition map, road elements and traffic-related dynamic elements are more abundant: such as detailed lane lines, road signs, traffic signs, traffic lights, lane curvature, slope and lane-level real-time traffic dynamic information. It mainly serving the automatic driving environment judgment, decision-making, control, etc.. While navigation map only reach the road level [49] [56].

The difference of normal navigation maps and HD map is shown in Figure 4.1.



**Figure 4.1:** Example of normal navigation map: Here the S110 intersection of the test stretch is shown in google maps. There is only route information in the map [34](left figure). As a comparison, the same area is presented in the HD map with much more detailed features, including buildings, traffic signs, lane marks etc. (right figure).

### Usage of HD Maps

HD maps can provide a full range of assistance for autonomous driving applications:

- Positioning: Combined with high-precision map and point cloud maps, the vehicle can achieve visual positioning and point cloud positioning [83].
- Perception: By providing road information and other prior knowledge to assist vehicle sensors to narrow the recognition range, improve recognition efficiency, reduce error rates. And at the same time make up for other sensor perception defects in certain environments (such as other sensor failures in rain, snow, and darkness) [17].
- Planning and prediction: High-precision maps can combine navigation planning, real-time road conditions, automatic driving function levels, and suitable automatic driving routes to help vehicles perform lane-level path planning, and predict road conditions and obstacles ahead based on prior knowledge and real-time road conditions [56].
- Safety: Based on the known road information on the map, such as known obstacles, road signs and other characteristic information; it can detect and match with sensors in real-time, improve redundancy and reduce misjudgment [49].

### HD Map Categories

At present, there is no unified opinion on the definition, content and layering of HD maps. A classification method for HD map is: vector map, feature map and point cloud map:

- Vector Map: Contains lane model, road components (objects) and road attributes.

- Feature Map: Is a feature vector layer that can support vehicle visual positioning extraction.
- Point Cloud Map: Is built up of several scanning LiDAR point cloud layers.

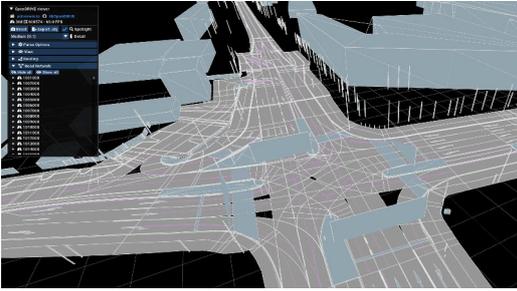


Figure 4.2: Vector map

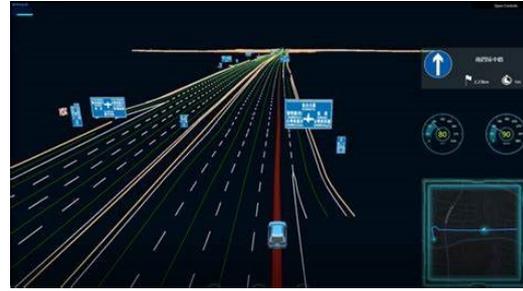


Figure 4.3: Feature map[26]

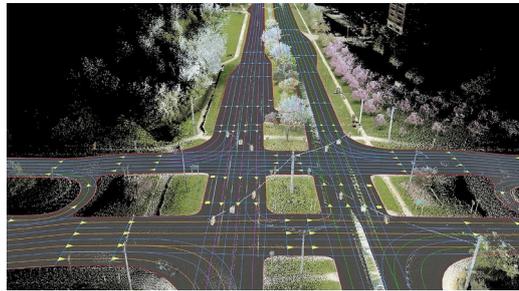


Figure 4.4: Point cloud map[83]

**Figure 4.5:** A vector map only contains the lane model, road components (objects) and other geometric features on the road. Feature Map represent those objects with more detailed textures and can be used for image recognition. A LiDAR point cloud map is made of 3D points perceived by LiDAR sensors.

Our HD map calibration will only use a vector map.

### 4.1.2 OpenDRIVE

*OpenDRIVE* is an international common standard formulated by ASAM, a German organisation. The *OpenDRIVE* format is based on the *Extensible Markup Language (XML)*, and the file suffix is in *.xodr* format, which is a general standard for describing roads and road networks. The data stored in *OpenDRIVE* files describes the geometry of the road and the features along it and defines the traffic signs that can affect the traffic logic as well as road infrastructure such as lanes and signal lights[25].

The road information is described in the *.xodr* file, which is built based on experience, and can also be generated based on real road data. The main purpose of *OpenDRIVE* is to provide a road network description that can be used for simulation, and to enable these roads and road network descriptions to be customized or changed in the simulation platform or simulation software[25].

*OpenDRIVE* describes various information in the road with nodes and elements according to the *XML* format. Such a common format facilitates a high degree of specialization for virtual simulation testing and maintains the interoperability required for data exchange between different countries.

## OpenDRIVE Structure

```

<object type="tree" name="tree" id="4056144" s="1.7026e+03" t="-4.0361e+01" zOffset="-5.387e-01" validLength="0.00" orienta
<object type="tree" name="tree" id="4056145" s="1.7176e+03" t="-3.4718e+01" zOffset="-5.644e-01" validLength="0.00" orienta
<object type="building" name="tollBooth" id="4056146" s="2.8276621279e+02" t="-2.3704908987e+01" zOffset="-0.4285" orientat
  <outline>
    <cornerLocal u="0.000000000e+00" v="0.000000000e+00" z="0.000000000e+00" height="1.2837047237e+00" />
    <cornerLocal u="3.6312009745e+00" v="-5.0815746188e-02" z="1.0774000000e-02" height="1.2602306427e+00" />
    <cornerLocal u="3.6124835318e+00" v="-2.1648569573e+00" z="1.0718000000e-02" height="1.3133864075e+00" />
    <cornerLocal u="-8.7459231145e-02" v="-2.1454425501e+00" z="-2.5699999998e-04" height="1.2809614649e+00" />
    <cornerLocal u="0.000000000e+00" v="0.000000000e+00" z="0.000000000e+00" height="1.2837047237e+00" />
  </outline>
</object>
<object type="gantry" name="gantry" id="4056147" s="2.9169303711e+02" t="4.0209471073e+00" zOffset="0.00">
  <outline>
    <cornerRoad s="2.9169303711e+02" t="4.0209471073e+00" dz="-4.2038590911e-01" height="0.0" />
    <cornerRoad s="2.9176139418e+02" t="4.0156112605e+00" dz="5.1911161624e+00" height="0.0" />
    <cornerRoad s="2.9180642315e+02" t="-8.7908609806e+00" dz="5.1866521146e+00" height="0.0" />
    <cornerRoad s="2.9184275463e+02" t="-1.9123721287e+01" dz="5.1830116453e+00" height="0.0" />
    <cornerRoad s="2.9192715165e+02" t="-1.9114584406e+01" dz="-9.1274923651e-01" height="0.0" />
  </outline>
</object>
<object type="barrier" name="wall" id="4056148" s="3.1701149966e+02" t="-1.9283941451e+01" zOffset="-0.3880" orientation="n
  <outline>
    <cornerLocal u="0.000000000e+00" v="0.000000000e+00" z="0.000000000e+00" height="2.5924670114e+00" />
    <cornerLocal u="-1.3759292348e-01" v="-2.6037341729e-01" z="-4.3500000004e-04" height="2.5356842680e+00" />
    <cornerLocal u="-6.1397528707e-01" v="-2.2301884368e-02" z="-1.9540000000e-03" height="2.5994102558e+00" />
    <cornerLocal u="-1.3004941157e+01" v="-8.5708373412e-01" z="-4.0724000000e-02" height="2.6509942693e+00" />
    <cornerLocal u="-1.3024528661e+01" v="-5.6531136204e-01" z="-4.0784000000e-02" height="2.7180259985e+00" />
    <cornerLocal u="-5.5042013584e-01" v="2.7507182583e-01" z="-1.7510000000e-03" height="2.6660930037e+00" />
    <cornerLocal u="0.000000000e+00" v="0.000000000e+00" z="0.000000000e+00" height="2.5924670114e+00" />
  </outline>

```

**Figure 4.6:** Example of .xodr file under standard of OpenDRIVE 1.4. As highlighted in the figure, the object "gantry" is stored as an "object" type and represent by a set of features in floating point number.

As figure 4.6 shows, *OpenDRIVE* data is stored in XML files with the .xodr extension. The *OpenDRIVE* file structure complies with XML rules. Elements are arranged by rank in the XML format, and elements with a rank greater than zero (0) are child elements. Elements with level (1) are called master elements. Each element can be extended with user-defined data. Each *OpenDRIVE* file will have a main element <OpenDRIVE>, and all feature classes describing roads are its child elements.

All floating point numbers used in *OpenDRIVE* are IEEE 754 double precision floating point numbers; in order to ensure the accurate representation of floating-point numbers in XML, floating-point numbers in XML should use a known correct precision[63]. Generally, 17 significant figures are reserved to describe numbers[63]. All attributes available in *OpenDRIVE* files are fully annotated in the UML model:

- Unit: Unit of e.g. road length or speed.
- Type: Describes the data type of an attribute, which can be a primitive data type, such as string, double, float, or a complex data type referring to an object.
- Values: Values determine the range of values for a given property, for example:

```

<geometry
s="4.9957524872074799e+02"
x="4.9469346060416666e+02"
y="5.3447643627860181e+01"

```

```

hdg="5.8804473418180125e-02"
length="6.2079164697363019e+01">
<line/>
</geometry>

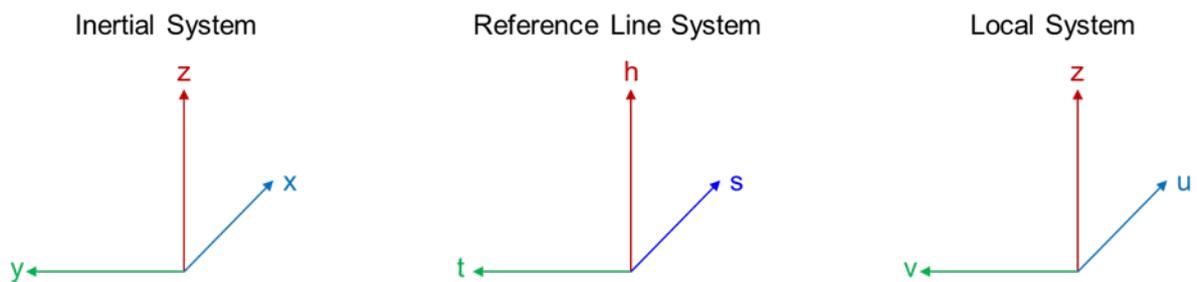
```

Among them, geometry represents the road unit to be described by the current element, and the attributes of geometry include s, x, y, hdg and length, and their values follow.

### OpenDRIVE Coordinate Systems

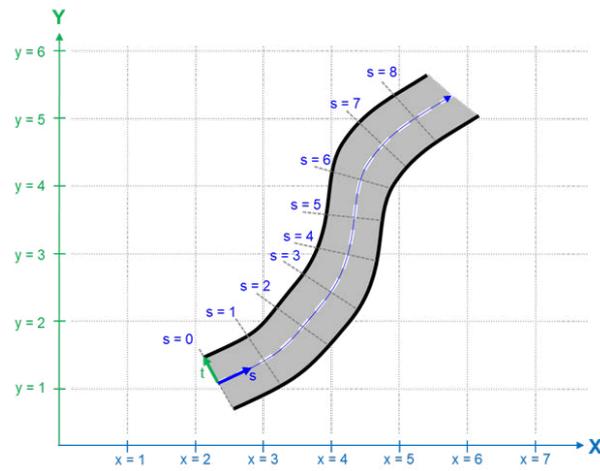
OpenDRIVE uses three types of coordinate systems, as presented in Figure 4.7:

- The inertial  $[x,y,z]$  coordinate system, also called Cartesian system (global)
- The reference line  $[s,t,h]$  coordinate system
- The local  $[u,v,z]$  coordinate system

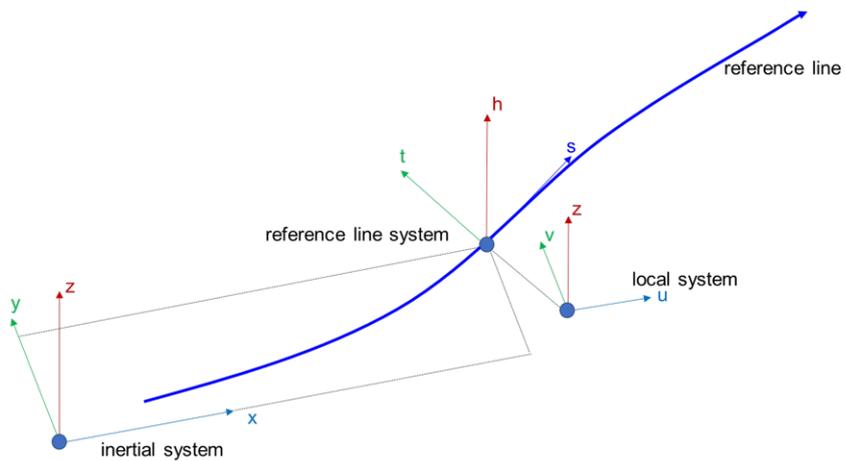


**Figure 4.7:** Inertial system (left), reference line system, which is based on a start point and a line (middle), local Cartesian system (right).

The reference line coordinate system applies to reference lines along corridors. Figure 4.8 shows how a reference line system is built. The relationship among coordinate systems is described in Figure 4.9.



**Figure 4.8:** Reference line coordinate system is a right-handed coordinate system. The  $s$  direction is along the tangent of the reference line and  $t$  direction is orthogonal to the  $s$  direction. Then  $h$  is defined as the up direction orthogonal to the  $x$ - and  $y$ -axes.[63]



**Figure 4.9:** Relationship of coordinate systems: The origin of a local system is recorded in the reference line system. And the reference line system is built based on a reference line in the global inertial system [63].

## OpenDRIVE Objects

In *OpenDRIVE*, objects are represented by the `<objects>` element within the `<object>` element. Objects are defined per `<road>` element. Each object establishes a local coordinate system based on the starting point of its road as the origin, and is represented by a reference line coordinate system, as presented in figure 4.10 below.

Therefore, to calibrate the LiDAR point cloud to a HD map, it is first necessary to convert the

```
<object type="gantry" name="gantry" id="4056147" s="2.9169303711e+02" t="4.0209471073e+00" zOffset="0.00">
  <outline>
    <cornerRoad s="2.9169303711e+02" t="4.0209471073e+00" dz="-4.2038590911e-01" height="0.0" />
    <cornerRoad s="2.9176139418e+02" t="4.0156112605e+00" dz="5.1911161624e+00" height="0.0" />
    <cornerRoad s="2.9180642315e+02" t="-8.7908609806e+00" dz="5.1866521146e+00" height="0.0" />
    <cornerRoad s="2.9184275463e+02" t="-1.9123721287e+01" dz="5.1830116453e+00" height="0.0" />
    <cornerRoad s="2.9192715165e+02" t="-1.9114584406e+01" dz="-9.1274923651e-01" height="0.0" />
  </outline>
</object>
```

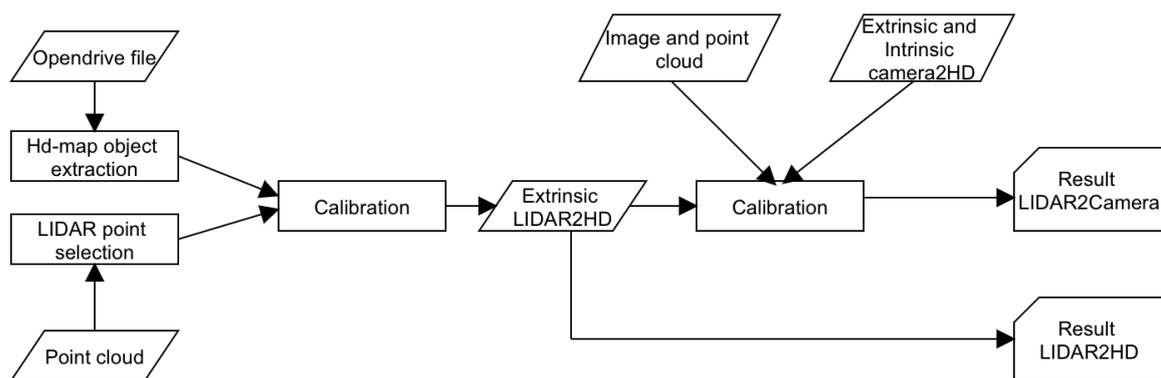
**Figure 4.10:** Gantry in S110 intersection on the A9 highway. The gantry is stored as object and described by a set of key points. Here five key points are used for the description, each point is represented with a reference line system in the format of  $[s, t, dz(h)]$

objects represented by the  $st$  local coordinate system to the global  $xyz$  coordinate system of the HD map. This step will make use of the work of our colleague Marcel Bruckner [10].

### 4.1.3 Solution

In this section we will introduce the HD map manual calibration pipeline in detail.

#### HD Map Manual Calibration Pipeline



**Figure 4.11:** Pipeline for HD map calibration. First specify the points used for calibration, and extract them from the `.odr` file [10] and also pick the corresponding LiDAR points in point clouds. Then perform SVD [6] based calibration to get the extrinsic. For visualization, project onto the image using the extrinsic and intrinsic parameter of camera-HD map calibration from early work of our team.

As shown in Fig. 4.11, LiDAR HD map manual calibration consists of three main steps. First obtain the coordinates of the corresponding points in the corresponding LiDAR and HD map in their respective coordinate systems, and then obtain extrinsic results through SVD

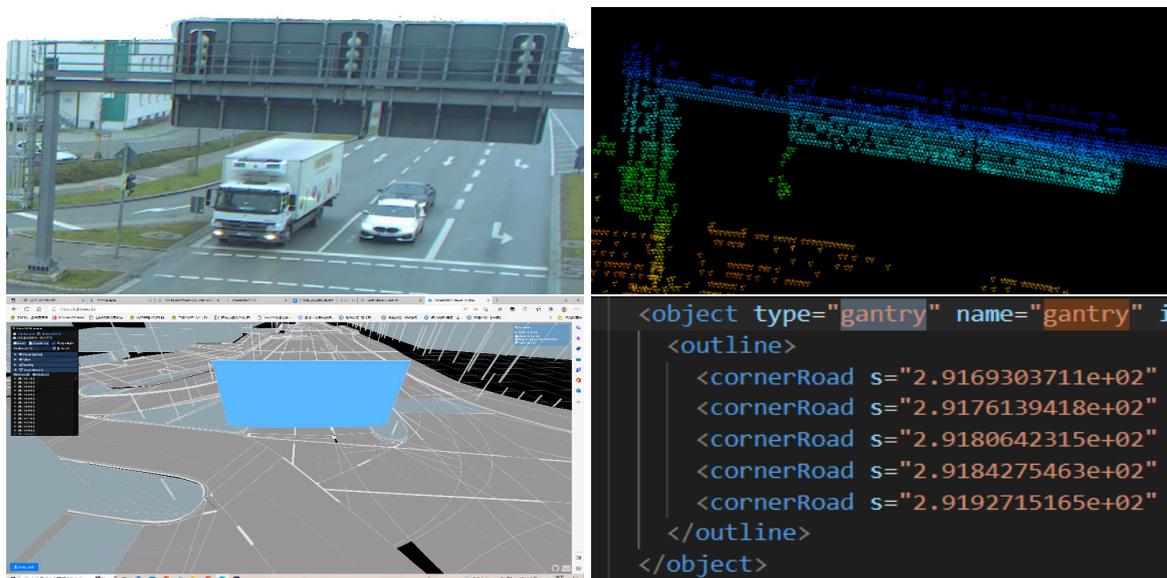
(singular value decomposition) calibration method, and finally use the obtained coordinates of the camera in the HD Map and the intrinsic[98] to project point cloud to the 2D image.

### Point Extraction

The calibration of LiDAR to HD map is essentially a 3D-3D calibration, and all manual methods for 3D point cloud registration apply here. For the automatic calibration method, due to the structure of the *OpenDRIVE* file, it is not applicable.

As shown in Fig. 4.12, the gantry with rich features in reality is only represented by a rectangle in the HD map. To make matters worse the bottom of the rectangle is not the bottom of the gantry board, but the bottom of the stand. This representation brings up two problems:

1. In the *.xodr* file of the HD map, different objects are represented in various ways. For example, the tree has only two points (bottom, top) and the height is different from the actual situation. However the gantry shown in the figure below is represented by five key points. Lane marks are even not classified as objects. The variety of formats makes it impossible to use automatic methods to extract *.xodr* files, because each object needs to be manually screened and corrected.
2. The sparsity of object representations in HD maps does not match the dense point clouds in LiDAR, which makes automatic calibration algorithms unsuitable for this calibration. Therefore, the 3D keypoints (corners) in the point cloud must be selected manually.



**Figure 4.12:** Gantry in real world (up left), gantry in point cloud (up right), gantry in *OpenDRIVE* viewer is represented only as a square (bottom left), gantry in *OpenDRIVE* file (bottom right).

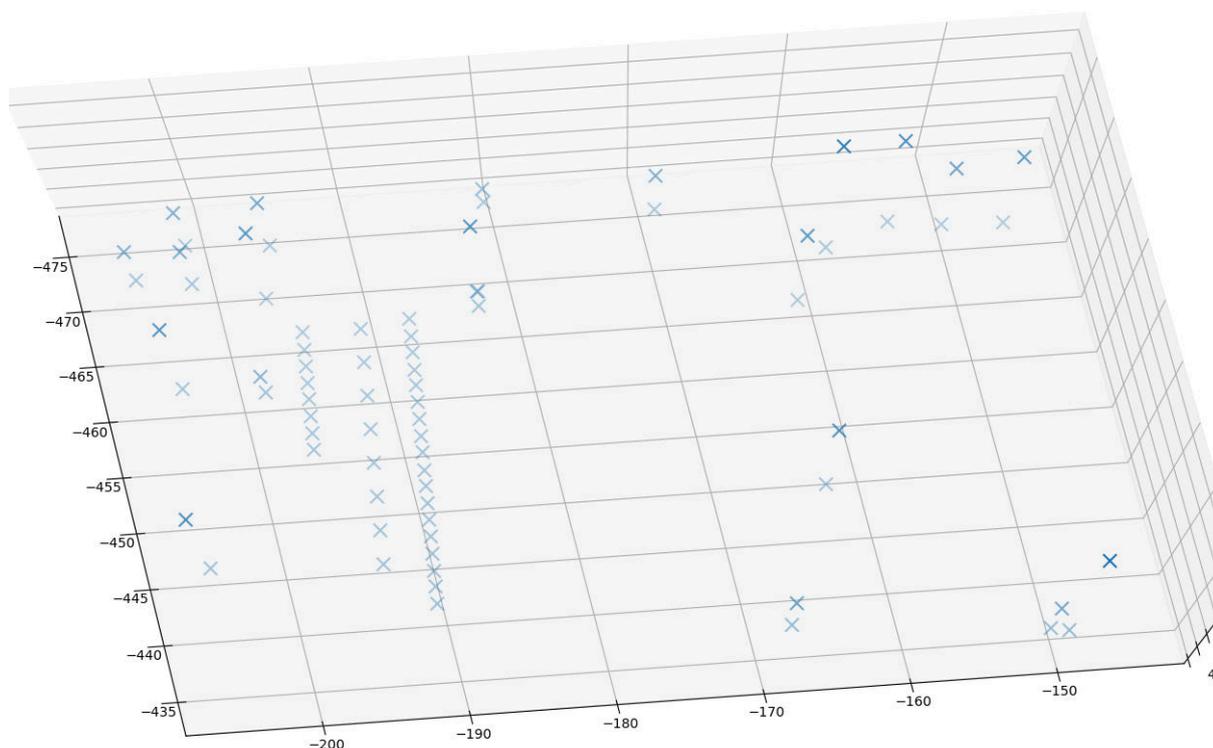
For point clouds, we manually pick keypoints using *pcl viewer*. For the HD map, we use the *xodr viewer* to get the object ID we need. Using the work from Bruckner[10], the objects in the *.xodr* file are transformed into the global coordinate system. Finally, the corresponding object is found through the ID, and its Cartesian coordinates relative to the HD map origin are obtained.

Since the lanemark does not belong to the object, in order to calibrate the ground features, we use a little trick. We change the lanemark type within the `.xodr` file to object and give it a special label as figure 4.13 shows.

Finally we get the points as the figure 4.14 represented.

```
<objects>
<object type="line1" name="line1" id="l1001" s="22.617370362" t="1.5" zoffset="-6.9987104520e-01" />
<object type="line1" name="line1" id="l1002" s="24.204210196" t="1.5" zoffset="-6.9987104520e-01" />
<object type="line1" name="line1" id="l1003" s="25.625711752" t="1.5" zoffset="-6.9987104520e-01" />
<object type="line1" name="line1" id="l1004" s="27.214659501" t="1.5" zoffset="-6.9987104520e-01" />
<object type="line1" name="line1" id="l1005" s="28.578288879" t="1.5" zoffset="-6.9987104520e-01" />
<object type="line1" name="line1" id="l1006" s="30.084425679" t="1.5" zoffset="-6.9987104520e-01" />
<object type="line1" name="line1" id="l1007" s="31.606220968" t="1.5" zoffset="-6.9987104520e-01" />
<object type="line1" name="line1" id="l1008" s="33.142563032" t="1.5" zoffset="-6.9987104520e-01" />
<object type="line1" name="line1" id="l1009" s="34.572700513" t="1.5" zoffset="-6.9987104520e-01" />
<object type="line1" name="line1" id="l1010" s="36.187237246" t="1.5" zoffset="-6.9987104520e-01" />
<object type="line1" name="line1" id="l1011" s="37.57604557" t="1.5" zoffset="-6.9987104520e-01" />
<object type="line1" name="line1" id="l1012" s="39.106170696" t="1.5" zoffset="-6.9987104520e-01" />
```

**Figure 4.13:** Lane marks transformed to objects, and each key point on lane marks is given a special index for parsing.



**Figure 4.14:** Points extracted from the HD map and visualized by *pyplot* (represented in HD map coordinates).

## Calibration

After the corresponding points are obtained manually, the point pairs can be calibrated to get the extrinsic parameter. We adopted the *SVD* method to calibrate the acquired point pairs. The reason is that the origin of the HD map is very far away from the LiDAR, and there is a problem with the accuracy of the initial extrinsic [98]: the initial extrinsic are calculated based on a false data about coordinate of s110 base station, it was about 2 meters and 10 degrees

deviated from the ground truth. On the basis of manually establishing the corresponding relationship, *SVD* can quickly obtain accurate and reliable calibration results without initial values. Our method for *SVD* calibration refers to [6].

### Projection

After obtaining the extrinsic information from LiDAR to HD map, we can use the already available extrinsic from camera to HD map to calculate the extrinsic from LiDAR to camera. This result can be used to project the point cloud to the image to obtain intuitive qualitative results, and can also be used as an initial extrinsic for automatic calibration. The projection result are shown in section 4.1.4

### 4.1.4 Results

In this section we will present some qualitative and quantitative results.

#### Problem statement

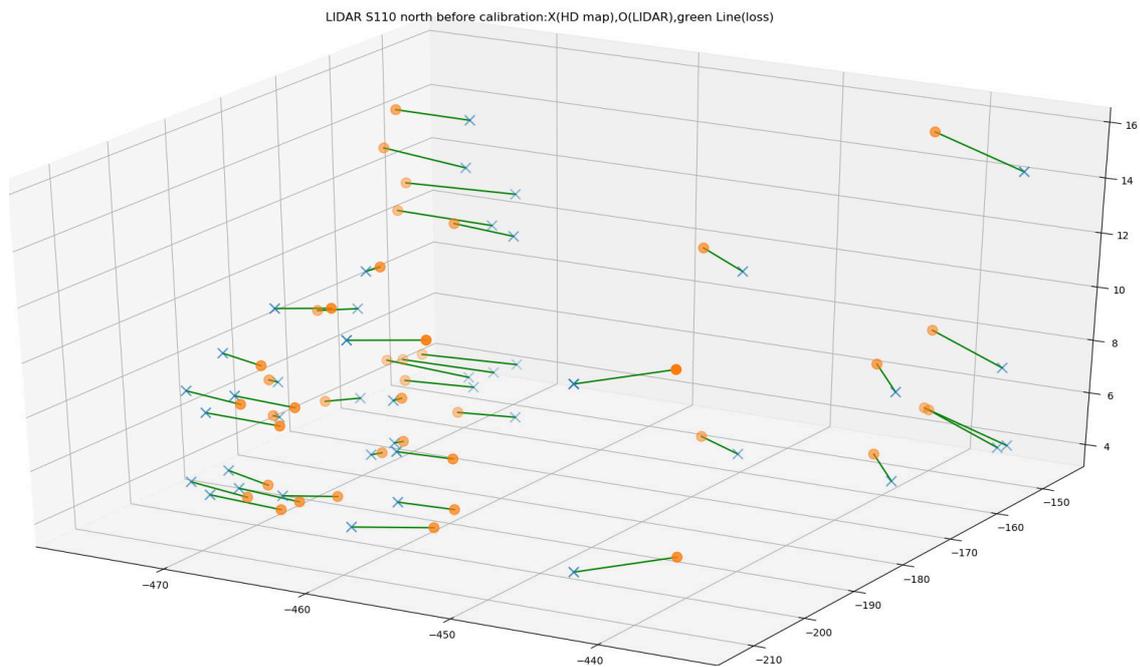
We will calibrate the two LiDARs (S110 north and S110 south) located at the S110 intersection of the Test Stretch to the HD map coordinate system. And use the available intrinsic of the S110 south2 camera and its extrinsic to get the projection result. Figure 4.15 shows the origin of HD map in A9 dataset.



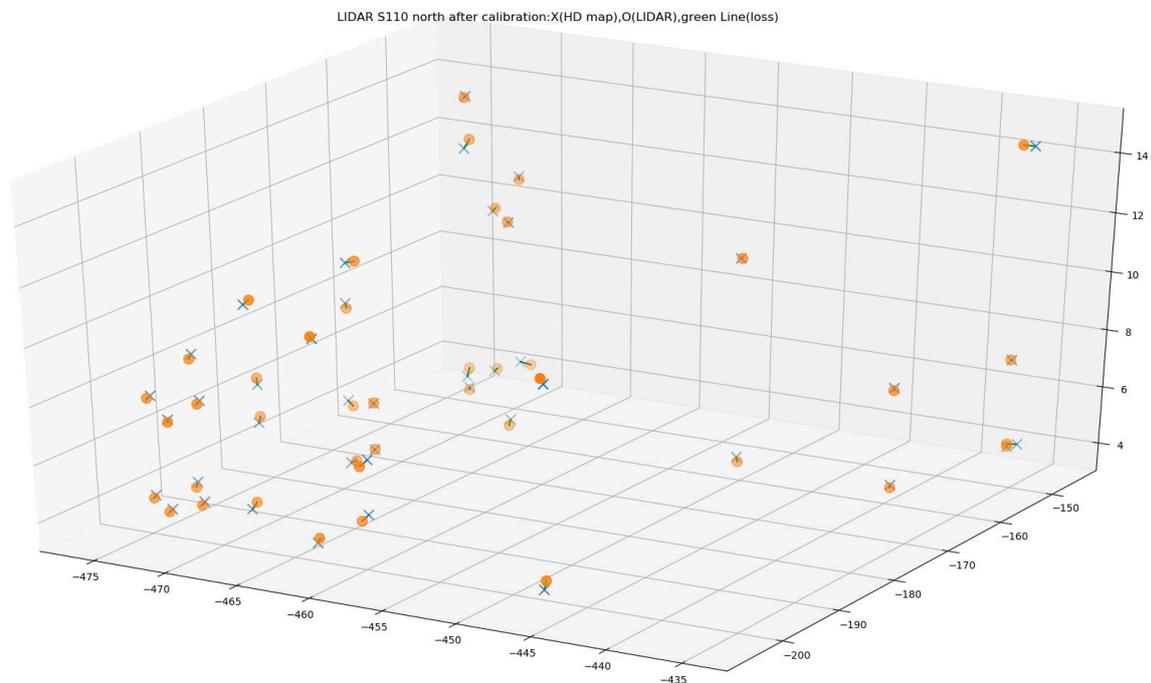
**Figure 4.15:** Overview of the origin of the HD map of the A9 dataset. A red square represents the origin of the HD map, a green square represent the origin of S110 sensor base station.

#### Results - S110 North LiDAR to HD Map Calibration

The calibration result of S110 north to HD map is visualized in Fig. 4.17.



**Figure 4.16:** LiDAR S110 north before calibration. A cross represent points in the HD map, a circle represents LiDAR points which are transformed into the HD Map coordinate system. A green line represent the residual error. The extrinsic used the false initial extrinsic in [98].

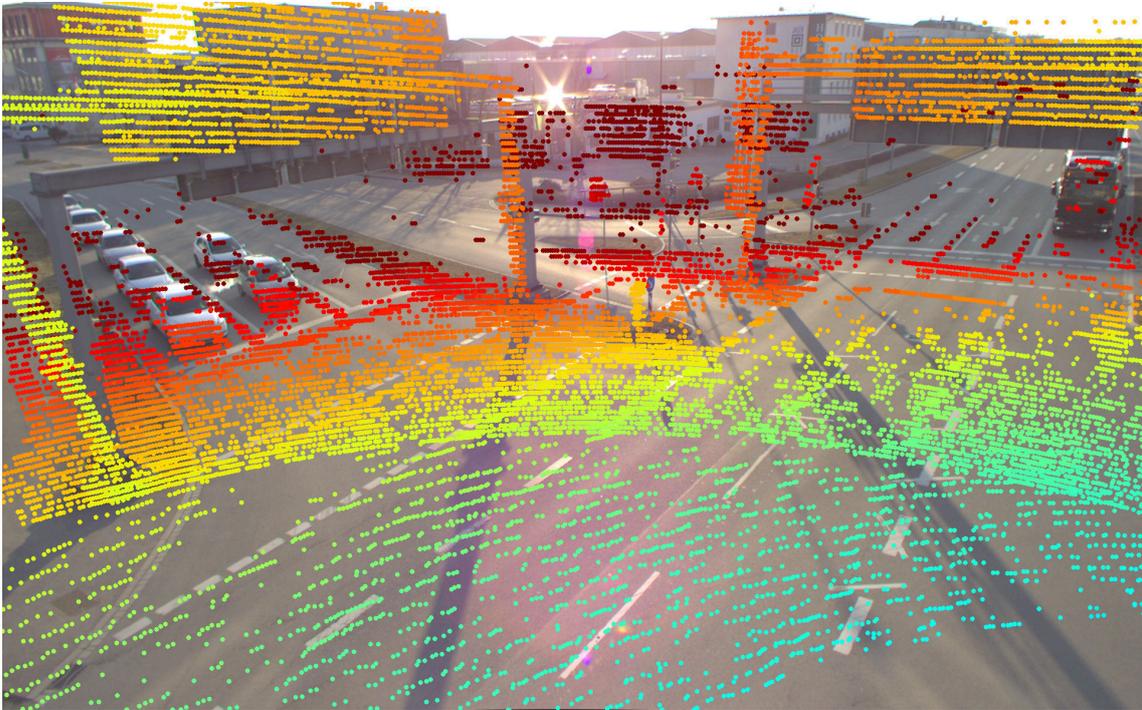


**Figure 4.17:** LiDAR S110 north after calibration. A cross represent points in the HD map, a circle represents LiDAR points which are transformed into the HD Map coordinate system. A green line represent the residual error. After our calibration the points are converged and errors (green line) are reduced.

The extrinsic of S110 north LiDAR into HD map is listed in Eq. 4.1.

$$C = \begin{bmatrix} -0.99748532 & 0.06865866 & -0.01757912 & -434.74851344 \\ -0.06910888 & -0.99725893 & 0.02643077 & -172.41941861 \\ -0.01571624 & 0.02757918 & 0.99949607 & 15.60420132 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.1)$$

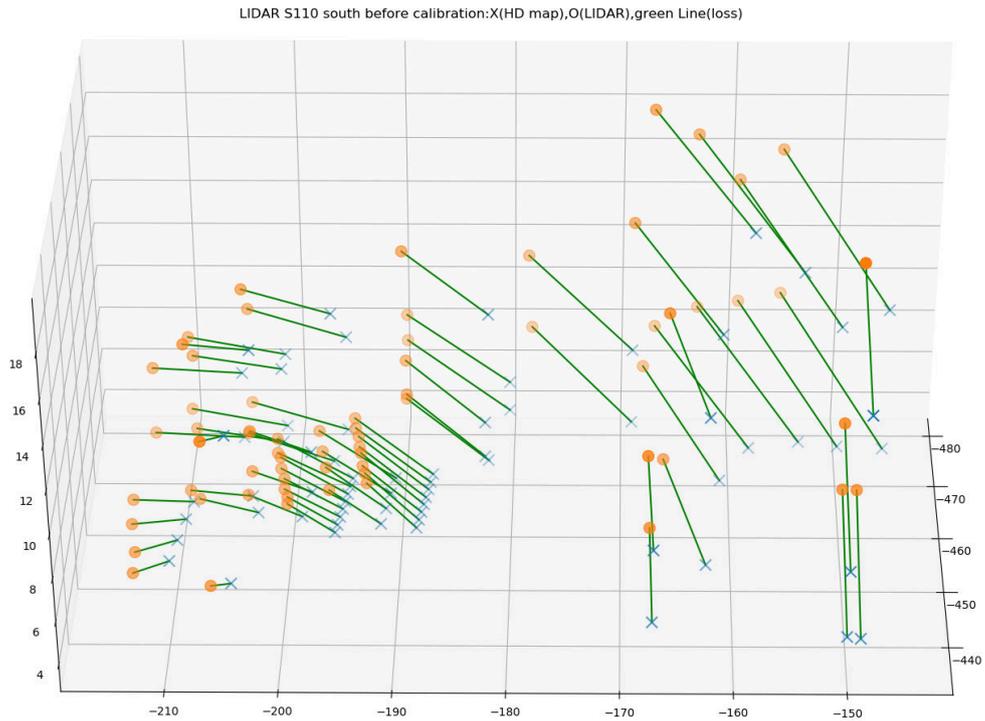
The *RMSE* loss is 0.418. The projection result is visualized in Fig. 4.18.



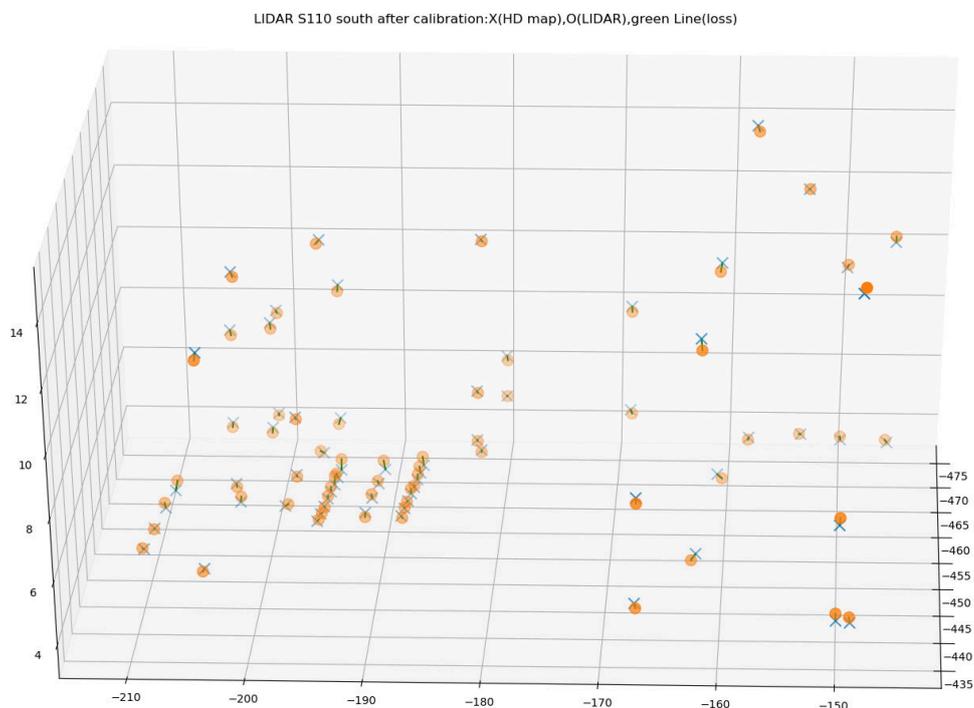
**Figure 4.18:** Projection of S110 north LiDAR point cloud into south2 camera. Most objects on the ground (lane markings) are converged, except for the gantry. This is because the intrinsic and extrinsic (from camera to HD map origin) [98] used for projection is optimized together. Hence, there is a mismatch between the intrinsic and our calibration result of LiDAR to HD map which leads to the distortion of the gantry bridge.

### Results - S110 South LiDAR to HD Map Calibration

The calibration result of S110 south to HD map is visualized in Fig. 4.20. The extrinsic of S110 south LiDAR into HD map is listed in Eq. 4.2.



**Figure 4.19:** LiDAR S110 south before calibration. A cross represent points in the HD map, a circle represents LiDAR points which are transformed into the HD Map coordinate system. A green line represent the residual error. The extrinsic used the false initial extrinsic in[98].

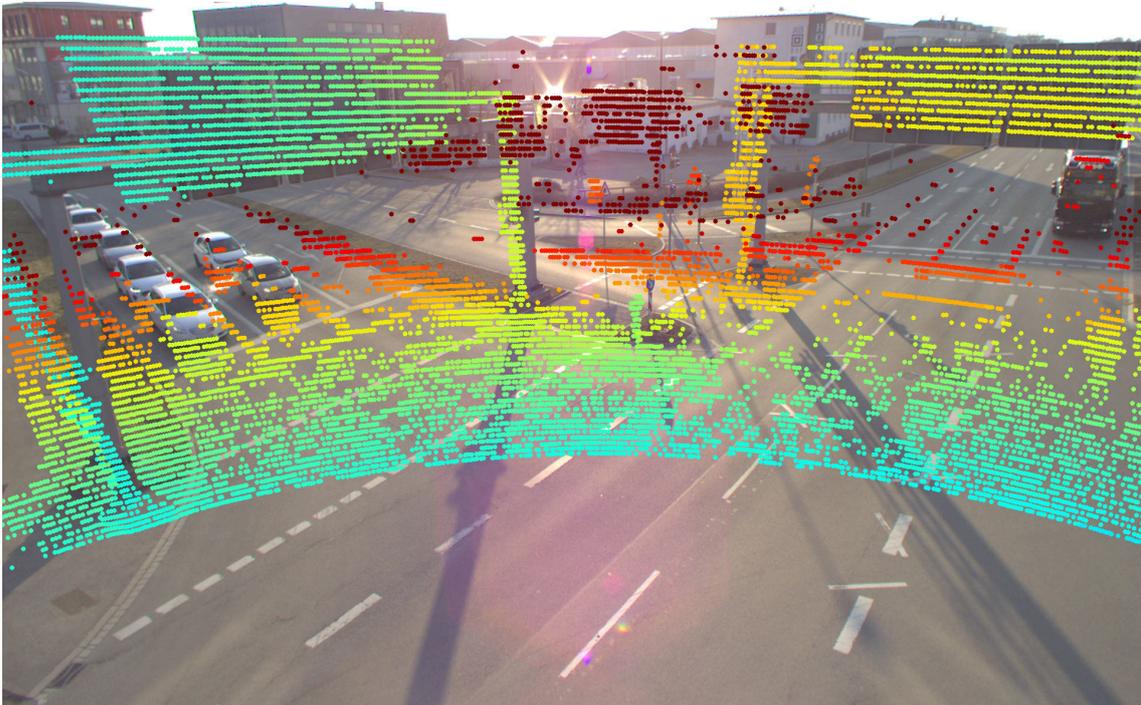


**Figure 4.20:** LiDAR S110 south after calibration. A cross represent points in the HD map, a circle represents LiDAR points which are transformed into the HD Map coordinate system. A green line represents the residual error. After our calibration the points are converged and errors (green line) are reduced.

$$C = \begin{bmatrix} -9.77802371e-01 & -2.09527130e-01 & 9.50926026e-04 & -436.15046201 \\ 2.09480651e-01 & -9.77467546e-01 & 2.59817903e-02 & -186.04099271 \\ -4.51439064e-03 & 2.56042568e-02 & 9.99661964e-01 & 14.50492285 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(4.2)

The *RMSE* loss is 0.309. The projection result is visualized in Fig. 4.21:



**Figure 4.21:** Projection of S110 south LiDAR to south2 camera. Most objects are converged except for the left gantry. This is because the intrinsic and extrinsic (from camera to HD map origin) [98] used for projection is optimized together. Hence, there is a mismatch between the intrinsic and our calibration result of LiDAR to HD map which leads to the distortion of the gantry bridge.

#### 4.1.5 Short discussion

After completing HD map calibration, we can draw two important conclusions: 1. The representation of objects in *OpenDRIVE* caused a lot of trouble for calibration. Because the reference line coordinate system and keypoint are used to represent obstacles, it is impossible to use an automated method to extract the coordinates matched with the point cloud.

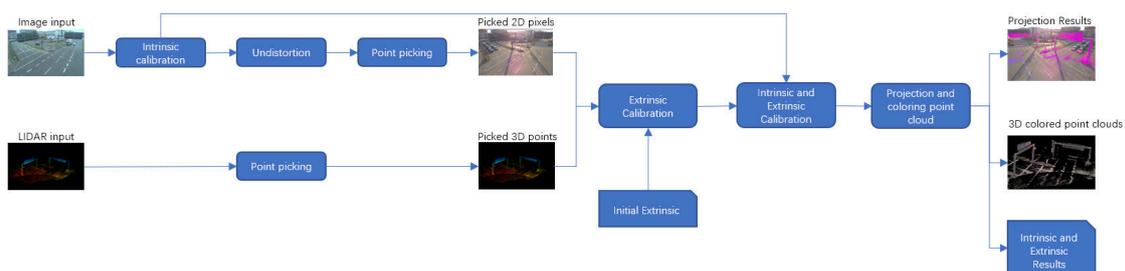
A possible solution is to use high-definition maps containing LiDAR point clouds instead of vector maps.

2. Comparing the HD map calibration results and projection results, it can be found that the performance of projection to 2D image is not satisfactory. The main reasons are that the calibration data from camera to HD map (especially the intrinsic) has been specially optimized, so that this intrinsic is not equivalent to the real intrinsic, causing distortion in the projection result.

## 4.2 LiDAR-Camera Manual Calibration

In this section we will show how we manually calibrate LiDAR to camera. In Section 4.2.1 we will briefly introduce the methods used in calibration and the calibration pipeline. In Section 4.2.2 we will show some calibration results. Finally, we discuss the results in Section 4.2.3.

### 4.2.1 Manual Calibration Pipeline



**Figure 4.22:** Manual Calibration Pipeline. We first start with manual point picking. We use PCL viewer to pick points in the point cloud. In the mean while, we undistort the image using the intrinsic and pick corresponding 2D points in the image. After we got the 2D-3D point pairs, we can perform *PnP* based manual calibration. Finally, we visualize the results through projection and point cloud coloring.

As shown in Fig. 4.22 above, manual LiDAR to camera calibration consists of multiple steps. First, we perform intrinsic calibration on the camera to obtain its intrinsic parameters. Then we use the obtained intrinsic to undistort the original image. Then comes the most important step, which is to establish the correspondences by manually picking key points in the camera image and the point cloud. Afterwards, *PnP*-based extrinsic calibration [9] was performed to obtain the extrinsic parameters. After that, synchronous optimization of intrinsic and extrinsic is performed to obtain the optimal solution. Finally, the obtained parameters are used for projection and point cloud coloring to obtain qualitative results.

### 4.2.2 Results

In this section we will present some qualitative and quantitative results.

### Results - S110 North LiDAR to S110 south1 Camera Calibration

In this scene we extract 44 point pairs:

The extrinsic of S110 north LiDAR into south1 camera is listed in Eq. 4.3.



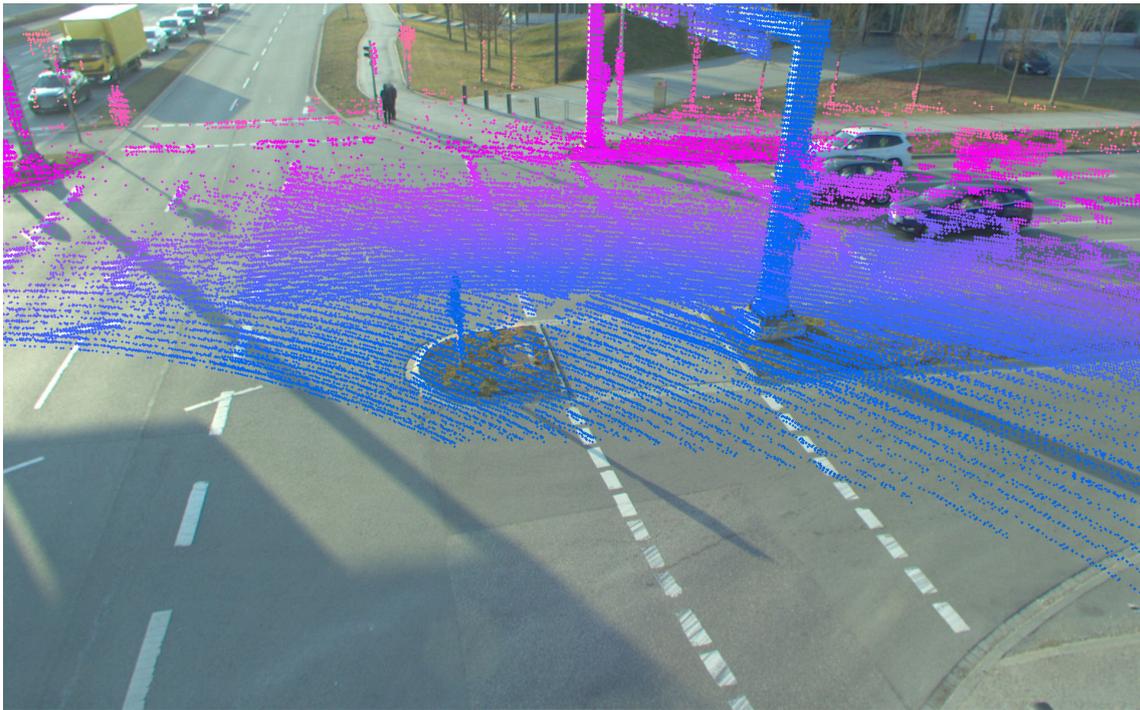
**Figure 4.23:** Here we extract 44 points in the camera image (s110\_camera\_basler\_south1\_8mm) and the LiDAR (s110\_lidar\_ouster\_north). We mainly focus on the corners of objects that are easy to be identified in the point cloud, including traffic signs, lane markings, gantry bridge etc..

$$C = \begin{bmatrix} -0.374855 & -0.926815 & 0.0222604 & -0.284537 \\ -0.374855 & -0.926815 & 0.0222604 & -0.284537 \\ 0.8017 & -0.336123 & -0.494264 & -0.837352 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.3)$$

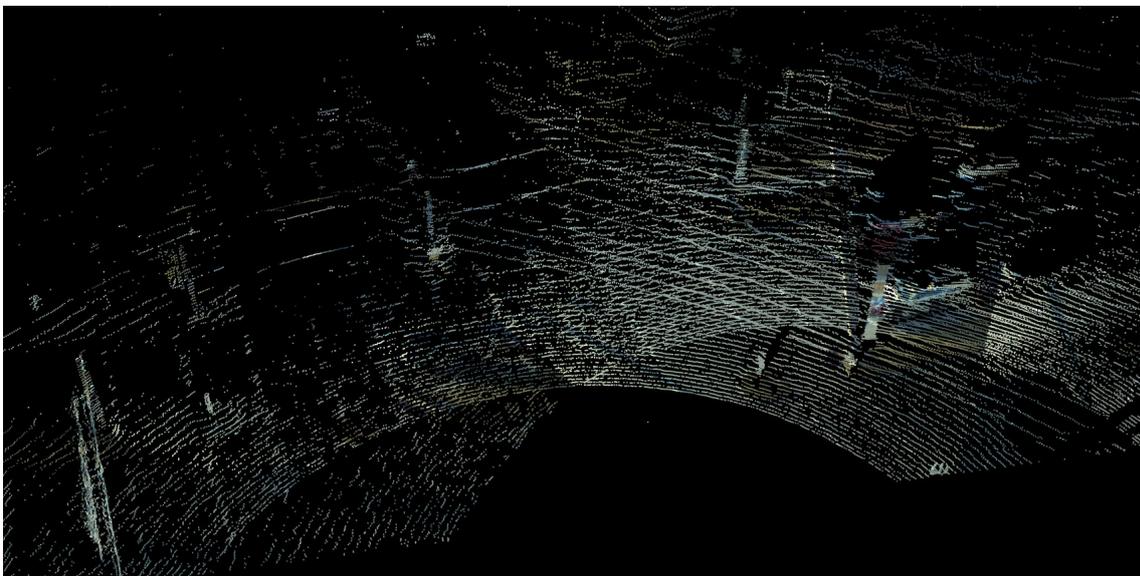
The intrinsic of S110 south1 camera is listed in Eq. 4.4.

$$K = \begin{bmatrix} 1253.45 & 0 & 957.907 \\ 0 & 1264.81 & 628.129 \\ 0 & 0 & 1 \end{bmatrix} \quad (4.4)$$

The projection result and colored point cloud is visualized in Fig. 4.24 and Fig.4.25.



**Figure 4.24:** Projection result of S110 north LiDAR into south1 camera. All infrastructure and lane markings are fitted well.



**Figure 4.25:** Colored point cloud from `s110_lidar_ouster_south` and `s110_camera_basler_south1_8mm`. We attach 2D pixels from the projected 3D points in the image to the 3D coordinates and color the point cloud inside the neighbourhood (voxel) of the transformed pixel with its color.

### S110 north LiDAR to S110 south2 camera

68 point pairs were extracted and visualized in Fig. 4.26. The extrinsic of S110 north LiDAR



**Figure 4.26:** Here we extract 68 points in the camera image (`s110_camera_basler_south1_8mm`) and the point cloud (`s110_lidar_ouster_north`). We mainly focus on the corners of objects that are easy to be identified in the point cloud, including traffic signs, lane markings, gantry bridge etc..

to south2 camera is listed in Eq. 4.5.

$$C = \begin{bmatrix} 0.37383 & -0.927155 & 0.0251845 & 14.2181 \\ -0.302544 & -0.147564 & -0.941643 & 3.50648 \\ 0.876766 & 0.344395 & -0.335669 & -7.26891 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.5)$$

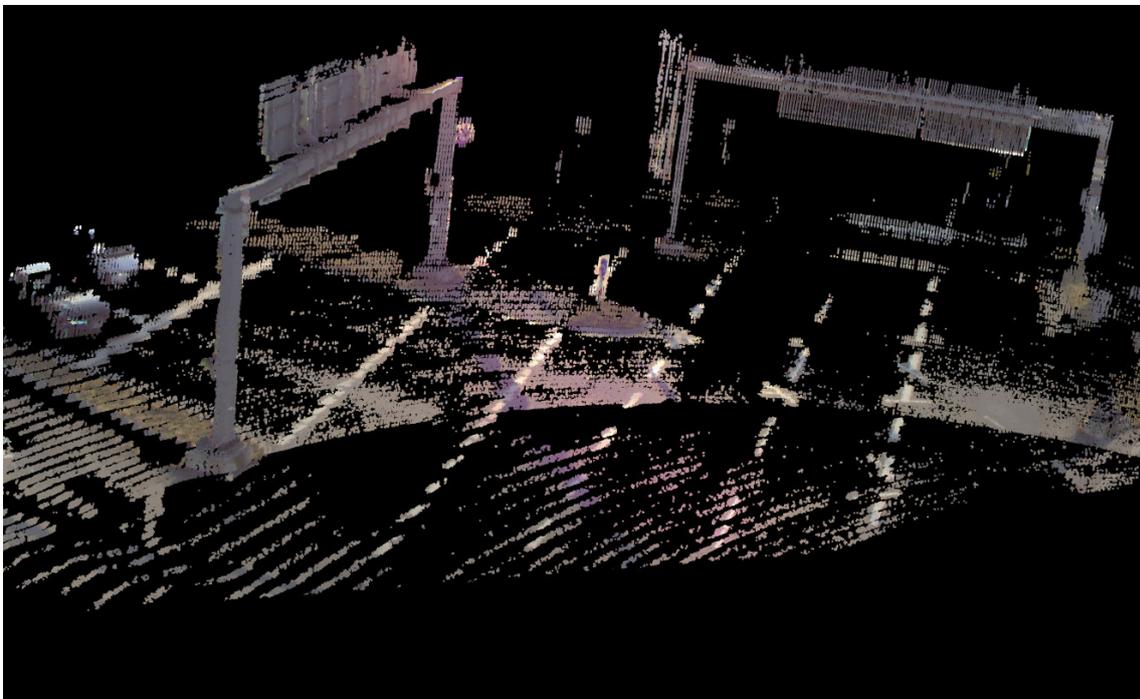
The intrinsic matrix  $K$  of the S110 south2 camera is listed in Eq. 4.6.

$$K = \begin{bmatrix} 1282.35 & 0 & 957.578 \\ 0 & 1334.48 & 563.157 \\ 0 & 0 & 1 \end{bmatrix} \quad (4.6)$$

The projection result and colored point cloud is visualized in Fig. 4.27 and Fig. 4.28.



**Figure 4.27:** Projection result of S110 north LiDAR into south2 camera. All infrastructure and lane marks are fitted well.



**Figure 4.28:** Colored point cloud of s110 north LiDAR and south2 camera. We attach 2D pixels from the projected 3D points in the image to the 3D coordinates and color the point cloud inside the neighbourhood (voxel) of the transformed pixel with its color.

### 4.2.3 Short discussion

From the projection above we can see, manual calibration can obtain excellent results. Because 2D and 3D key points are directly selected and matched manually to build the correspondences. On the other hand, manual calibration requires very complex processing steps, each of which has to be done manually on the static data. And those processes are very time-consuming. The shortcoming makes manual calibration impossible for real-world autonomous driving systems. In the next chapter we will describe automatic calibration in detail, which will be capable for the real-world calibration.

# Chapter 5

## Automatic Calibration

In this chapter, we will introduce the solution and experimental details for automatic calibration. In Section 5.1 we will introduce the method of automatic LiDAR camera calibration from HongKong university, our calibration model is based on their early works. From Section 5.2 to 5.7 we will make a full description about the contribution we made to the model. In Section 5.2 we will make a brief description about our calibration pipeline(both single frame and multiple frame calibration). In Section 5.3 we will make a brief statement about our work of hyper parameter searching and the influence of those parameter. In Section 5.4 we will introduce our image preprocessing module for background filtering in detail. In Section 5.5 we will introduce our image preprocessing module for shadow filtering in detail. In Section 5.6 we will make a complete description about our LiDAR point cloud preprocessing module (including format transformation, registration, scattering, outlier removal and upsampling). In Section 5.7 we will introduce a new method of voxel building , which can make our model adaptive to different scenes.

### 5.1 Voxel-Based Automatic LiDAR-to-Camera Calibration

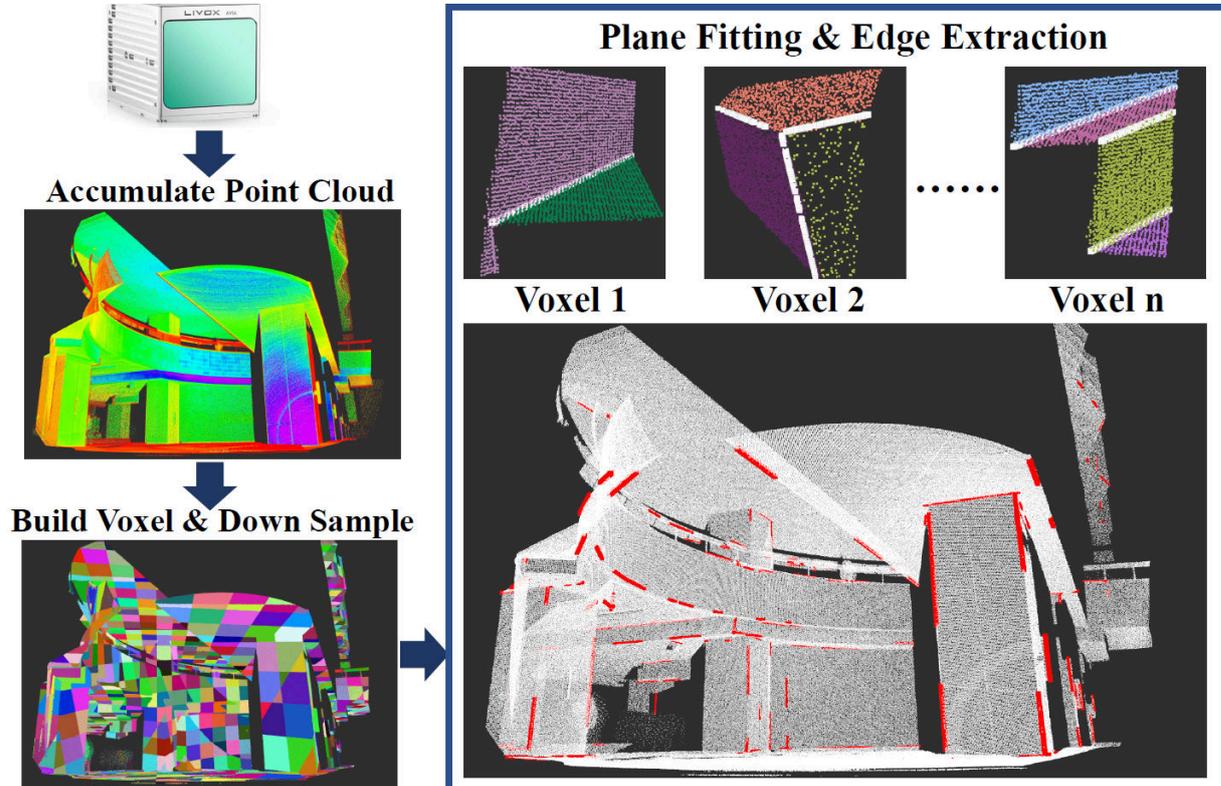
In any calibration task between LiDAR and camera, the most difficult and interesting part is to establish the corresponding relationship between 3D point cloud and 2D image, which involves complex algorithms such as image processing, image feature recognition, and point cloud feature extraction. Once the correspondence (3D-2D) between point clouds and image is established, the calibration task will be transformed into a pure mathematical problem, which can be easily solved by simple algorithms such as SVD [6] or BA [80].

Classical automatic calibration methods rely on specific objects placed in the scene, such as a chessboard, AprilTag, etc. Since these specialized targets contain obvious features (textures, patterns), they are very easy to be identified. This kind of methods are easy to implement, the algorithm is simple, and robust to the disturbance of the environment. But the dependence on artificial targets makes them very limited in real-world application.

Therefore, the targetless calibration method has become a new direction of automatic calibration. Our work is mainly based on the paper "Pixel-level Extrinsic Self Calibration of High Resolution LiDAR and Camera in Targetless Environments" [91] by Hong Kong University. In this paper, they introduced a targetless calibration method based on RANSAC plane fitting and Voxel cutting. This method can automatically cut the point cloud into voxels of fixed size, then use RANSAC plane fitting[29] to obtain the plane in the voxel, and then use the normal vector of the plane to find the edge of the plane. By matching between the LiDAR

edges and the Canny edges in image [13] to build the error, and finally using the PnL[52] algorithm to optimize the reprojection error for calibration.

### 5.1.1 Voxel-based Edge Extraction



**Figure 5.1:** "Depth-continuous edge extraction. In each voxel grid, different colors represent different voxels. Within the voxel, different colors represent different planes, and the white lines represent the intersections between planes" [91].

Firstly divide the point cloud into small voxels of given sizes (e.g., 0.05 meter). For each voxel, we repeatedly use RANSAC[29] to extract planes in the voxel. Then, find plane pairs that are connected and whose normal form an angle within a certain range (e.g.,  $[20^\circ, 160^\circ]$ ) and calculate the intersection lines (so-called depth-continuous edge) [91]. As shown in Fig. 5.1, after plane fitting we can extract edges to solve for the intersection lines between plane pairs form the angle within a specific range.

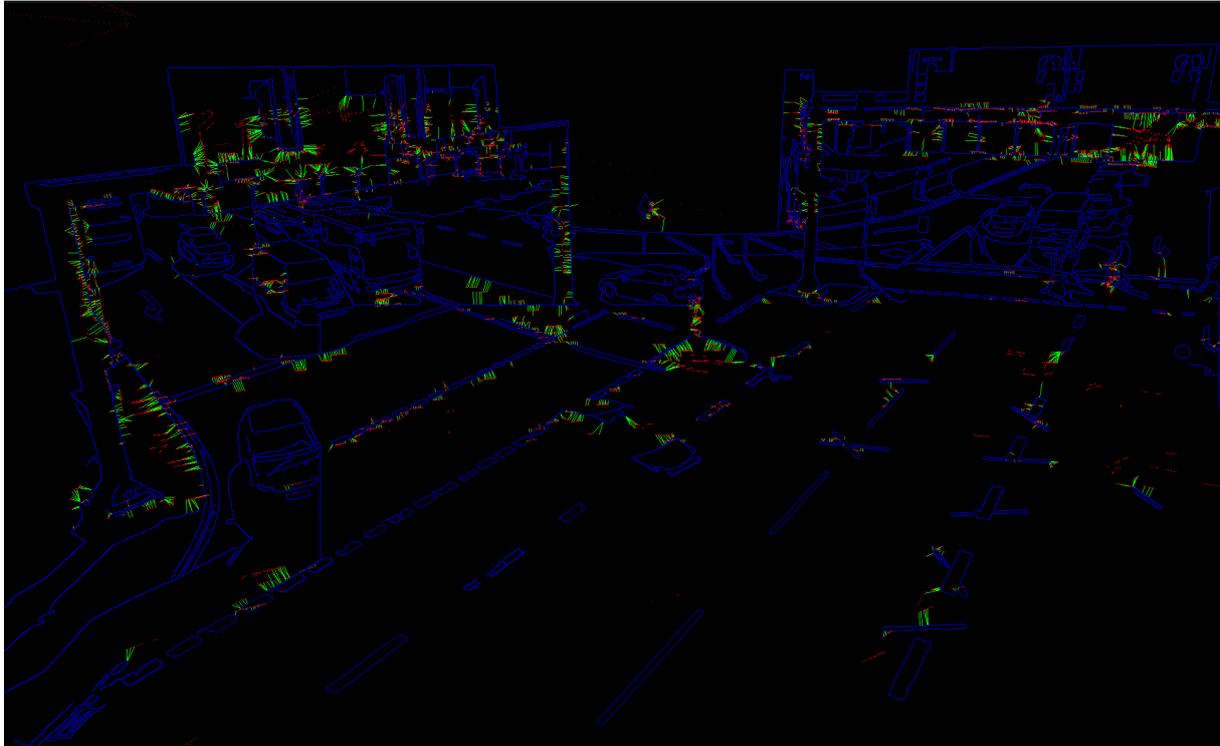
### 5.1.2 LiDAR-Camera Edge Pair Matching

After the edge extraction process described above, we obtain the LiDAR edges in the point cloud, which are composed of a series of individual 3D LiDAR points. Then we project these edges to the image plane through initial extrinsic and intrinsic to obtain 2D LiDAR edges. By pre-setting the search radius, we can establish a correspondence between LiDAR edges and Canny edges within the radius.

To be more specific, we search the  $\kappa$  nearest neighbor of  $\mathbf{p}_i$  in the  $k$ -D tree built from the image edge pixels [91]. Denotes  $\mathbf{Q}_i = \{\mathbf{q}_i^j; j = 1, \dots, \kappa\}$  the  $\kappa$  nearest neighbours and we have:

$$\mathbf{q}_i = \frac{1}{\kappa} \sum_{j=1}^{\kappa} \mathbf{q}_i^j; \quad \mathbf{S}_i = \sum_{j=1}^{\kappa} (\mathbf{q}_i^j - \mathbf{q}_i)(\mathbf{q}_i^j - \mathbf{q}_i)^T. \quad (5.1)$$

Fig. 5.2 shows an example of the edge pairs we extracted from A9 dataset.



**Figure 5.2:** LiDAR edges (red lines), image edge pixels (blue lines) and their correspondences (green lines).

### 5.1.3 Calibration Formulation and Optimization

As equation 5.2 shows,  ${}^L\mathbf{p}_i$  represent a 3D edge point extracted from the LiDAR point cloud and the corresponding 2D edge in the image is represented by its normal vector  $\mathbf{n}_i \in \mathbb{S}^1$  and a point  $\mathbf{q}_i \in \mathbb{R}^2$  lying on the edge; then we project it to the image plane using the optimal extrinsic and compensating the noise in  ${}^L\mathbf{p}_i$ . The projection should lie on the edge  $(\mathbf{n}_i, \mathbf{q}_i)$  extracted from the image [91].

$$0 = \mathbf{n}_i^T \left( \mathbf{f} \left( \pi \left( {}^C\mathbf{T} \left( {}^L\mathbf{p}_i + {}^L\mathbf{w}_i \right) \right) \right) - \left( \mathbf{q}_i + {}^I\mathbf{w}_i \right) \right) \quad (5.2)$$

where  ${}^L\mathbf{w}_i \in N(\mathbf{0}, {}^L\Sigma_i)$  and  ${}^I\mathbf{w}_i \in N(\mathbf{0}, {}^I\Sigma_i)$  are noise.

Then we can establish a maximum likelihood estimation (MLE) of extrinsic and construct a least square cost function (5.3).

$$\begin{aligned} \max_{\delta \mathbf{T}} \log p(\mathbf{v}; \delta \mathbf{T}) &= \max_{\delta \mathbf{T}} \log \frac{e^{-\frac{1}{2} \mathbf{v}^T (\mathbf{J}_w \Sigma \mathbf{J}_w^T)^{-1} \mathbf{v}}}{\sqrt{(2\pi)^N \det(\mathbf{J}_w \Sigma \mathbf{J}_w^T)}} \\ &= \min_{\delta \mathbf{T}} (\mathbf{r} + \mathbf{J}_T \delta \mathbf{T})^T (\mathbf{J}_w \Sigma \mathbf{J}_w^T)^{-1} (\mathbf{r} + \mathbf{J}_T \delta \mathbf{T}) \end{aligned} \quad (5.3)$$

The optimal solution is:

$$\delta \mathbf{T}^* = - \left( \mathbf{J}_T^T (\mathbf{J}_w \Sigma \mathbf{J}_w^T)^{-1} \mathbf{J}_T \right)^{-1} \mathbf{J}_T^T (\mathbf{J}_w \Sigma \mathbf{J}_w^T)^{-1} \mathbf{r} \quad (5.4)$$

In each iteration, this solution will be updated to the transformation matrix and used for next iteration.

For evaluation, we can also normalize this cost with the number of matched edge pairs:

$$\frac{1}{N_{match}} \mathbf{r}^T (\mathbf{J}_w \Sigma \mathbf{J}_w^T)^{-1} \mathbf{r} \quad (5.5)$$

## 5.2 Automatic Calibration Pipeline

In this section we will briefly introduce our calibration pipeline. From here on, the rest of this chapter will be about our contribution to the project.

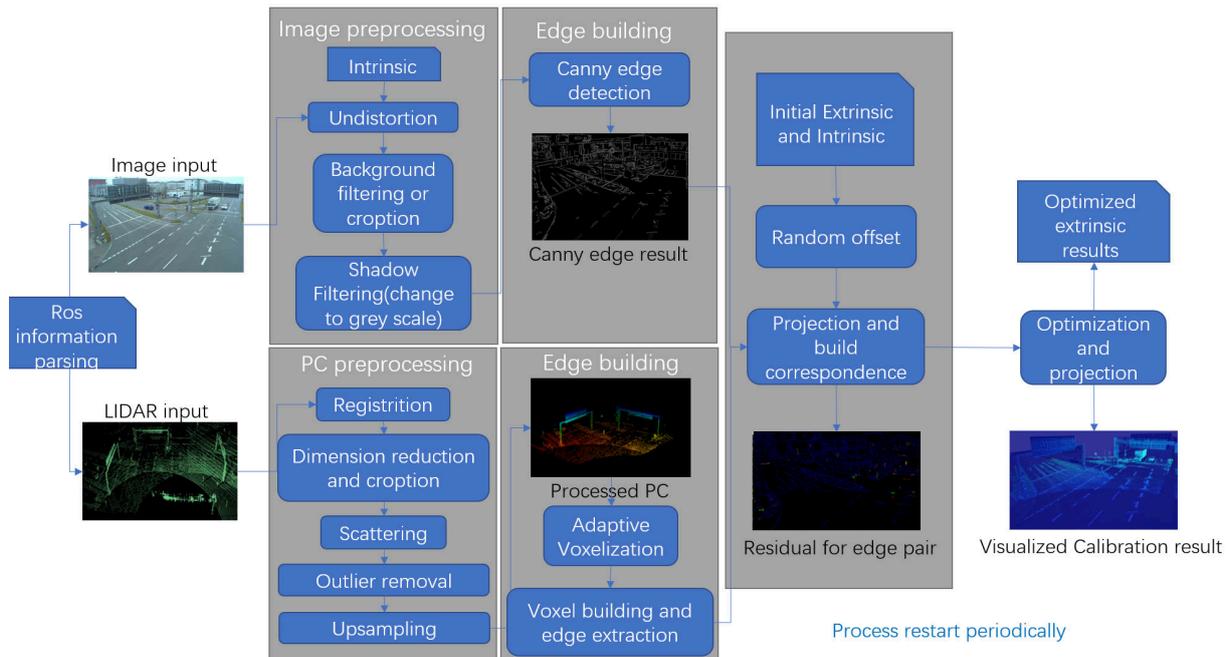
### 5.2.1 Single Frame Calibration

Figure 5.2 shows the pipeline for our Automatic LiDAR-Camera Calibration process using live sensor data. We can specify a time period (e.g. every 5 minutes), in that the system will automatically subscribe to the sensor information (LiDAR point cloud and camera image) via ROS (robot operating system) and go through the whole pipeline automatically. All we need to do is to activate the program with one command.

For image processing, the program will automatically subscribe to the image and do the undistortion to get an undistorted image input. Afterwards, it will perform background filtering to crop the background which is out of range of the LiDAR's FoV. If there is a shadow on the ground, we can activate the shadow filtering component to remove the shadow in the image, which will improve the performance of the Canny edge detector. Finally, we use the Canny edge detection algorithm to get the edge map.

For LiDAR preprocessing, we will first register three LiDARs with different FoV at the S110 intersection to the one we want to calibrate. The reason is that one single LiDAR can not perceive some parts of the objects in camera image. Afterwards, we pick the first four dimensions (x,y,z location and intensity) and crop the region outside the camera FoV. After that we scatter the point cloud to increase the density. Then we do outlier removal where we remove the noisy points with a radius outlier removal method implemented in the *PCL* library. Finally, we perform point cloud upsampling to increase the number of 3D points.

After preprocessing, we use the method stated in Section 5.1 to extract LiDAR edges. Then we project those edges onto the image with initial extrinsic and intrinsic. Using the KNN



**Figure 5.3:** Automatic Calibration Pipeline. We add many camera images and LiDAR point cloud preprocessing modules in order to make it capable for real-world calibration. The algorithm will take the input data by subscribing to ROS topics on the live system and output both calibration results and qualitative projection of point clouds on the images.

method we build correspondences between edges. After the correspondences are built, the ceres solver will be used for maximum likelihood optimization.

## 5.2.2 Multiple Frame Calibration

For better robustness against the changing scenes, we also develop an optimization process to use multiple frames as input. In the live data, the scenes in the highway intersection always change. For example, the pedestrians and vehicles position within the FoV changes by the time, and thus the edges extracted will be different. In order to solve this problem, we use multiple frames in one time step, and optimize them simultaneously. Finally, we compute a mean of those results.

## 5.2.3 Offline Calibration

We use offline calibration for local tests and debugging. This module can operate without any live sensor input and instead loads a camera image and LiDAR point cloud from the hard drive. Like the live version, it is also capable to use multiple frames for calibration if multiple inputs are given.

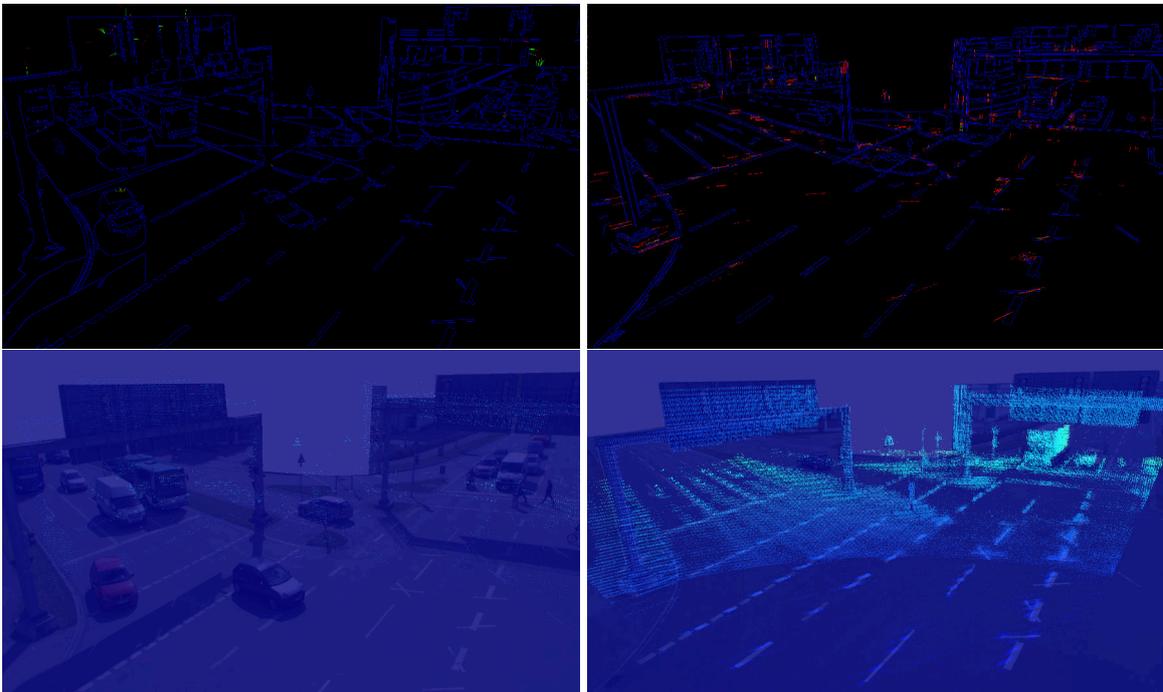
### 5.3 Hyperparameter

In this section we will introduce some important hyperparameters, which will influence the results of calibration highly.

#### 5.3.1 Key Principle of Calibration

Since our calibration is a voxel-based edge extraction method, maximum likelihood optimization is performed based on the established edge pairs. Therefore, the key is to establish correct and sufficient edge pairs and try to distribute them evenly throughout the FoV. Once a wrong correspondence is established or the distribution is uneven, even if a lower cost is obtained, the projection results will still be unsatisfactory. Fig. 5.4 shows a comparison between good and bad edges.

Hence, all of the work below is about building the correct edge pairs.



**Figure 5.4:** The figure in the top left corner shows an example of bad edge pairs. The corresponding calibration result can be seen in the bottom left corner. The figure in the top right corner shows an example of good edge pairs. The corresponding calibration result is visible in the bottom right corner. We can see that even distributed edge pairs are crucial for the calibration. Due to the sparsity of point cloud in the left, we can extract few edges (top left), and lead to a bad calibration result.

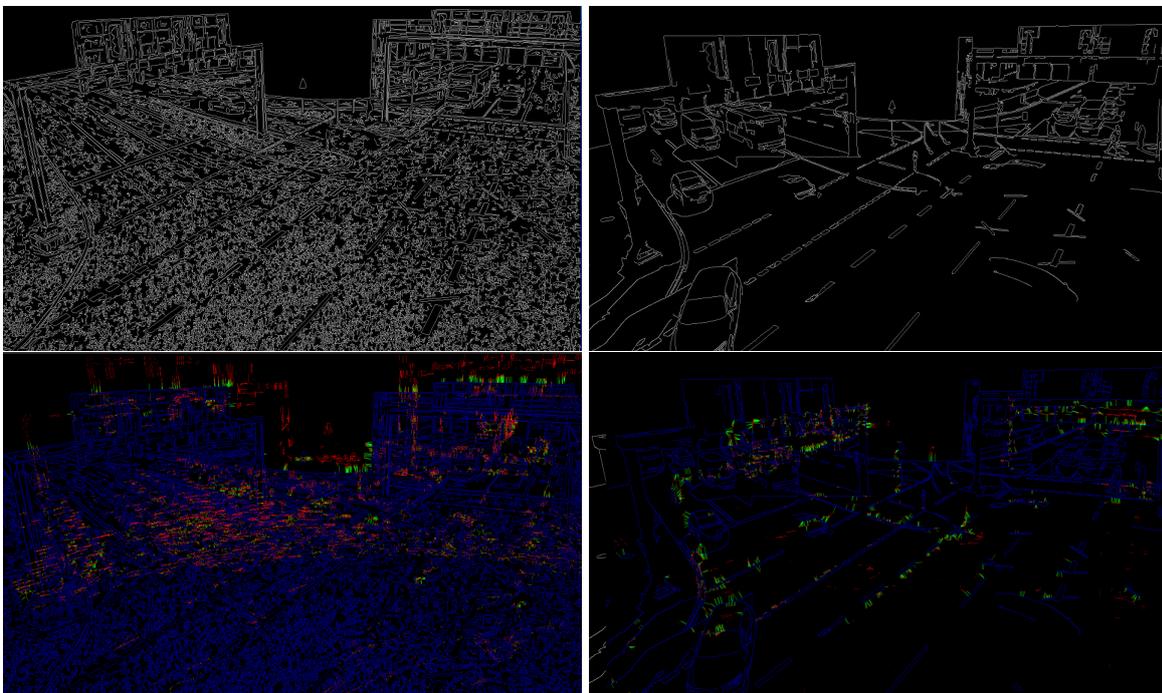
#### 5.3.2 Canny Edge Extraction

Canny edge detector is used to extract edges in 2D images, and all correspondences are established based on these edges. The performance of the Canny edge detector is determined

by the following parameters:

- `canny_image_edge_gray_threshold_low`: Specifies the lowest threshold for the grey value of image. Adjust it for different environments.
- `canny_image_edge_gray_threshold_high`: Specifies the highest threshold for the grey value of image. Adjust it for different environments.
- `canny_image_edge_min_length_threshold`: Performs edge picking based on the Canny's result. Only pick edges with a sufficient length. Reducing its value will pick more edges.

Fig. 5.5 shows an example:



**Figure 5.5:** Example of bad Canny results that generate too much edges from noise and thus build many wrong correspondences(left).A nice canny result will always and only contains the necessary edges.

### 5.3.3 Voxel Cutting

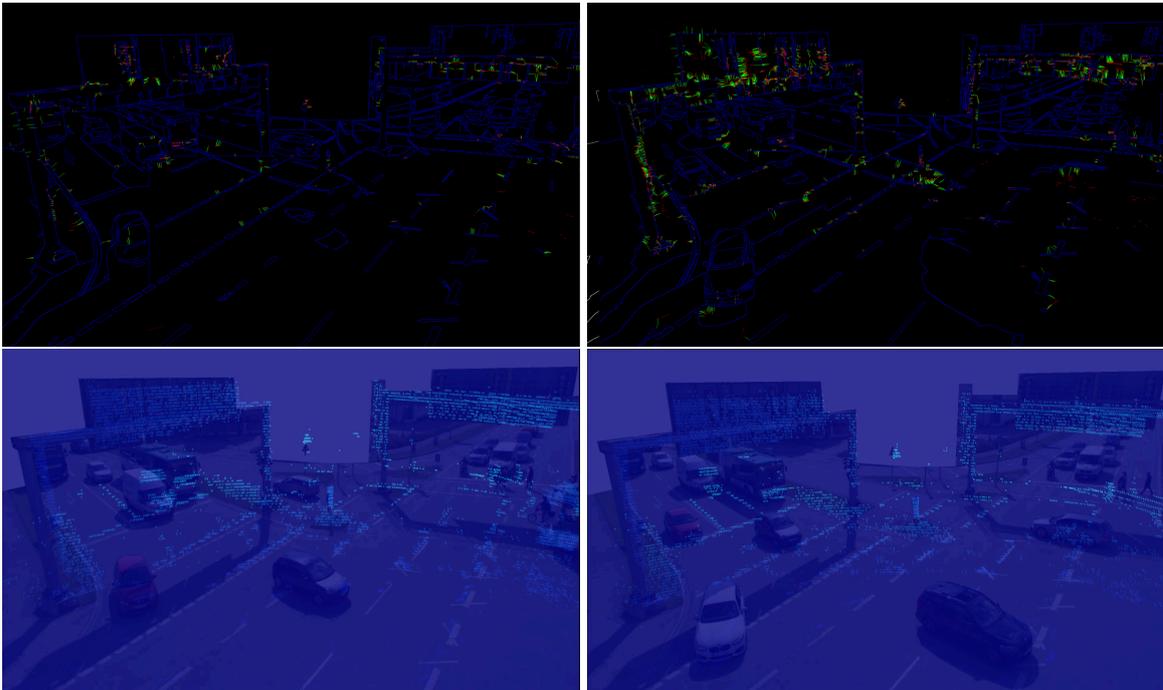
The following parameters determine the voxel size and downsampling scale:

- `Voxel.size`: Specifies the size of the voxel. Reducing its value will be possible to extract more planes but will be time consuming and increase the computation resources.
- `Voxel.down_sample_size`: Specifies the downsample size of voxels.

### 5.3.4 Plane Fitting

There are many parameters that determine the plane fitting, but most of them are being used to adjust to different PCD input sizes. Only two parameters are very important for the result, they are used to justify whether two planes are considered intersect.

- **Plane.normal\_theta\_min:** Specifies the minimal value between the normal vector of the two planes. If the angle is larger than this threshold, the two planes will be considered as different surfaces, and the intersection line between the two planes will be added to edges. In another word, all points on this intersection line will contribute for the calibration.
- **Plane.normal\_theta\_max:** Specifies the maximal value between the normal vector of the two planes. If the angle is smaller than this threshold, the two planes will be considered as different surfaces, and the intersection line between the two planes will be added to edges.



**Figure 5.6:** LiDAR edges extracted with plane threshold in  $(70,110)^\circ$  range. Only pairs of planes that are nearly parallel will be considered intersecting, so only few edges can be extracted. The results also show a underfitting of lane marks and vehicles. **Figure 5.7:** LiDAR edges extracted with plane threshold in  $(15,165)^\circ$  range. Now all plane pairs which are orthogonal to each other can be considered intersecting, so many more edges can be extracted. The results also show a better fit of lane marks and vehicles.

As Figures 5.6 and 5.7 show, if we expand the range of angles, we can extract more edges from the plane. For lane markings in particular, they usually have very small angles to the ground, so a wider range is required to capture these edges.

### 5.3.5 Edge Cutting

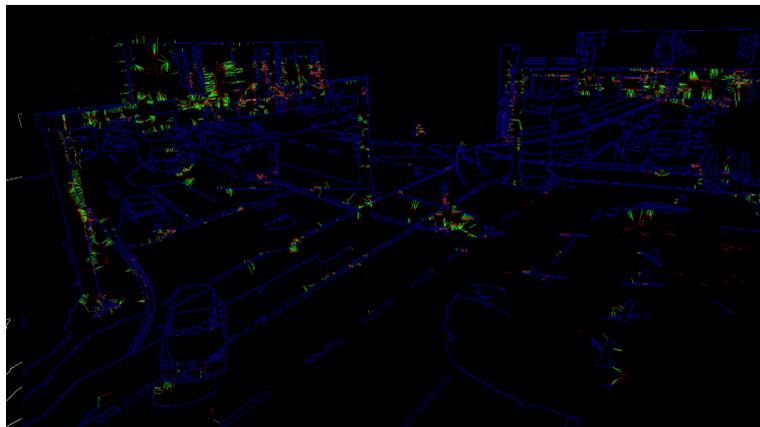
In this subsection we will introduce some parameters for edge cutting:

- `Edge.min_dis_threshold`: Specifies the minimum distance between edges. Increasing this value can avoid overfitting.
- `Edge.max_dis_threshold`: Specifies the maximum distance between edges. Any edge pair with distance larger than this will not be selected. Increasing this value can find correspondences in larger distance, but will also make it possible to build wrong edge pairs.

The use of these two parameters is quite tricky. Here are some representative cases:

#### Unequally Distributed Features

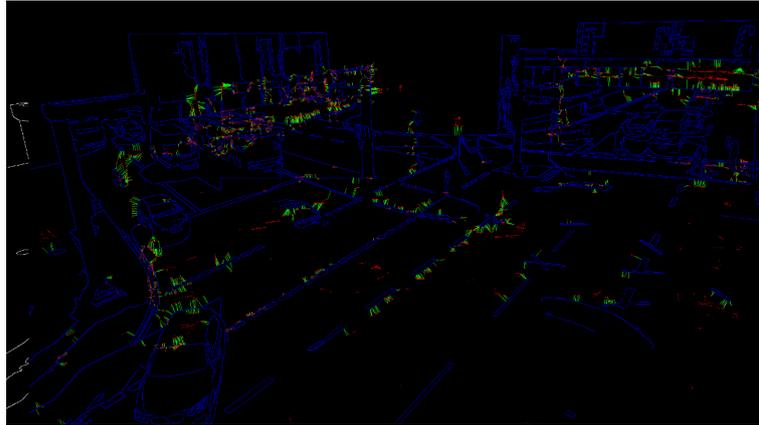
Because the texture of the gantry bridge is much more complicated than of the ground (see Fig. 5.8), and the reflectivity of both LiDAR and camera is higher in this area, more edges are captured on the gantry. This leads to an overfitting problem: The ceres optimizer will overfit the gantry and ignore the landmarkings, which lead to a bad optimization result. Increasing the minimal threshold of the edge distance will mitigate this problem. As the iterations continue, the fitted edge pairs get closer to each other. Edge lengths larger than the threshold are filtered. They will not contributing to the cost function anymore.



**Figure 5.8:** Edges are unequally distributed. Obviously the edges on the gantry are much more abundant than on the ground.

#### Wrong Correspondences

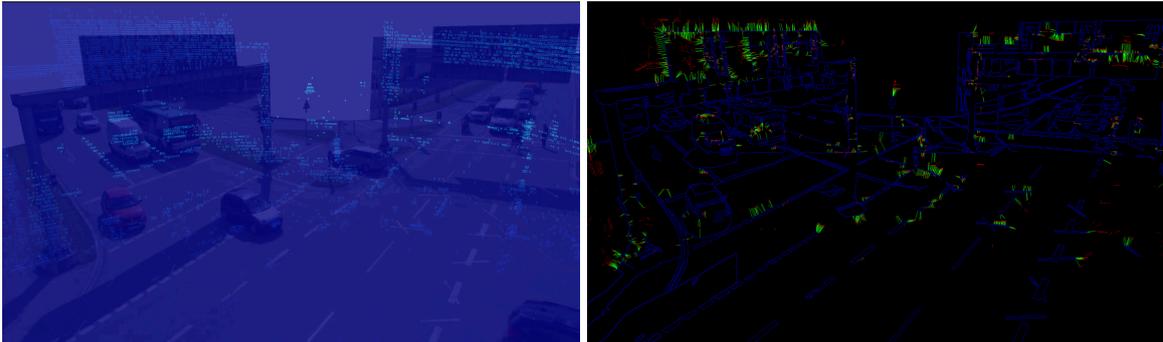
As Figure 5.9 shows, wrong correspondences are built between the gantry point cloud and Canny edges from vehicles. Reducing the maximal edge distance will mitigate this problem.



**Figure 5.9:** There are many vehicles behind the gantry. The wrong edge pairs are built between the gantry point cloud and the vehicle Canny edges.

### Importance of Initial Extrinsic

As Fig.5.10 shows, if the initial extrinsic is seriously deviated, which lead to the first projection of point cloud far away from Canny edges, the algorithm can hardly build a correct correspondences.



**Figure 5.10:** In this case, the initial extrinsic is quite inaccurate and thus the first projection is quite far away (left figure) which leads to wrong correspondences (right figure).

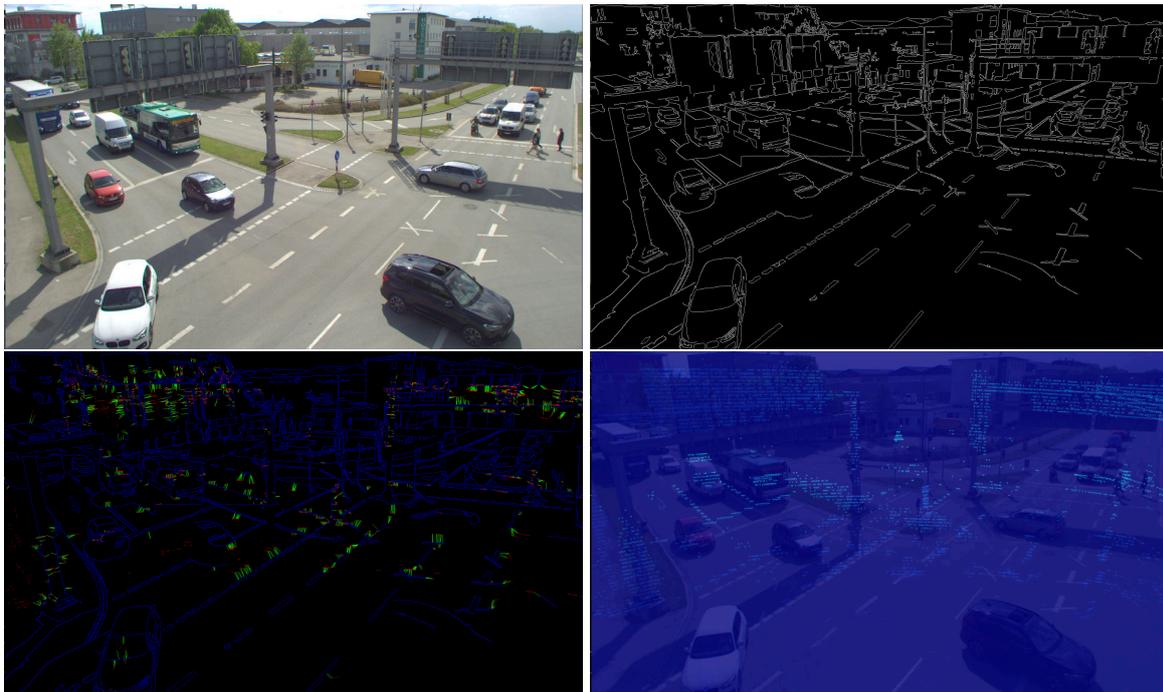
## 5.4 Background Filtering and Cropping

In this section we will introduce an automatic background detection and cropping method, and also a template based background cropping method. In 5.4.1 we will make a brief problem statement. In 5.4.2 we will introduce the automatic background detection method. In 5.4.3 we will introduce our template base method. In 5.4.4 we will present some results of background filtering, and see how it influences the edge pairs.

### 5.4.1 Problem Statement

In the real world, the data mismatch always exists. This is because of the limitation of the hardware itself. LiDARs have a limited range. The intensity of a laser beam reduces with the distance because the various particles in the air will hinder the transmission of light. Hence, many buildings captured by cameras are not visible in LiDAR. During the calibration, this data mismatch will build wrong correspondences between point cloud and 2D camera edges.

As the Figure 5.11 shows, the buildings are outside of the range of LiDAR, but the camera still can capture them. As a result, if the initial extrinsic is deviated in Z axis, our algorithm will build wrong correspondences. In order to solve this problem, background filtering is applied.



**Figure 5.11:** Raw input image with buildings (top left), edge map with buildings (top right), edge pairs with buildings (bottom left) and projection results (bottom right). We can see from top right, the canny edge detector extract edges from buildings. And correspondences are built between building edges and LiDAR point clouds (bottom left), and thus lead to a bad calibration result.

### 5.4.2 Automatic Background Removal

#### Monocular Depth Estimation

In order to automatically remove the background buildings in our experiment data, we make use of an excellent pretrained monocular depth estimation network proposed in [68]. With this network, we can get an inverse depth map from the original input. It is worth mentioning that this approach is semantically based and does not require a video to capture dynamic objects. This makes it very practical for our calibration.

Based on the work from the *MiDaS* group, we developed our own background removal procedure. It can use the depth threshold to create a mask map. This mask map will represent

all pixels below the threshold as background. By pixel-wise checking the mask map, we crop the original input image and remove the background. This process is fully automatic and can handle multiple inputs simultaneously. Some results are shown in Figure 5.12 and 5.13:



**Figure 5.12:** S110 south2

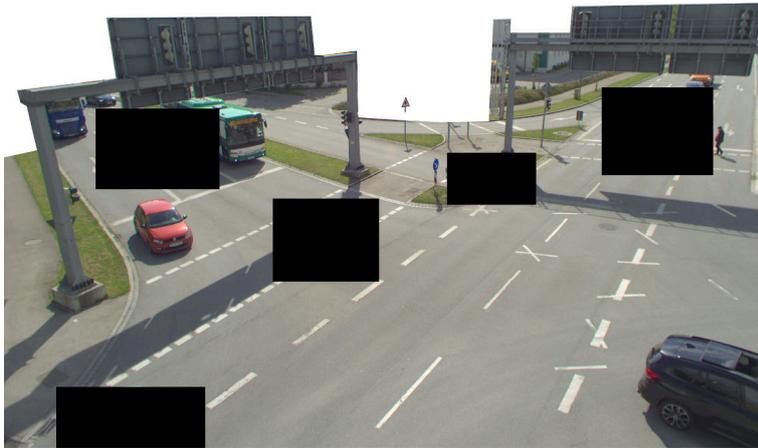
**Figure 5.13:** S110 south1

**Figure 5.14:** From top to bottom: original input, the inverse depth map, the colored depth and the cropped output.

### 5.4.3 Template based Background Cropping

We also develop another method based on a previously cropped background template. In this template image, the background that we want to remove is already covered by the mask. This template will be loaded at the beginning of calibration. During the calibration the background in the input images will be automatically removed. Fig. 5.15 shows an example of hand cropped template.

The template cropping is loaded as a matrix in the program. Once the input image is parsed

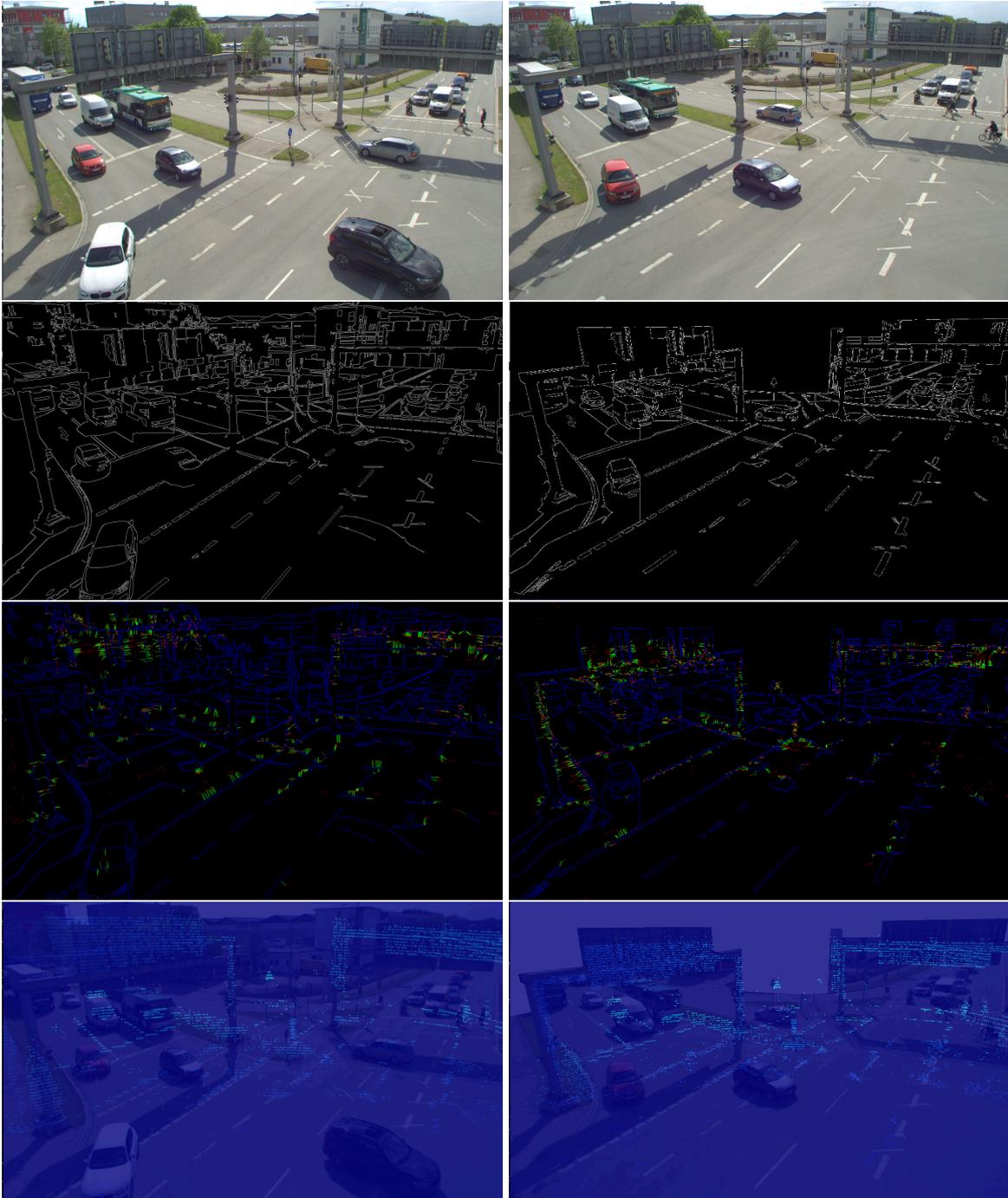


**Figure 5.15:** An example input template. The background is filled with white color and white vehicles are boxed with black color to get better correspondences.

from the ROS information flow, the algorithm will check this matrix and pixel wise crop the input images.

### 5.4.4 Template based Background Cropping Results

As Figure 5.16 shows, our background cropping successfully solve the problem and improve the performance of the Canny edge detector.



**Figure 5.16:** From top to bottom: Raw camera input, edge map, edge pairs with residual (green lines), calibration results. We can see that without background filtering the edges of buildings are also connected to the LiDAR edges and lead to a poor result (see bottom left figure). After filtering the bad correspondences the problem is solved (see bottom right figure).

## 5.5 Dealing with Shadows

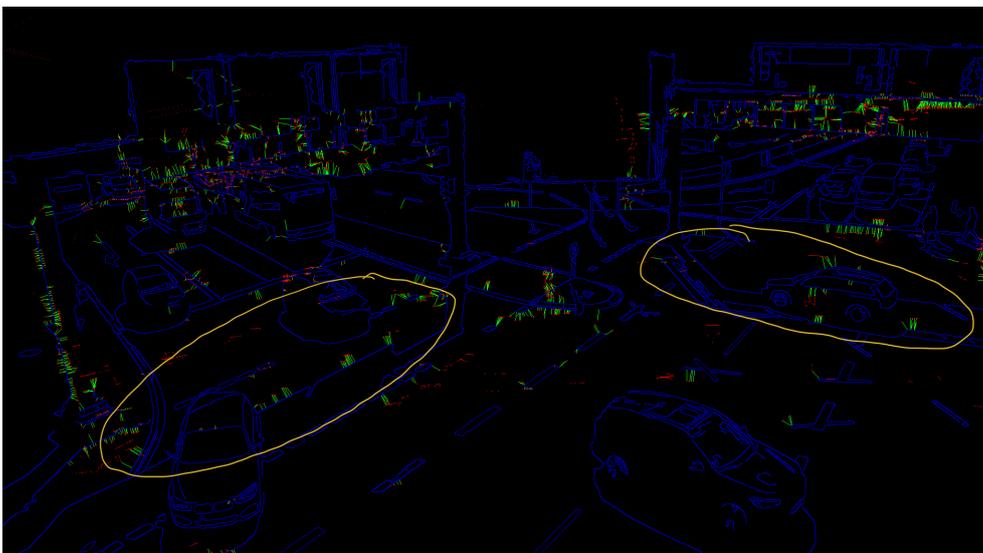
### 5.5.1 Effect of Shadows

At certain times of the day, sunlight casts shadows into the calibration area. Unfortunately, our canny edge detector is not very good at distinguishing between shadows and lane markings on the ground. As a result, many false edges are generated on the ground, which confuses our algorithm and establishes many false correspondences.

In order to mitigate this problem, we apply a trick to modify the shadow in the raw image.



**Figure 5.17:** Shadow on the raw input. Some shadows mix with lane markings and vehicles in the zone of interest.



**Figure 5.18:** Shadows within the yellow circle show their influence on our model. The algorithm takes the edge from the shadow as the lane markings and builds wrong correspondences.

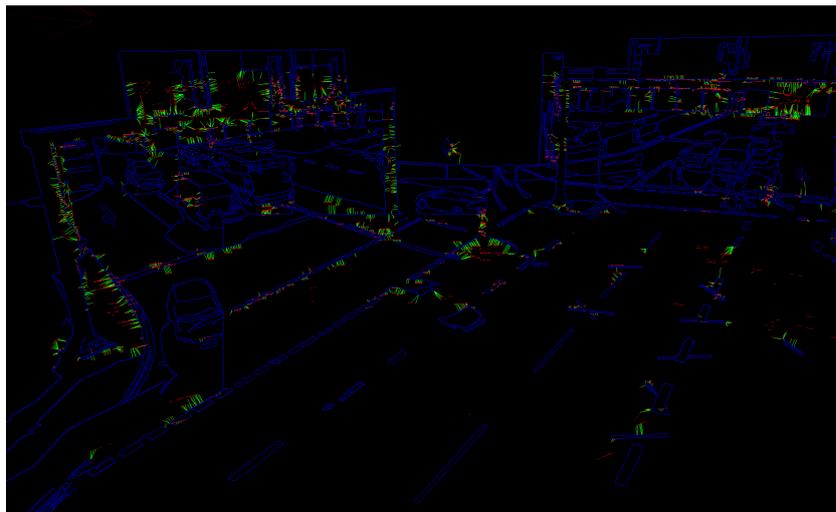
### 5.5.2 Shadow filtering

An intuitive way to solve this problem is to modify the parameter of Canny's edge detector, but actually it fails. If we look into the figures in the last subsection, we can find the brightness of shadows is very close to the back side of the gantry, if we simply adjust the parameter, the gantry will also be filtered out. This is what we would not expect to happen.

In order to reduce the effect of shadows as much as possible while preserving other features, we came up with an interesting idea. We pick a pixel on the ground which is exposed to sunlight, and specify a square zone. All pixels inside the filtering zone will be checked on their grey scale value. If the gray scale value is smaller than the benchmark pixel, the grey scale of this pixel will be increased with a specific level (specified by a special parameter in the yaml file). Fig.5.19 and Fig.5.20 represent an example for edge filtering.



**Figure 5.19:** The shadow within the window will be filtered, but the gantry which is outside of the window will not be affected.



**Figure 5.20:** The shadow is filtered and will not generate any confusing edges.

## 5.6 LiDAR preprocessing

In this section we will make a detailed description of our LiDAR preprocessing process. The first step is registration. The method from our colleague Walter Zimmer will be adopted [101]. With this method we can directly register three LiDAR sensors to the one we want to calibrate. Subsection 5.6.1 will be about dimension reduction and point cloud cropping. Subsection 5.6.2 will be about scattering of the point cloud. Subsection 5.6.3 will introduce the noise filtering (outlier removal). Subsection 5.6.4 will represent a method of point cloud upsampling.

### 5.6.1 Dimension Reduction and Cropping

LiDAR point clouds are stored in the PCD format and published through ROS through PointCloud2 messages. Usually the PCD file is large and with abundant information, each point is represented as a high-dimensional vector. In order to reduce the computation costs, we only pick 4 dimensions: the  $[X, Y, Z]$  Cartesian coordinates and the intensity. Fig. 5.21 shows an original multi-dimensional PCD file, Fig. 5.22 shows a PCD after dimensionality reduction.

Furthermore, the raw point cloud contains regions outside of the camera's FoV, which we do

```
# .PCD v0.7 - Point Cloud Data file format
VERSION 0.7
FIELDS x y z intensity t reflectivity ring ambient range
SIZE 4 4 4 4 2 1 2 4
TYPE F F F F U U U U
COUNT 1 1 1 1 1 1 1 1 1
WIDTH 49835
HEIGHT 1
VIEWPOINT 0 0 0 1 0 0 0
POINTS 49835
DATA ascii
-1.073641 1.65874 0.0194164 46 14710570 20 0 308 1976
-1.05342 1.638466 0.01965586 93 14760780 41 0 246 1948
-1.049463 1.643376 0.01963875 71 14809540 31 0 210 1950
-1.0396 1.638986 0.01971572 93 14858950 41 0 228 1941
-1.034567 1.642167 0.01971572 90 14906300 39 0 247 1941
```

Figure 5.21: Original PCD file.

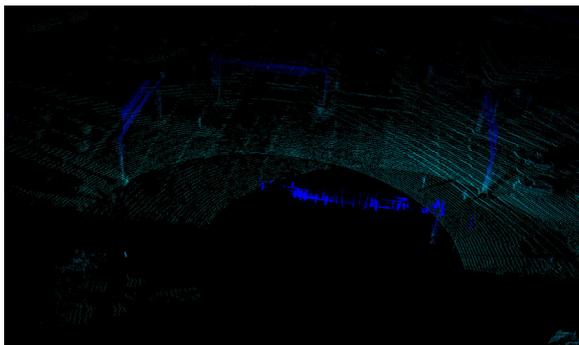
```
# .PCD v.7 - Point Cloud Data file format
VERSION .7
FIELDS x y z rgb
SIZE 4 4 4 4
TYPE F F F F
COUNT 1 1 1 1
WIDTH 213
HEIGHT 1
VIEWPOINT 0 0 0 1 0 0 0
POINTS 213
DATA ascii
0.93773 0.33763 0 4.2108e+06
0.90805 0.35641 0 4.2108e+06
```

Figure 5.22: PCD file with four dimensions.

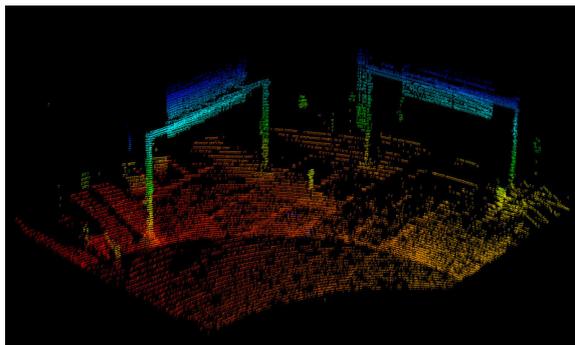
not need. They not only increase the computational load, but also create false edge pairs. Therefore, we implemented a module to crop out uninteresting regions. Fig. 5.23 shows an original point cloud at the S110 intersection, Fig. 5.24 shows a cropped point cloud.

### 5.6.2 Scattering

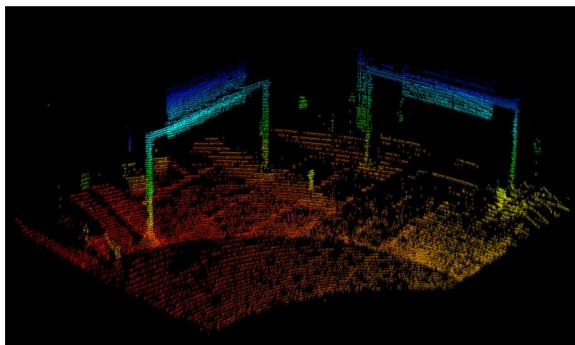
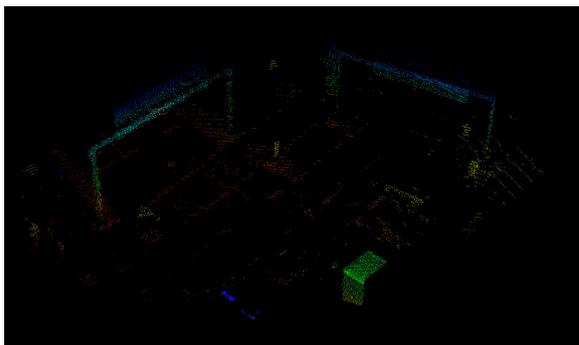
The point cloud contained in a single-frame PCD is too sparse, making it difficult for plane fitting and edge extraction. As a result, edge pairs established will not be sufficient for the calibration. One solution is to extract multi-frames of point clouds for synthesis. But due to the dynamic environments in real-live data, objects in the scene are moving. the point cloud synthesized from multiple frame will not find a matching picture. So we found out a solution called scattering. We only extract one frame of point cloud, and then set a range (usually 10 cm) around this point cloud. Then randomly generate points within the neighbouring range of original point. Fig. 5.25 shows the result of scattering.



**Figure 5.23:** Full range PCD, including the whole s110 intersection and also the buildings.



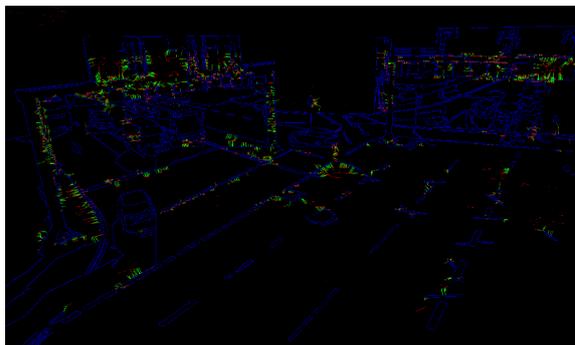
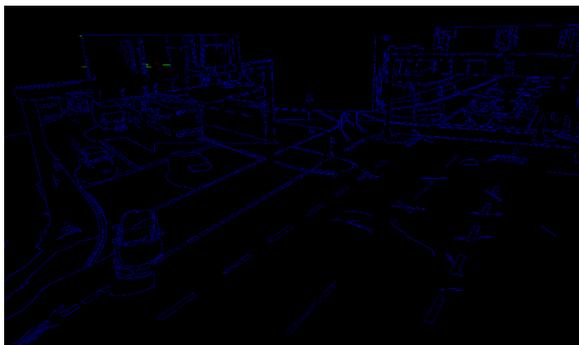
**Figure 5.24:** Cropped PCD, which only contains the zone of interest we need.



**Figure 5.25:** Point cloud from a single frame (left), point cloud with scattering applied 50 times (right). The density of point cloud is significantly increased.

We can also see the difference of edge pairs built in Fig. 5.26:

As we can see from the figure, after scattering much more edges can be extracted.



**Figure 5.26:** Edges from a single frame (left), edges with applying scattering 150 times (right). Using scattering we can extract much more edges in the point clouds.

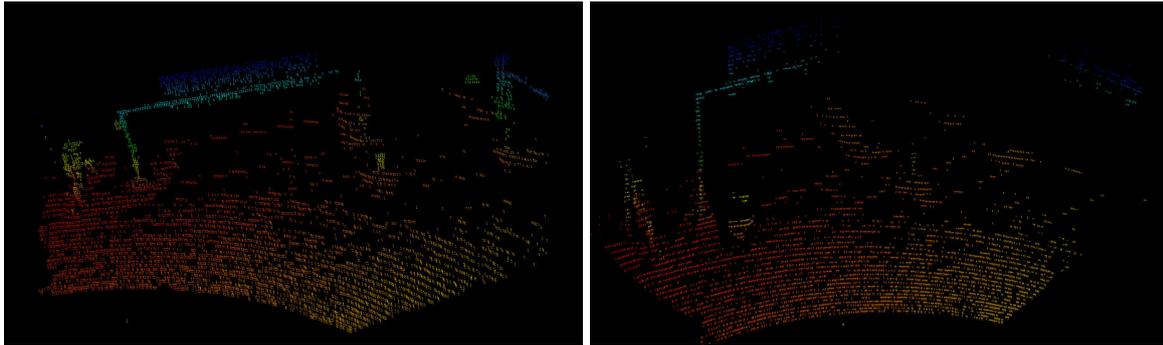
### 5.6.3 Outlier Removal

In real-life scanning, the point cloud scanned by a LiDAR usually contains lots of noise (outliers). Some of them are caused by dirt on the surface of the LiDAR instrument, and some are caused by environmental interference. Due to these outliers, some false correspondences

may be established. As a consequence, the calibration results will be changed by those wrong correspondences. In our calibration, this problem is especially serious on the ground. A large number of useless points on the ground greatly interfere with our calibration of lane marks. Hence outlier removal of the ground is necessary.

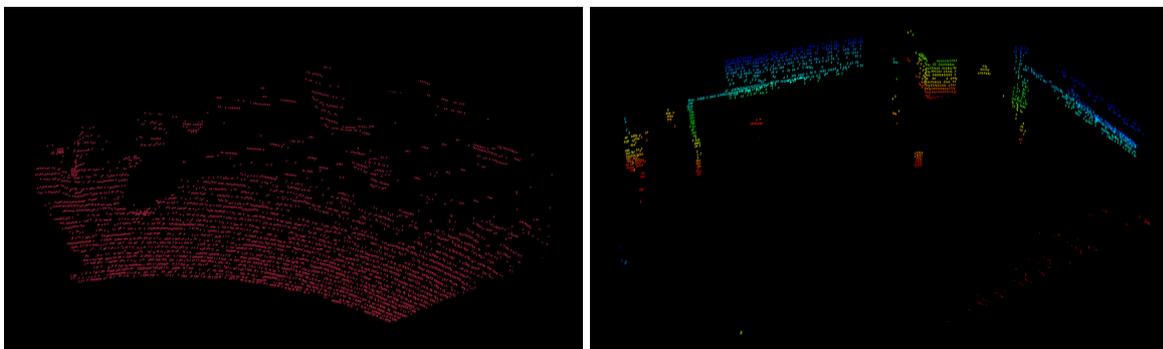
In our calibration case, a challenge lies in the different sparsity of the point cloud of objects and of the ground. As shown in Figure 5.27, some areas on the gantry bridge and traffic signs only reflect very sparse point clouds. If global filtering is used, we will lose quite a few object point clouds before the noise on the ground is filtered.

Therefore, we propose to filter the ground and background separately. As shown in Figure



**Figure 5.27:** An example of rough outlier removal. We directly apply outlier removal to the original point cloud (left). Then we filter more points on the gantry than on the ground (right).

5.28, we divide the complete point cloud into two parts, the ground and the objects, and perform more strict radius outlier removal on the ground.

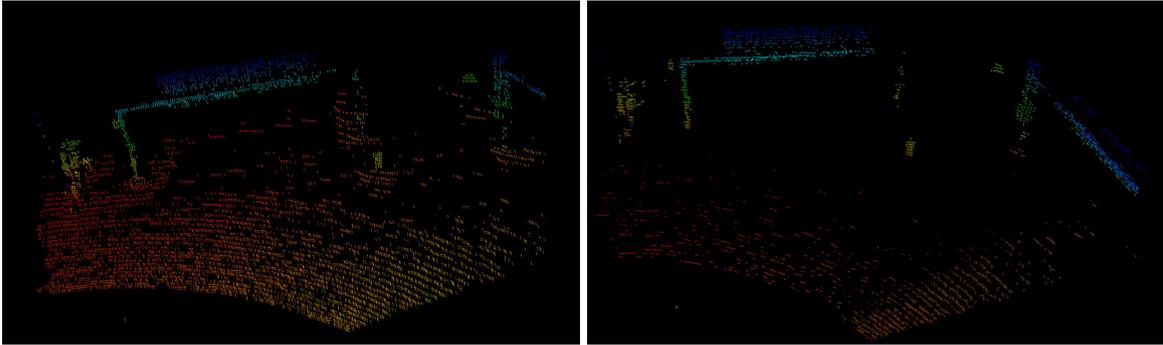


**Figure 5.28:** We separate the input point cloud into ground and objects (left), then filter them with different level of outlier removal (right).

Then, we combine them together to get the whole filtered point cloud, as the Figure 5.29 shows. This method works well with ground outlier removal while preserving the objects.

#### 5.6.4 Point Upsampling

As the last step, in order to improve the performance of the calibration, we applied an optimization based upsampling method to our point cloud [19]. We use the *Moving Least Squares*

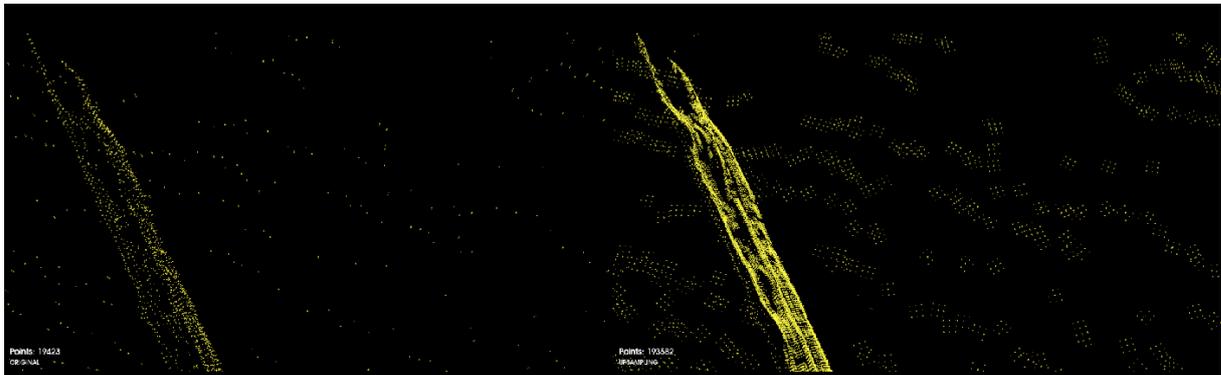


**Figure 5.29:** Before removal (left), after removal (right). The noise on the ground is filtered apparently.

(MLS) method of *PCL* to upsample the point cloud which works well on our S110 LiDAR scans. The method will generate points near the original points with the same intensity values (see Fig. 5.30). It will significantly refine the surface texture, as Figure 5.31 shows.



**Figure 5.30:** Before upsampling (left) and after upsampling (right).



**Figure 5.31:** Left: before upsampling with 49,835 points. Right: after upsampling with 493,949 points. About ten times more points were generated.

## 5.7 Adaptive Voxelization

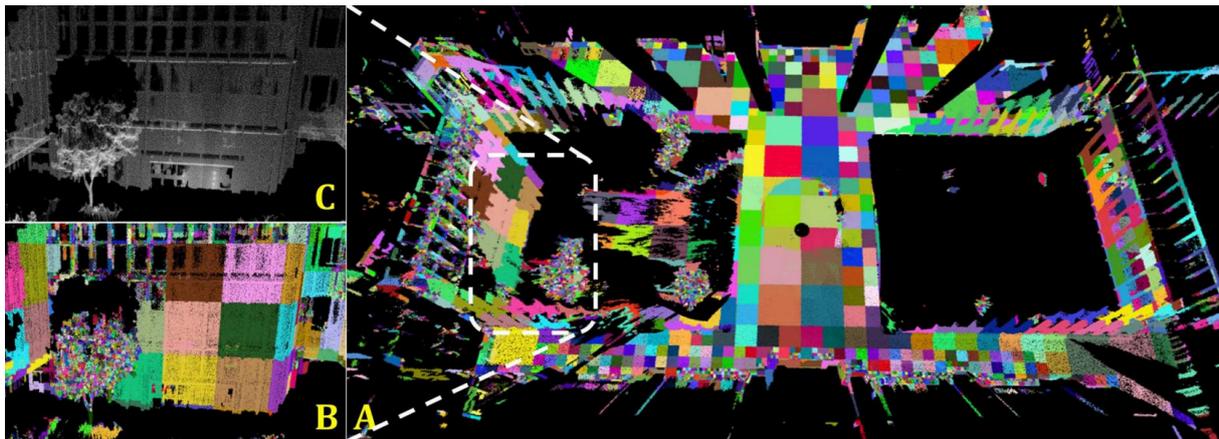
In this section we will make a short introduction about adaptive voxelization. In 5.7.1 we will provide a description about the algorithm. In 5.7.2 we will represent some results of edge pair building.

### 5.7.1 Adaptive Voxelization

We have already mentioned voxelization in Section 5.3. We determine the size of the voxel through pre-set parameters, and then divide the entire point cloud into voxels of equal size. In outdoor scenes, this approach may have some drawbacks. Take our S110 intersection as an example: objects such as gantry bridges and signal lights, have complex textures. There may be multiple planes in a small range. Thus relatively small voxels should be applied for those objects. On the other hand, the ground can almost be regarded as a plane, and its texture changes very little, so if small voxels are used, it will lead to a waste of calculation, because in the end these small planes will be merged into a large plane.

In order to save the calculation resources and shorten the convergence time while ensuring that the features of the complex surface are not lost, we introduce adaptive voxelization [51].

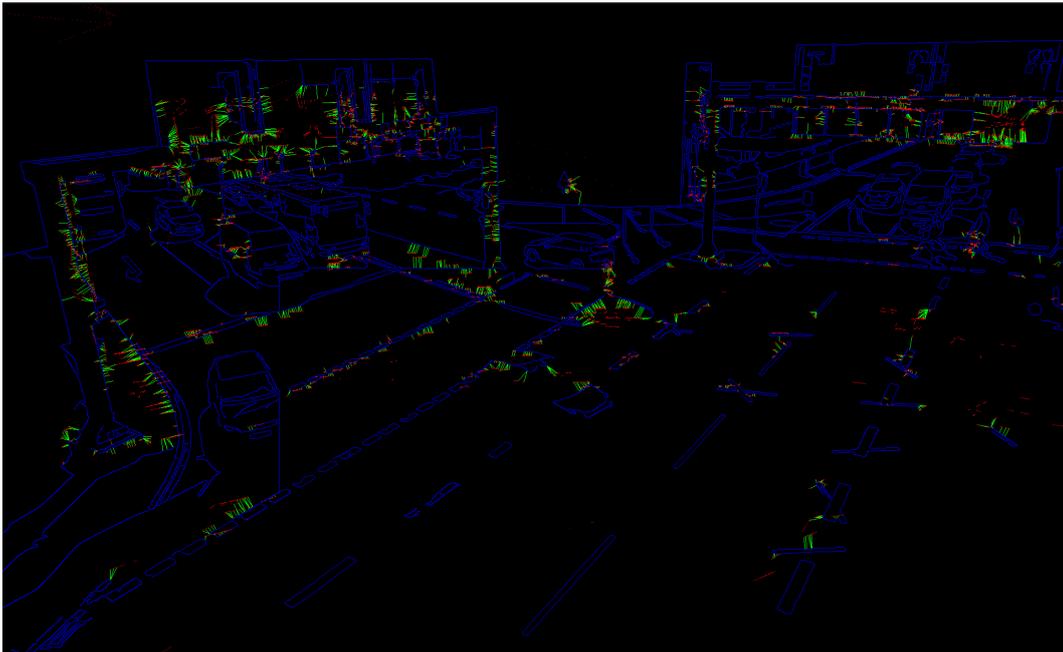
In the adaptive voxelization, the entire map is first cut into voxels with a preset size (usually large, e.g., 4 m). Then, for each voxel, if the contained points from all LiDAR scans roughly form a plane (by checking the eigenvalues), it is treated as a planar voxel. Otherwise, they will be divided into eight octants, where each will be examined again until the contained points roughly form a plane or the voxel size reaches the preset minimum lower bound. Moreover, the adaptive voxelization is performed directly on the raw point clouds, so no prior feature points extraction is needed [51]. Fig. 5.32 shows how adaptive voxelization works.



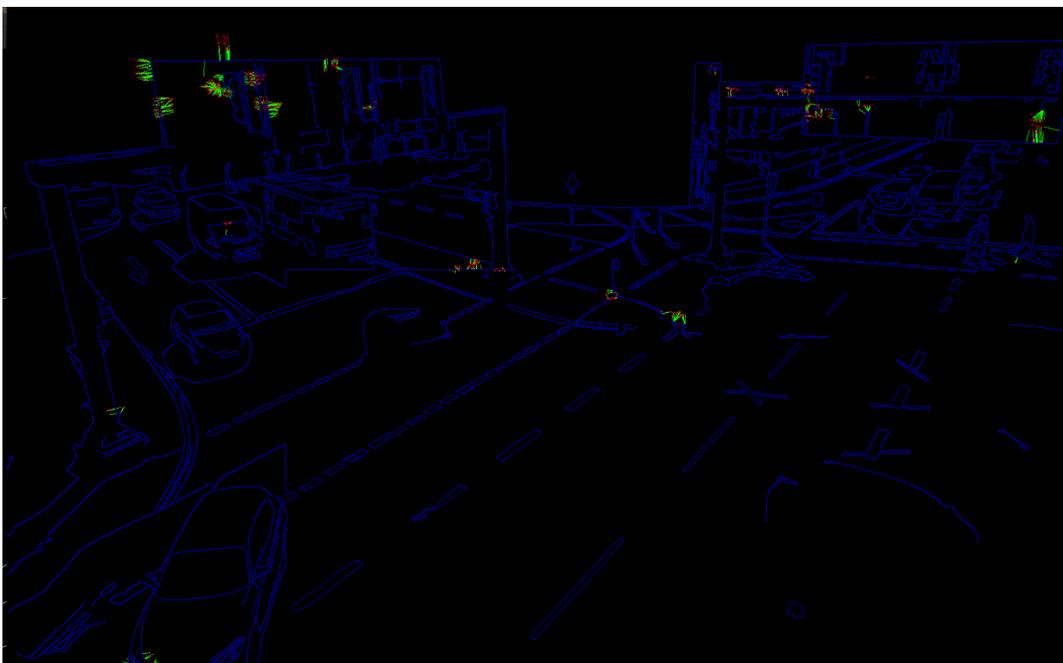
**Figure 5.32:** "(a) LiDAR point cloud segmented with the adaptive voxelization. Points within the same voxel are colored identically. The detailed adaptive voxelization of points in the dashed white rectangle could be viewed in (b) colored points and (c) original points. The default size for the initial voxelization is 4 m, and the minimum voxel size is 0.25 m." [51]

### 5.7.2 Result of Adaptive Voxelization

Figure 5.34 shows the comparison between fixed size voxelization and adaptive one.



**Figure 5.33:** Edges from Normal voxelization (this result was obtained with the fine tuned parameter), which costed 16.0128 seconds.



**Figure 5.34:** Edges from adaptive voxelization, costed 5.2306 seconds. This results is obtained from live data of LiDAR S110 south. With adaptive voxelization we can save much time but also lose some edges.

# Chapter 6

## Results

This chapter represents both qualitative and quantitative calibration results at the S110 intersection between different LiDARs and cameras. Section 6.1 will represent the calibration result between the south Ouster LiDAR (s110\_lidar\_south\_south) of the S110 intersection and the south Basler camera (s110\_camera\_basler\_south2\_8mm). Calibration results in different weathers conditions, daytime or traffic situations will be presented. Section 6.2 presents results between the north Ouster LiDAR (s110\_lidar\_ouster\_north) and the north Basler camera (s110\_camera\_basler\_south1\_8mm).

### 6.1 Result South LiDAR-South2 Camera

In this Section, we will represent both qualitative and quantitative calibration results between S110 south LiDAR and basler south2 camera in 3 different situations. The calibration will be based on the manual calibration result and use it as initial extrinsic.

The results are based on intrinsic and the initial extrinsic (shown in Equation 6.1 and 6.2).



**Figure 6.1:** The S110 south LiDAR is shown in the left red square. The S110 Basler south2 camera is shown in the right red square.

Intrinsic of Basler south2 camera:

$$K = \begin{bmatrix} 1320.7 & 0 & 975.038 \\ 0.0 & 1378.94 & 586.556 \\ 0.0 & 0.0 & 1.0 \end{bmatrix} \quad (6.1)$$

Initial extrinsic between S110 south LiDAR and basler south2 camera from manual calibration:

$$C = \begin{bmatrix} 0.641509 & -0.766975 & 0.0146997 & 1.99131 \\ -0.258939 & -0.234538 & -0.936986 & 1.31464 \\ 0.722092 & 0.597278 & -0.349058 & -1.50021 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (6.2)$$

The corresponding Euler angles in pitch (X axis), yaw (Y axis), and roll (Z axis) format:

$$r = [ 110.4319868 \quad 0.8422576 \quad 50.0903922 ] \quad (6.3)$$

### 6.1.1 Scene 1

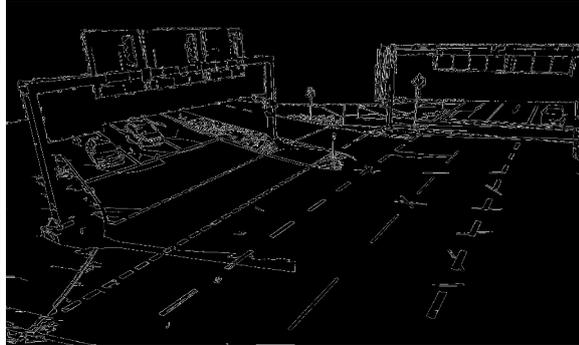
In this scene, there are only two LiDARs available (S110 Valeo north west LiDAR and S110 Ouster south LiDAR). In order to improve the texture of point clouds in the S110 Ouster south LiDAR, we register the S110 Valeo north west LiDAR to the S110 Ouster south LiDAR. It is worth mentioning that the ground point cloud is very sparse in this scenario, so we enabled upsampling in the calibration.

#### Qualitative Results

Figures 6.2 to 6.7 show the qualitative results of the automatic calibration and also the intermediate results: The raw image input from the camera, extracted canny edges and the residual map of the edge pairs.



**Figure 6.2:** Image input.



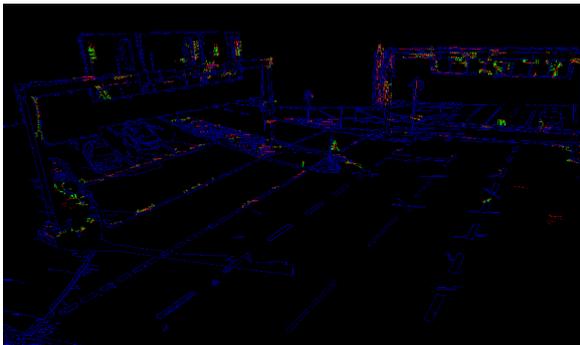
**Figure 6.3:** Extracted edge map.



**Figure 6.4:** Projection of the point cloud using the initial extrinsic.



**Figure 6.5:** Projection of the point cloud after rough calibration.



**Figure 6.6:** residuals from edge pairs.



**Figure 6.7:** Final calibration result.

**Figure 6.8:** Comparing the initial extrinsic in Fig. 6.4 and the calibration result in Fig.6.7, it is obvious that after optimization the gantry on the left and the traffic signs are converged. The optimizer also narrows the gap between Canny edges and LiDAR edges of lane markings.

### Quantitative Results

The extrinsic between the S110 south LiDAR and Basler south2 camera after automatic calibration is shown below:

$$K = \begin{bmatrix} 0.6377 & -0.770201 & 0.0113613 & 1.83058 \\ -0.237882 & -0.210944 & -0.948111 & 0.515441 \\ 0.732633 & 0.601908 & -0.317736 & -1.80184 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (6.4)$$

The corresponding Euler angles in pitch (X axis), yaw (Y axis), and roll (Z axis) format:

$$r = [ 108.5272968 \quad 0.6509706 \quad 50.3764282 ] \quad (6.5)$$

### 6.1.2 Scene 2

In this scene, there are three LiDARs available (S110 Valeo north west, S110 Ouster north and S110 Ouster south LiDAR). In order to improve the texture of point clouds in the S110 Ouster south LiDAR, we register the two other LiDARs to S110 Ouster south. As Figure 6.9 shows, in this scene there are a lot of shadows on the ground, and some of them are very close to the lane markings, which will lead to wrong correspondences between LiDAR edges and Canny edges. Hence, we enable shadow filtering here. We also enabled upsampling here.



**Figure 6.9:** The input image in test scene 2. There is heavy traffic in the scene, multiple vehicles are passing through the intersection. There are also shadows of the gantry bridge falling on the ground, which are near to the lane markings.

### Qualitative Results

Figure 6.10 to 6.15 show the qualitative result of the automatic calibration. After optimization the gantry on the left and the traffic signs are converged (see Figure 6.15). Comparing



Figure 6.10: Image input.

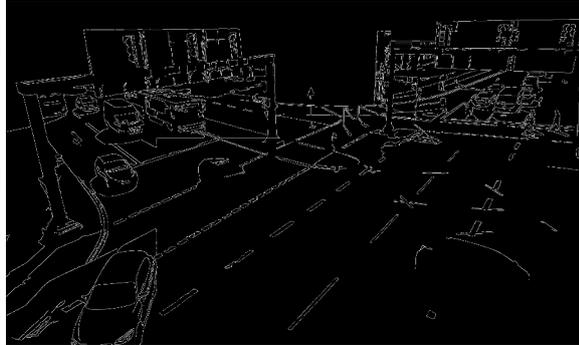


Figure 6.11: Extracted edge map.



Figure 6.12: Projection of the point cloud using the initial extrinsic.



Figure 6.13: Projection of point clouds after the rough calibration.

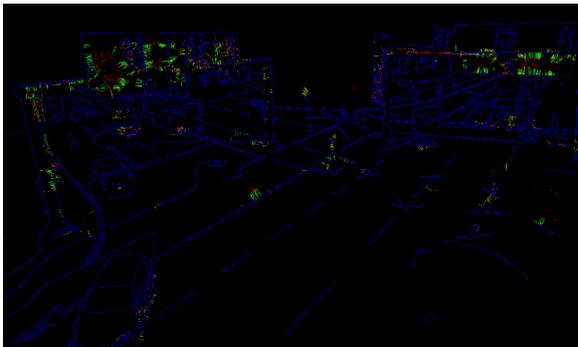


Figure 6.14: Visualization of residuals from edge pairs.



Figure 6.15: Final calibration results.

**Figure 6.16:** Calibration result. From 6.11 we can see the shadows of two gantry bridges on the ground are filtered and not extracted by the Canny edge detector. But the vehicle in dark color (in the bottom right of the image) is also influenced by the filter. Part of the car (the lower half of the hull) is gone.

the initial result in Figure 6.12 and our calibration result in Figure 6.15, the optimizer also narrows the gap between Canny edges and LiDAR edges of lane markings on the left. However, part of the lane markings on the right side are not converged.

### Quantitative Results

The extrinsic between S110 south LiDAR and Basler south2 camera after automatic calibration is shown below:

$$C = \begin{bmatrix} 0.617616 & -0.786437 & 0.00824804 & 2.68856 \\ -0.253955 & -0.209343 & -0.944289 & 0.830296 \\ 0.74435 & 0.581113 & -0.329013 & -1.98766 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (6.6)$$

The corresponding Euler angles in pitch (X axis), yaw (Y axis), and roll (Z axis) format are given below:

$$r = [ 109.21 \quad 0.472583 \quad 51.8562 ] \quad (6.7)$$

### 6.1.3 Scene 3

In this scene, there are three LiDARs available (S110 Valeo north west, S110 Ouster north and S110 Ouster south LiDAR). We register the two other LiDARs to S110 Ouster south. As Figure 6.17 shows, this scene was captured in the afternoon. Hence the sunlight is slanting towards the camera and leaving shadows of traffic signs on the ground. In addition, due to the backlight, the brightness of the back of the gantry board is similar to the ground. So we reselected the area for shadow filtering and adjusted the brightness threshold. It is also worth mentioning that in this scene, the LiDAR captured the point cloud of many vehicles, which will help the calibration of the ground. We also upsampled the point cloud in this scene.



**Figure 6.17:** The input image in test scene 3. The sunlight is slanting towards the camera and leaving shadows of traffic signs on the ground.

## Qualitative Results



Figure 6.18: Image input.

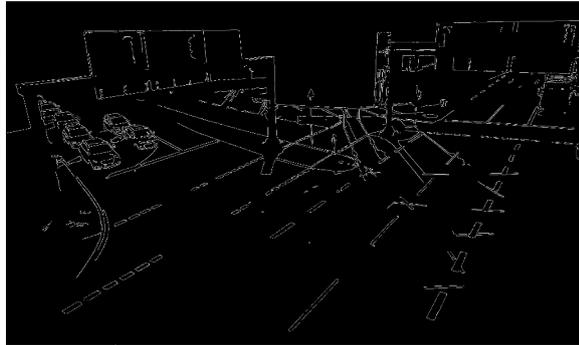


Figure 6.19: Extracted edge map.



Figure 6.20: Projection of the point cloud using the initial extrinsic.

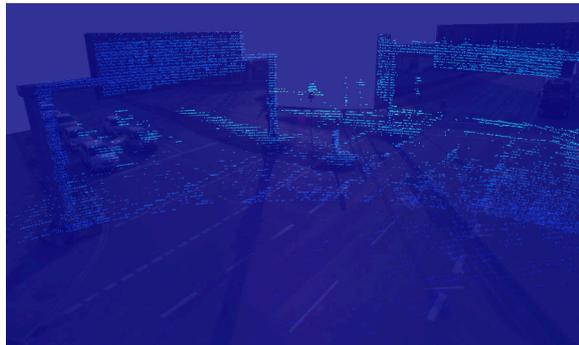


Figure 6.21: Projection of the point cloud after rough calibration.

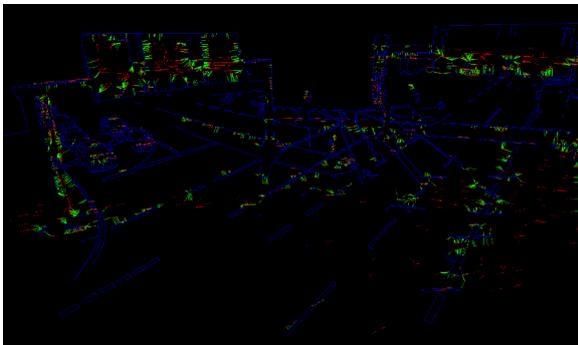


Figure 6.22: residuals from edge pairs.

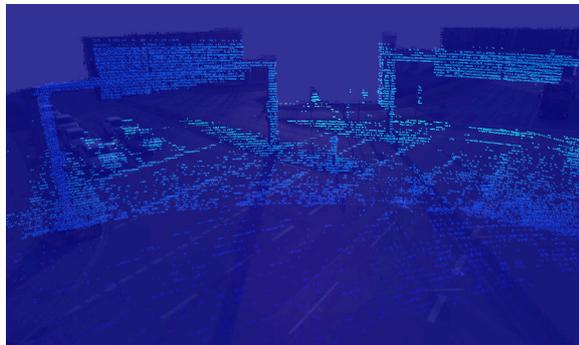


Figure 6.23: Final calibration result.

Figure 6.18 to 6.23 show the qualitative result of calibration. From the edge result in Figure 6.19 we can see that the filter works well and filtered the shadow inside the zone we selected. In the residual map (Figure 6.22) the correspondences are built correctly. The algorithm also built correspondences of vehicles. After optimization (Figure 6.23) the gantry, traffic signs and lane markings are converged. The algorithm also tries to converge the residual of vehicles. However, it stuck in the local minimum and didn't fit the vehicle on the left perfectly.

## Quantitative Results

The extrinsic between the S110 south LiDAR and Basler south2 camera after automatic calibration is shown below:

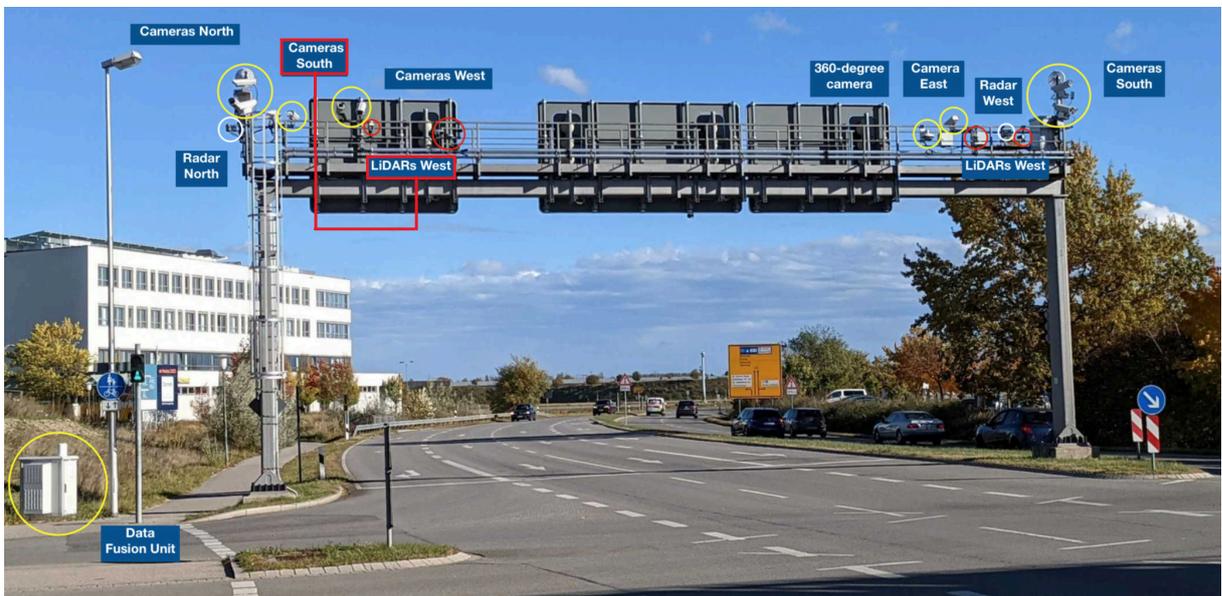
$$C = \begin{bmatrix} 0.619813 & -0.784603 & 0.0151551 & 2.70408 \\ -0.242379 & -0.209769 & -0.947233 & 0.569591 \\ 0.746381 & 0.583434 & -0.320189 & -1.48734 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (6.8)$$

The corresponding Euler angles in pitch (X axis), yaw (Y axis), and roll (Z axis) format are shown below:

$$r = [ 108.677 \quad 0.868357 \quad 51.6923 ] \quad (6.9)$$

## 6.2 Result North LiDAR-South1 Camera

In this section, we will present both qualitative and quantitative calibration results of the S110 north LiDAR to basler south1 camera. The calibration will be based on the manual calibration result and use it as initial extrinsic.



**Figure 6.24:** The Basler south1 camera is marked in the left red square. The S110 north LiDAR is marked in the right red square.

The results are based on the intrinsic and the initial extrinsic values:

Intrinsic of Basler south1 camera:

$$K = \begin{bmatrix} 1312.68 & 0 & 930.852 \\ 0.0 & 1325.01 & 613.035 \\ 0.0 & 0.0 & 1.0 \end{bmatrix} \quad (6.10)$$

The initial extrinsic between the S110 north LiDAR and Basler south1 camera from manual calibration:

$$C = \begin{bmatrix} -0.35982 & -0.932866 & 0.0170408 & -0.212078 \\ -0.446603 & 0.156168 & -0.880998 & 0.400684 \\ 0.819192 & -0.324611 & -0.472813 & -0.0976544 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (6.11)$$

The corresponding Euler angles in pitch (X axis), yaw (Y axis), roll (Z axis) format are shown below:

$$r = [ 118.2215211 \quad 0.9764078 \quad 111.0923517 ] \quad (6.12)$$

### 6.2.1 Scene 1

In this scene, we only use the point cloud from S110 Ouster north. We enabled upsampling in this calibration. As Figure 6.25 shows, there are no shadows on the ground, and the background is also within the range of the LiDAR (Figure 6.27). Hence, we didn't enable shadow filtering and background cropping.

#### Quantitative Results

The extrinsic between the S110 south LiDAR and Basler south2 camera after automatic calibration is shown below:

$$C = \begin{bmatrix} -0.380191 & -0.924759 & 0.0165841 & 0.213112 \\ -0.415685 & 0.154826 & -0.896234 & -0.232763 \\ 0.826233 & -0.347633 & -0.443272 & -0.37722 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (6.13)$$

The corresponding Euler angles in pitch (X axis), yaw (Y axis), and roll (Z axis) format are shown below:

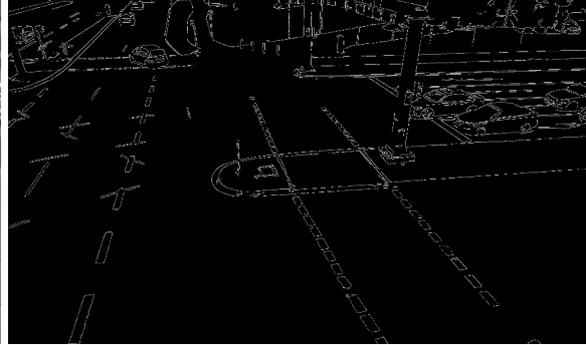
$$r = [ 116.317 \quad 0.950244 \quad 112.349 ] \quad (6.14)$$

### Qualitative Results

Figure 6.25 to 6.30 show the qualitative results of the automatic calibration and also the intermediate result.



**Figure 6.25:** Image input.



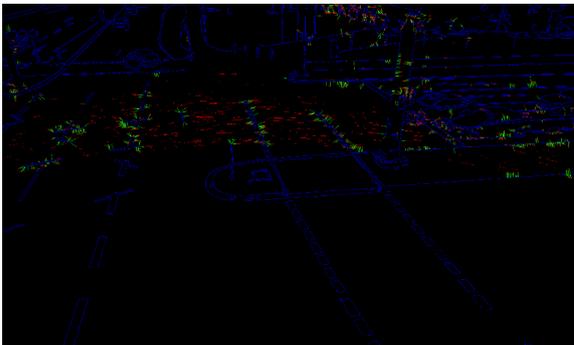
**Figure 6.26:** The extracted edge map.



**Figure 6.27:** Projection of the point cloud using the initial extrinsic.



**Figure 6.28:** Projection of the point cloud after rough calibration.



**Figure 6.29:** Visualization of residuals from edge pairs.



**Figure 6.30:** Final calibration results.

**Figure 6.31:** Comparing the initial extrinsic in Fig. 6.27 and the calibration result in Fig. 6.30, the calibration only improves the convergence of the gantry on the left, since the manual calibration result is already good enough (see Figure 6.30).

### 6.2.2 Scene 2: Initial Extrinsic with Offset and Heavy Traffic

The manual calibration result is already good enough and the optimization of automatic calibration doesn't improve a lot. In order to test the performance of automatic calibration in

s110 north LiDAR and south2 camera, we manually add a random offset to the initial extrinsic, namely:

- `rotation_euler_yaw_pitch_roll_offset`:  $\pm(0.5,1)$  degree
- `translation_offset_x_y_z`:  $\pm(0.5,1)$  meters

We also select the scene with more vehicles in this FoV, for example, a truck in the middle of the image, as Figure 6.32 shows.



**Figure 6.32:** We select a time slot in that a truck and a SUV drives through the scene to test the automatic calibration under heavy traffic conditions.

### Quantitative Results

The extrinsic between the S110 north LiDAR and Basler south1 camera after automatic calibration:

$$C = \begin{bmatrix} -0.365152 & -0.930737 & 0.0198163 & -0.395592 \\ -0.444471 & 0.155595 & -0.882177 & 0.292076 \\ 0.817991 & -0.330936 & -0.470501 & -0.934264 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (6.15)$$

The corresponding Euler angles in pitch (X axis), yaw (Y axis), and roll (Z axis) format are shown below:

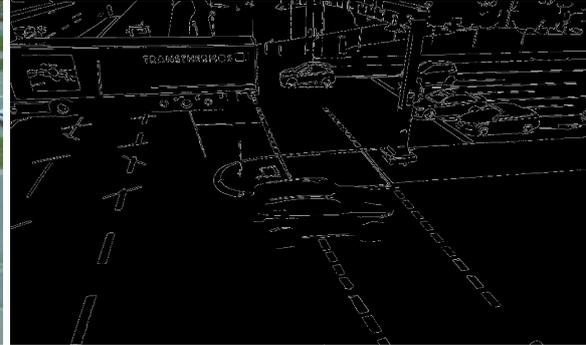
$$r = [ 118.073 \quad 1.13547 \quad 111.421 ] \quad (6.16)$$

### Qualitative Result

Figure 6.33 to 6.38 show the qualitative result of automatic calibration. As we can see from Figure 6.37, the algorithm successfully builds the edge correspondences between the point clouds and the canny edges of the truck, and fixes the error we manually added to the initial extrinsic. The gantry and truck are converged and the rotation matrix is optimized (see Figure 6.38).



**Figure 6.33:** Image input.



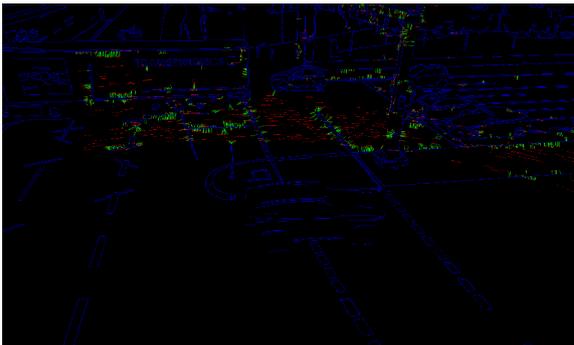
**Figure 6.34:** Extracted edge map.



**Figure 6.35:** Projection of the point cloud using the initial extrinsic.



**Figure 6.36:** Projection of the point cloud after rough calibration.



**Figure 6.37:** Visualization of residuals from edge pairs.



**Figure 6.38:** Final calibration result.

**Figure 6.39:** In this case, the trucks in the FoV reflect lots of point clouds, which is also calibrated as 6.38 shows. The trees in the background are also converged after calibration.

# Chapter 7

## Evaluation and Discussion

In this chapter, our automatic calibration model will be evaluated with a variety of real world data. In total, 3 LiDARs (s110 Ouster north, s110 Ouster south and S110 Valeo north west) and 2 cameras (s110 Basler south1 and south2) will be used to test our model. The algorithm will be tested under different conditions (scenery, traffic situation, shadows, quality of point cloud etc.). In Section 7.1 and Section 7.2 different levels of offset are applied to the initial extrinsic (manual calibration results are taken as initial) and the convergence will be tested. In Section 7.1 the calibration between s110 Ouster south LiDAR and s110 Basler south2 camera is tested. In Section 7.2 the s110 Ouster north LiDAR and the s110 Basler south1 camera are used for the limit test.

### 7.1 Limit Test - South LiDAR and South2 Camera

In this section, the calibration between s110 Ouster south LiDAR and s110 Basler south2 camera is tested. Rotation and translation offsets are added to the initial extrinsic. The model will be evaluated with maximal 40 iterations. In Section 7.1.1 rotation Euler angle shifts are applied to the initial extrinsic. In Section 7.1.2 translation offsets are added to the initial extrinsic. In Section 7.1.3 we make a brief summary of our limit test for this scene.

Both quantitative and qualitative results are presented below. The convergence of calibration parameters are visualized with PYR-XYZ loss (L1 loss), which is obtained through directly calculating the difference between automatic and manual calibration results of the six extrinsic parameters (pitch, yaw, roll Euler angles in degree and x,y,z translation offsets in meters). Besides that, the convergence of normalized optimization cost (which is given below in Eq. 7.1 [50]) during the calibration is also represented:

$$J = \frac{1}{N_{match}} r^T (J_w \Sigma J_w)^{-1} r \quad (7.1)$$

Qualitative calibration results and PYR-XYZ loss of each test are analyzed together. It is worth noting that since our calibration is performed on the real-live data, there is no such a real "ground truth" value available for the evaluation. Instead, the manual calibration results are taken as the benchmark for the calculation of loss function. Therefore, although many auto-calibration results do not converge to manual results(the benchmark), it may indicates a better calibration result than manual one is obtained.

In this scene, the lighting conditions are good. As Fig. 7.1 shows, although there is a shadow of the gantry on the ground, it does not fall into the area of interest. So we didn't enable

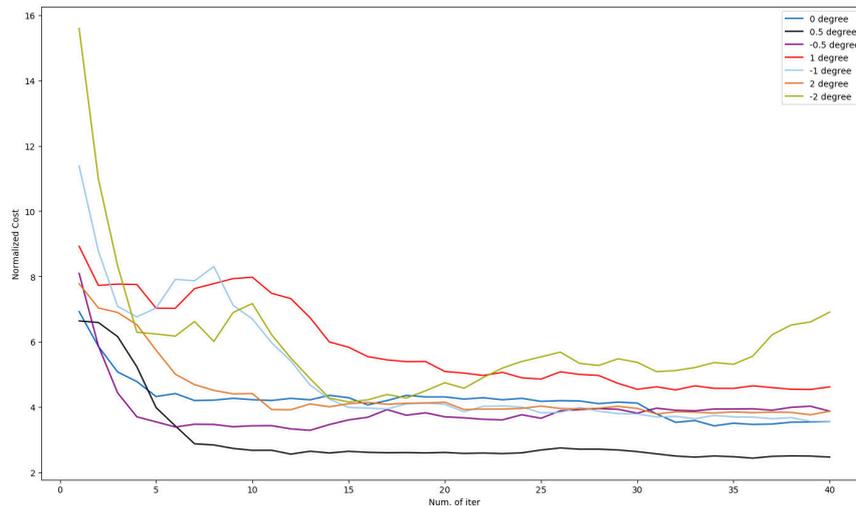
shadow filtering here. On the other hand, the input point cloud is relatively sparse (Fig. 7.1 right), and there is little noise on the ground. Under these conditions the road markings on the ground are easy to recognize. Due to the sparsity of ground point clouds, we employed upsampling and scattering to increase the density of point clouds.



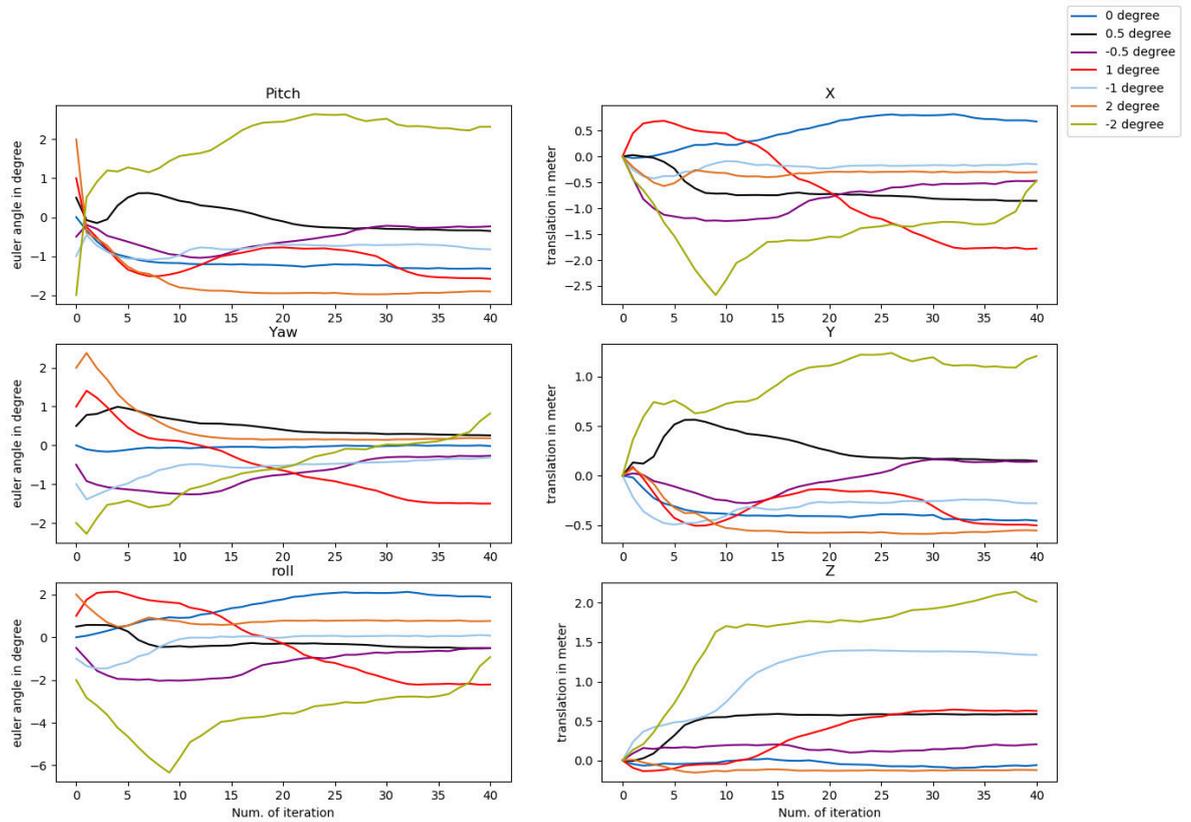
**Figure 7.1:** Input image and point cloud. Left: the ground is well lit, shadows have less effect on the calibration, and there are few vehicles in the FoV. Right: the point cloud is relatively sparse, but on the other hand it contains less ground noise, which is a favorable condition for our calibration.

### 7.1.1 Rotation Limit Test

Limit test with shifts in three Euler angles is shown in this section. We rotate the initial Euler angle in the same direction within the range of  $[-2, 2]$  degrees. Fig. 7.2 shows the normalized optimization cost and Fig. 7.3 represents the PYR-XYZ loss.



**Figure 7.2:** Convergence of normalized optimization cost. The x axis represents the number of iterations and y the normalized cost. We tested the optimization performance in the range of  $[-2, -1, -0.5, 0, 0.5, 1, 2]^\circ$  with rotation shift. The figure shows that our model can converge within 40 iterations within the rotation shift in most cases. It is worth noting that when the rotation reaches  $-2^\circ$ , because the shift is too large, the cost starts to diverge again in the later optimization. Most of them converge after about 25 iterations.



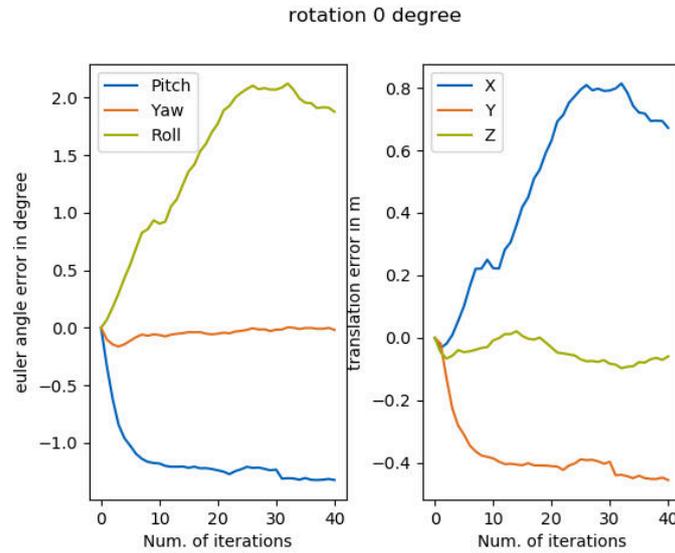
**Figure 7.3:** Convergence of the **PYR-XYZ Loss** during the optimization. The Y axis represents the corresponding difference to the manual calibration result (Euler angle in degree and translation distance in meter). We tested the optimization performance in range of  $[-2, -1, -0.5, 0, 0.5, 1, 2]^\circ$  with rotation shift on all three Euler angles. In the **Pitch** direction, all cases except  $-2^\circ$  converged to similar results. In the **Yaw** direction, except for  $+1^\circ$ , all cases basically converge to the manual results. In the **Roll** direction, most of them going towards the manual result. It is worth noting that roll with an offset of  $+1^\circ$  converges to a different result, and  $-2^\circ$  converges to a result close to manual even though the initial shift is large. In most cases in the **X** direction, it converges to a result of  $-0.5$  m. In the **Y** direction, the case of  $-2^\circ$  converges to  $1.5$  m, and  $+0.5^\circ$  converges to a result close to the manual one. All other cases converge to  $-0.5$  m. There are four cases in the **Z** direction that are different to the manual results. Overall, the convergence of the model under different initial conditions shows consistency.

### Qualitative Result and Analysis

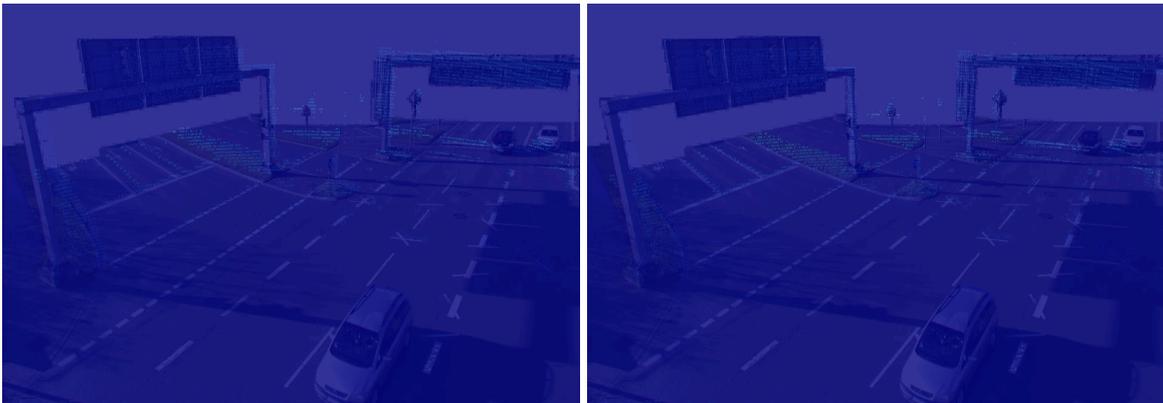
Next, there are some representative examples of qualitative results. Including good results as well as the case in which our model perform not so well. All those cases are analyzed with the loss function.

#### Case 1: $0^\circ$ initial rotation shift:

Fig. 7.4 shows the PYR-XYZ Loss and Fig. 7.5 shows the qualitative results.



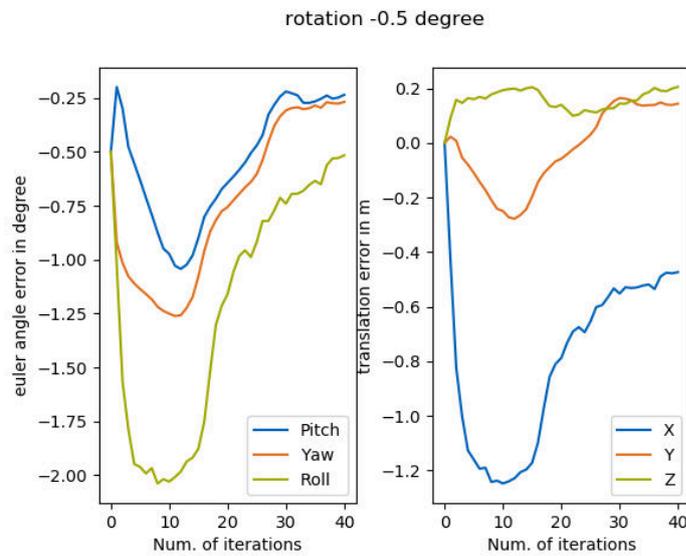
**Figure 7.4:** The PYR-XYZ Loss of initial rotation shifts with  $0^\circ$ . The X axis represents the number of iterations. The Y axis represents the corresponding difference to the manual calibration result. Although the manual result is used as the initial extrinsic, the final optimization result does not converge to the manual result, but a different local minimum.



**Figure 7.5:** Qualitative results of initial rotation shifts with  $0^\circ$ . On the left the projection is displayed using the **initial extrinsic**. The right figure shows the **optimization result**. We can see from the figures that although the result of convergence is different from the manual result, the result of automatic calibration is actually better. The automatic calibration is better in fitting the left gantry, triangle traffic sign and the lane markings on the left.

**Case 2:  $-0.5^\circ$  initial rotation shift:**

Fig. 7.6 shows the PYR-XYZ Loss and Fig. 7.7 shows the qualitative results.



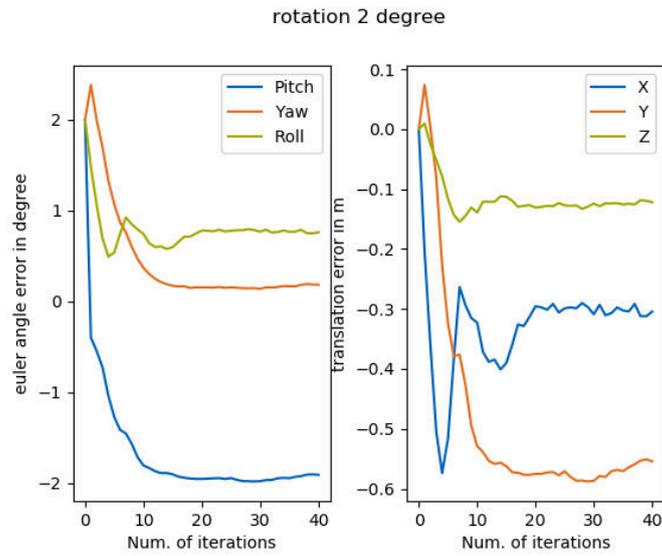
**Figure 7.6:** PYR-XYZ Loss of initial rotation shifts with  $-0.5^\circ$ . The X axis represents the number of iterations. The Y axis represents the corresponding difference to the manual calibration result. The optimizer starts from a high error to the manual benchmark and converges with a lower error.



**Figure 7.7:** Qualitative results of initial rotation shifts with  $-0.5^\circ$ . Left: projection result using the **initial extrinsic**. Right: **optimization result**. The initial extrinsic (shifted) gives a bad projection. From the left image we can see that the lane markings under the left gantry are shifted and the traffic signs are not fitted. But after automatic calibration it improves the projection, most of objects are fitted after calibration.

### Case 3: 2° initial rotation shift:

Fig. 7.8 shows the PYR-XYZ Loss and Fig. 7.9 shows the qualitative results.



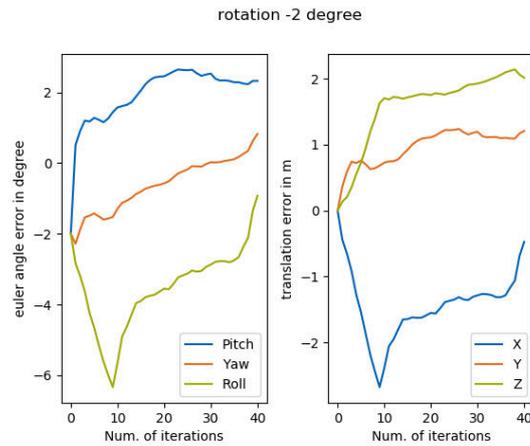
**Figure 7.8: PYR-XYZ Loss** of initial rotation shifts with 2°. The X axis represents the number of iterations. The Y axis represents the corresponding difference to the manual calibration result. The final optimization result does not converge to the manual one, but a different value. Most of the parameters converge after about 20 iterations and remain at this value until the end.



**Figure 7.9:** Qualitative result of initial rotation shifts with 2°. Left: **initial extrinsic**. Right: **optimization result**. The automatic calibration has a very good performance under this bad initial extrinsic. The rotation shift of the initial extrinsic leads to a projection with a large shift of the lane markings and rotation of the right gantry (in the left figure). After the optimization the rotation matrix is fixed and the lane markings are converged.

**Case 4:  $-2^\circ$  initial rotation shift:**

Fig. 7.10 shows the PYR-XYZ Loss and Fig. 7.11 shows the qualitative results.



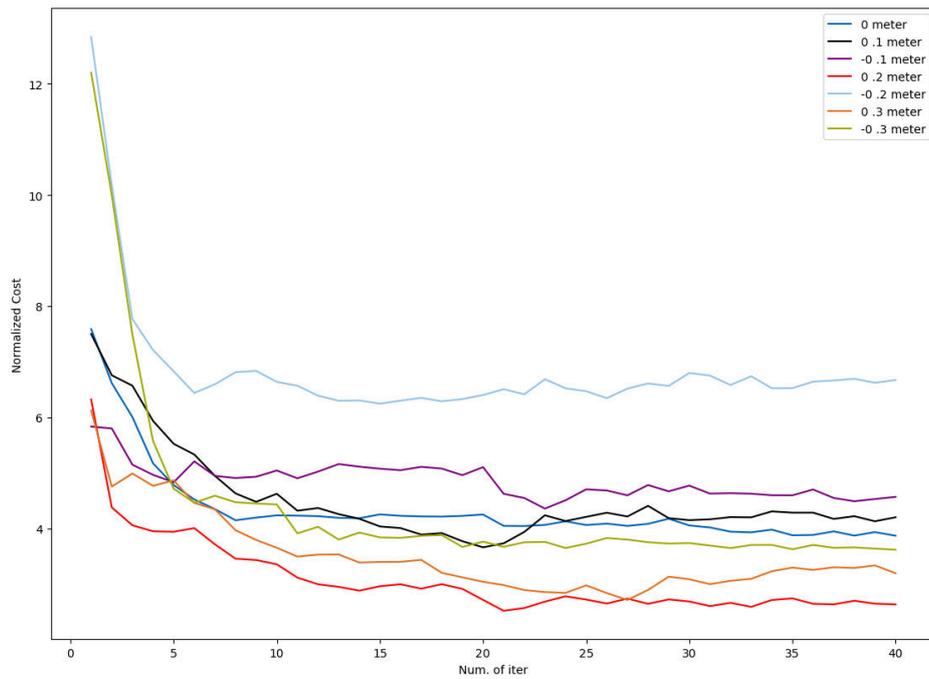
**Figure 7.10: PYR-XYZ Loss** of initial rotation shifts in  $-2^\circ$ . X axis represent number of iteration. Y axis represent corresponding difference to the manual calibration result. The optimization start from a very bad initial extrinsic. In Roll and X direction the error increased dramatically, and turning back at about 10 iterations. Finally they converge to near the manual results. However, the Pitch and Z direction converges to a different value to the benchmark. We also try with 80 iterations, but the performance didn't show much difference than this. Hence the  $-2^\circ$  is beyond the limit of the algorithm.



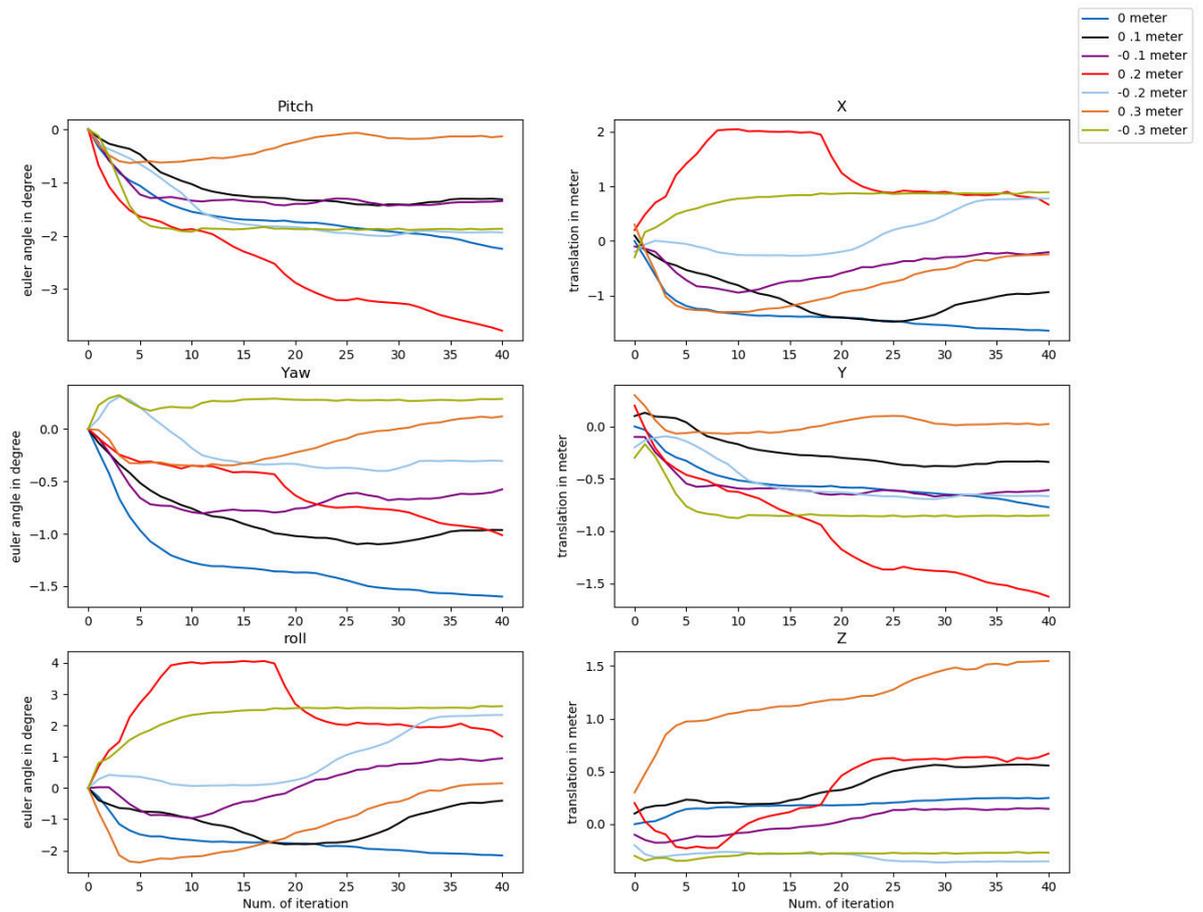
**Figure 7.11: Qualitative results** of initial rotation shifts with  $-2^\circ$ . The initial extrinsic is strongly shifted and has a very bad projection result (Left figure). As a consequence, from the first projection our model can extract much less edge pairs than other cases. But it still improves the rotation and fitted the gantry on the right as well as the traffic signs in the middle. Part of the lane markings are also fitted (right figure). We also tried with more iterations, but due to the wrong correspondences from the initial projection, the algorithm oscillates around the local minimum shown in the right figure.

### 7.1.2 Translation Limit Test

In the following we show limit tests with offsets in 3 directions of translation. We add translation offsets to the initial translation vector within the range of  $[-0.3, 0.3]$  meters. Fig. 7.12 shows the optimization cost and Fig. 7.13 represents the PYR-XYZ loss.



**Figure 7.12:** Convergence of normalized optimization cost. The X axis represents the number of iterations. The Y axis represents the normalized cost. We tested the optimization performance under level of  $[-0.3, -0.2, -0.1, 0, 0.1, 0.2, 0.3]$  meters of translation offsets, the offsets are applied to all 3 directions. It can be seen from the figure that our model can converge within 40 iterations within the translation offsets in range of  $[-0.3, 0.3]$  meters. Notice that the cost of -0.2 meter converges at a higher value than others, we will study this case in the upcoming parts.



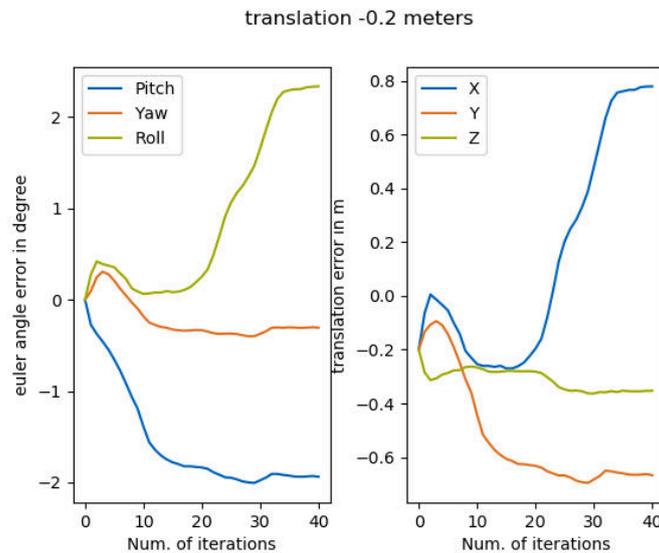
**Figure 7.13:** Convergence of the **PYR-XYZ loss** during the optimization. The X axis represents the number of iterations. The Y axis represents the corresponding difference to the manual calibration result. The results are presented in six dimensions separately: Pitch, Roll, Yaw, X, Y, and Z. It is worth noting that our model rarely converges to a manual calibration result. Besides that, because different offsets are added to the initial value of the translation vector, the rotation parameters in each case converge to different values. Even though the calibration results are different to the manual ones, the most of the qualitative results (projection) with  $[-0.3, 0.3]$  meters of initial offsets are better than manual calibration. In another word, they achieve a better extrinsic than manual result. On the other hand, We also try with offsets beyond this range (e.g.,  $+0.5\text{m}$ ,  $+2\text{m}$ ), but didn't achieve good qualitative results. So in this scene the limit of our model is  $[-0.3, 0.3]$  m.

### Qualitative Results and Analysis

In the following we show some qualitative results including examples with a good performance as well as the case in which our model performs not so well. The initial translation of 0 m (same to  $0^\circ$ ) has been discussed in the previous section, it will not be presented here.

#### Case 1: -0.2 meters initial rotation shift:

Fig. 7.14 shows the PYR-XYZ loss and Fig. 7.15 shows the qualitative results.



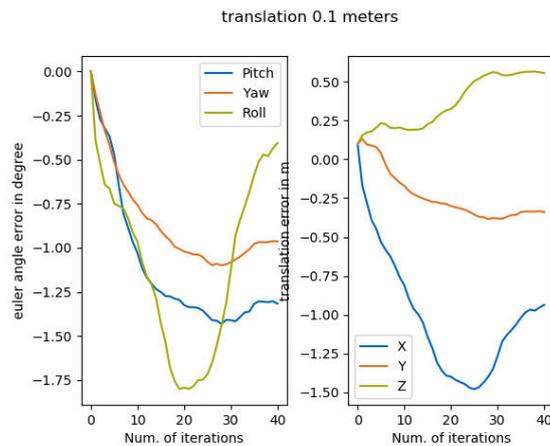
**Figure 7.14:** The **PYR-XYZ loss** of translation offsets with **-0.2 meters**. The X axis represents the number of iterations. The Y axis represents the corresponding difference to the manual calibration result. The final optimization result does not converge to the manual result. Especially in **Roll** and **X** direction.



**Figure 7.15:** Qualitative results of initial translation offsets in **-0.2 meters**. Left: **initial extrinsic**. Right: **optimization result**. It is not difficult to see that although the result of convergence is different from the manual result, the automatic calibration actually converged to a better extrinsic. The gantry is completely fitted and the lane markings converge better than the initial projection (left figure).

**Case 2: 0.1 meters initial rotation shift:**

Fig. 7.16 shows the PYR-XYZ loss and Fig. 7.17 shows the qualitative results.



**Figure 7.16:** The **PYR-XYZ loss** of translation offsets with **0.1 meters**. The X axis represents the number of iterations. The Y axis represents the corresponding difference to the manual calibration result. From the figure we can see that the Roll loss increases dramatically in the negative direction to the benchmark, but turned back after 20 iterations. Finally, it converges to  $-0.5^\circ$ . In total, the final optimization result does not converge to the manual result.



**Figure 7.17:** Qualitative results of initial translation offsets with **0.1 meters**. With 0.1 m offset to the initial translation, the point clouds are projected in front of the gantry in the image. Besides that, the lane markings are projected above the ground (left figure). After calibration the lane markings are fitted and the gantry bridge is converged. However, the lane markings on the right side didn't converge well. This is because the ground contains much fewer LiDAR points than the gantry. As a consequence, it will build less edge pairs of lane markings compared to the gantry. When the point cloud of the gantry is far from the Canny edges, the optimizer will give priority to optimize the gantry, because they contribute more to the cost function (right figure).

### 7.1.3 Short Summary

In general, in this limit test, the model has shown sufficient robustness with the shifts in the initial extrinsic within the range of  $[-2,2]^\circ$  and  $[-0.3,0.3]$  meters. In most cases the algorithm converged to an excellent result.

But in the case of  $-2^\circ$ , the model did not converge to the ideal result, mainly because the excessive displacement of the left gantry made the algorithm unable to establish enough edge pairs. This result shows that the voxel based calibration method has a requirement for the initial extrinsic. If the distance between the point cloud and the target is too far, it will not be capable to establish correct and sufficient edge pairs. It also shows that in outdoor calibration, it is very important to make sure that the edge pairs are evenly distributed throughout the camera image.

## 7.2 Limit Test - North LiDAR and South1 Camera

In this section, the calibration between the s110 Ouster north LiDAR and s110 south1 camera is tested. The model will be tested with maximal 40 iterations. In Section 7.2.1 rotation Euler angel shifts are applied to the initial extrinsic. In Section 7.2.2 different levels of translation offsets are applied. In Section 7.2.3 we draw a brief conclusion.

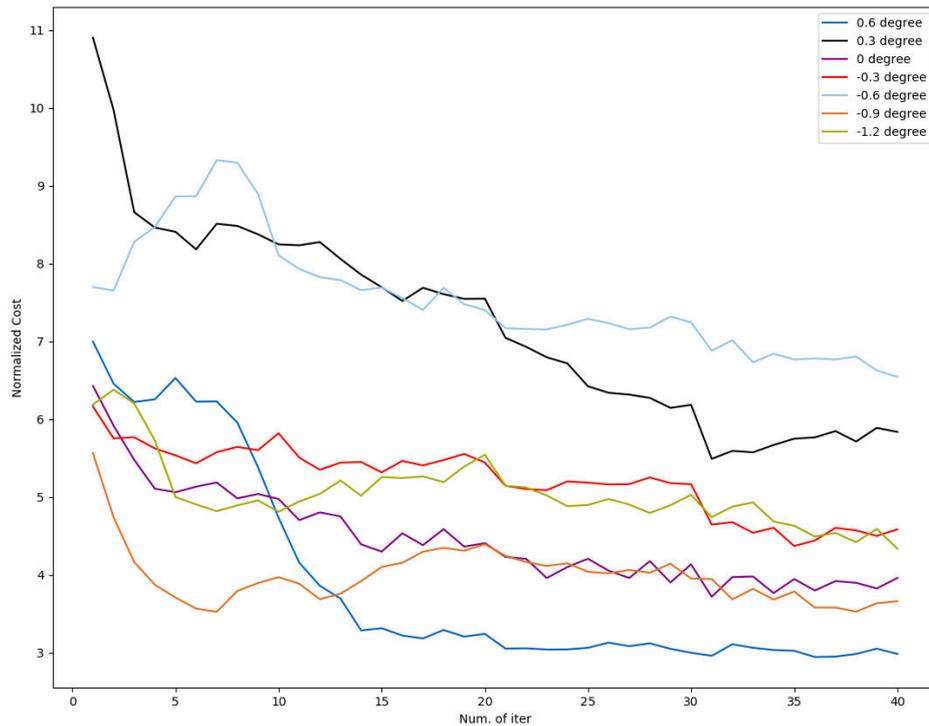
In this scene, there is no shadow on the ground. So we didn't enable shadow filtering. There is heavy traffic in this scene, lots of vehicles are waiting for the traffic light. Sometimes there are even some trucks driving through the intersection. Hence we can test the performance of our model in the complex environment. The input point clouds are more dense than in the last experiment. They also contain a lot of noise on the ground. As a consequence, the lane markings on the ground are difficult to be distinguished from the noise. We use a strong filter parameter for noise filtering here. In order to get more features from the gantry bridge, we still upsample and scatter the point cloud.



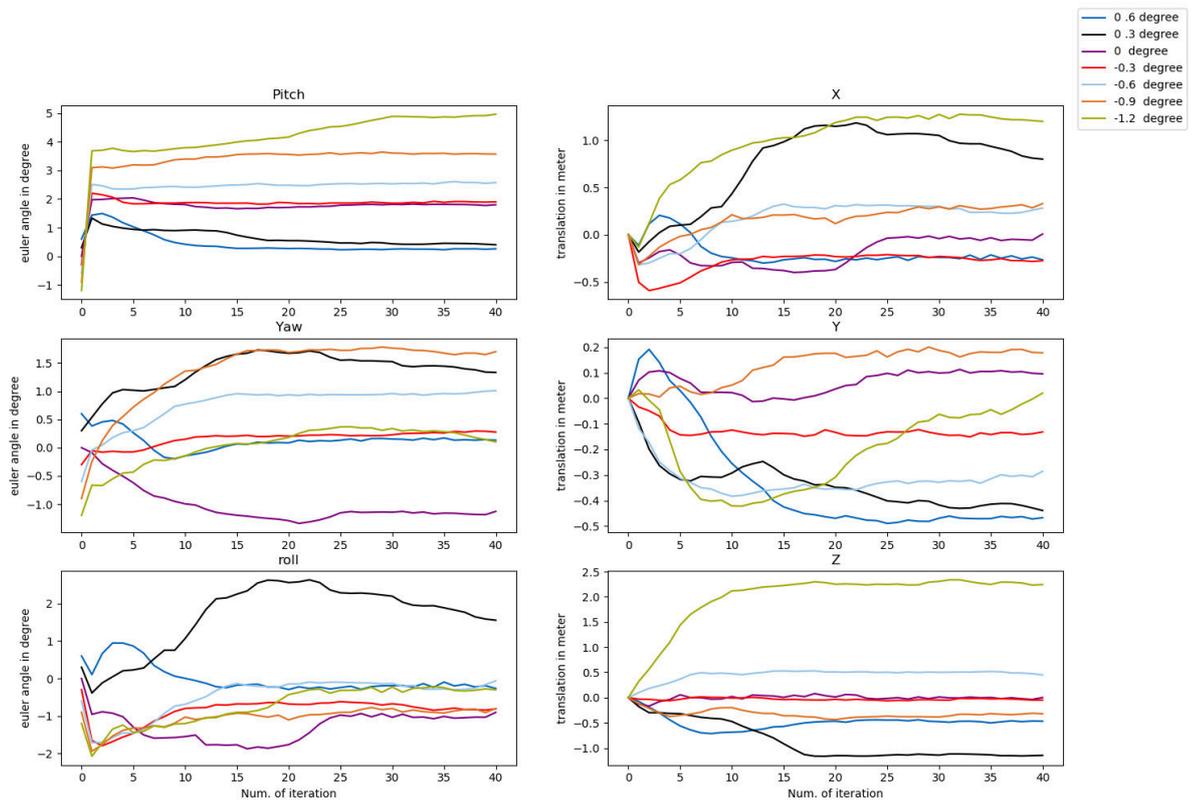
**Figure 7.18:** Input images and point clouds. The ground is well lit and without shadows. The traffic situation is complex and sometimes there are trucks in the scene (left figure). There is lots of noise on the ground and the lane markings are difficult to be identified. As a consequence, vehicles contribute much more features to the calibration instead of lane markings.

### 7.2.1 Rotation Limit Test

In the following we show the results of our rotation limit test. We rotate the initial Euler angles in the same direction within the range of  $[-1.2, 0.6]^\circ$ . In this scene, we didn't use background filtering. For the point cloud, there is only one LiDAR available. We adopted pre-processing including cropping, scattering, outlier removal (noise filtering) and upsampling. Fig. 7.19 shows the normalized optimization cost and Fig. 7.20 represents the PYR-XYZ loss.



**Figure 7.19:** Convergence of normalized optimization costs. The X axis represents the number of iterations. The Y axis represents the normalized cost. We tested the optimization performance under levels of  $[-1.2, -0.9, -0.6, -0.3, 0, 0.3, 0.6]^\circ$  of rotation shift. It can be seen from the figure that our model can converge within 40 iterations with the rotation shift in  $[-1.2, 0.6]^\circ$ .



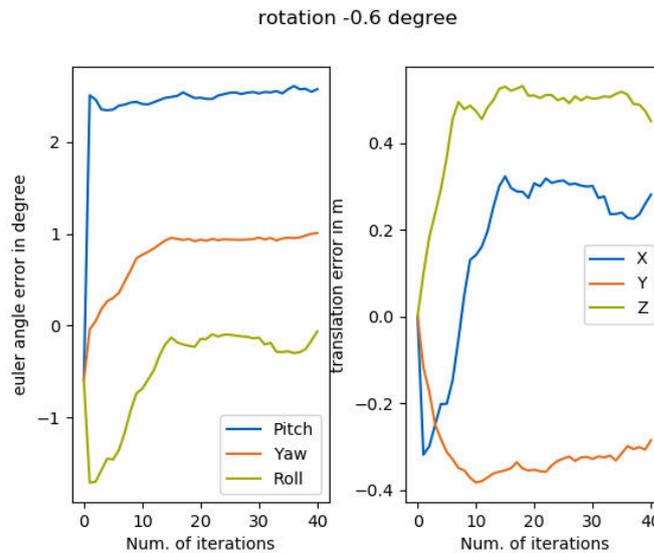
**Figure 7.20:** Convergence of the **PYR-XYZ loss** during the optimization. The X axis represents the number of iterations. The Y axis represents the corresponding difference to the manual calibration result. We tested the optimization performance in the range of  $[-1.2, -0.9, -0.6, -0.3, 0, 0.3, 0.6]^\circ$  with rotation shift. The rotation shifts are applied to all three Euler angles. The results are presented in six dimensions separately: Pitch, Roll, Yaw, X, Y, and Z. Like the test in the last section, the automatic calibration results converged to different values compared with manual results. It is worth noting that the converged result with an initial shift of  $-1.2^\circ$  is significantly different from others in **Pitch**, **X** and **Z** dimensions. And its projection results also show this difference. We will analyze this situation below.

### Qualitative Results and Analysis

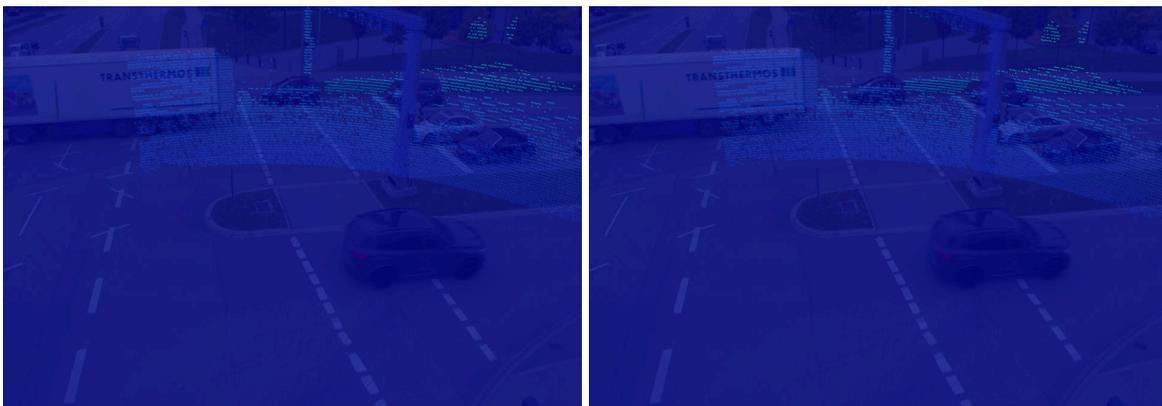
Most of our automatic calibration results did not converge to the results from manual calibration, but came close. Here we analyze two special cases in the test. One is the situation where there is a special reference object coming into the FoV. In this case a truck comes into the view and reflects a lot of points. Our automatic calibration module achieved good results in this case. Another case is the performance of the algorithm under the shift of  $-1.2^\circ$ , which happens to be the lower boundary of our test. In this test our model achieves different results than in other cases.

#### Case 1: $-0.6^\circ$ initial rotation shift:

In this case there is a truck on the left and the model has build correspondences with truck edges. Our model works well in this situation.



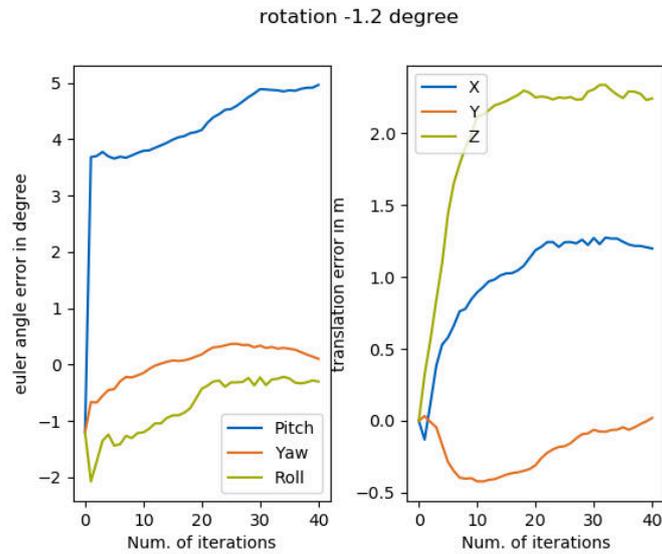
**Figure 7.21: PYR-XYZ loss** of initial rotation shifts with  $-0.6^\circ$ . The X axis represents the number of iterations. The Y axis represent the corresponding difference to the manual calibration result (Euler angle in degree and translation distance in meter). The final optimization result does not converge to the manual result.



**Figure 7.22: Qualitative results** of initial rotation shifts with  $-0.6^\circ$ . Left: **initial extrinsic**. Right: **optimization result**. In this scenario, our calibration model works well despite a truck coming into the view. Compared with the initial projection with  $-0.6^\circ$  shift added (left figure), our model is not affected by the truck, and still successfully converged the trees in the background and the gantry to their corresponding point cloud edges (right figure).

### Case 2: $-1.2^\circ$ initial rotation shift:

In this case the model has achieved a different result compared to other test cases.



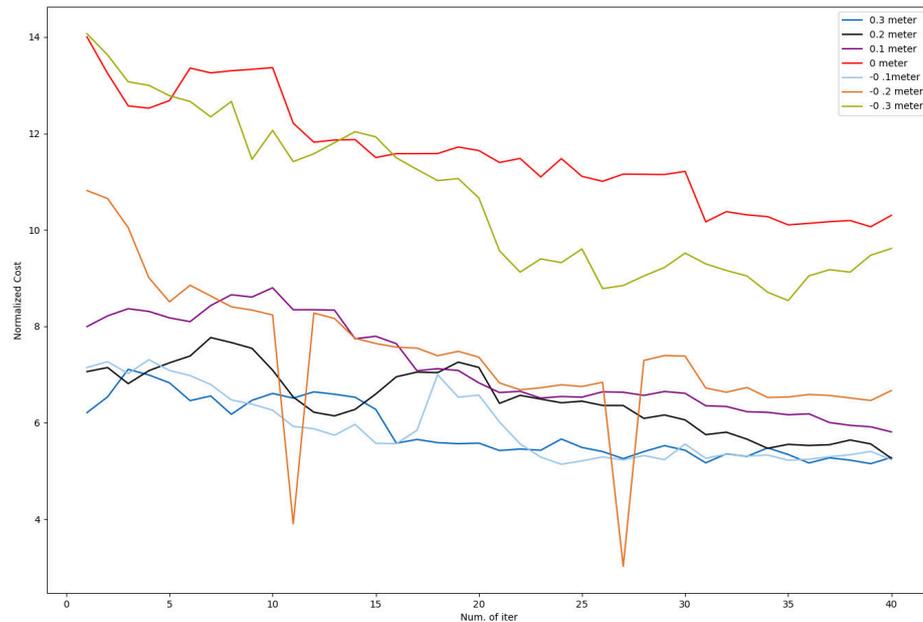
**Figure 7.23:** PYR-XYZ loss of initial rotation shifts with  $-1.2^\circ$ . The X axis represents the number of iterations. The Y axis represents the corresponding difference to the manual calibration result. From the figure we can see that the convergence of the **Pitch** angle and **Z** axis are far from the manual result.



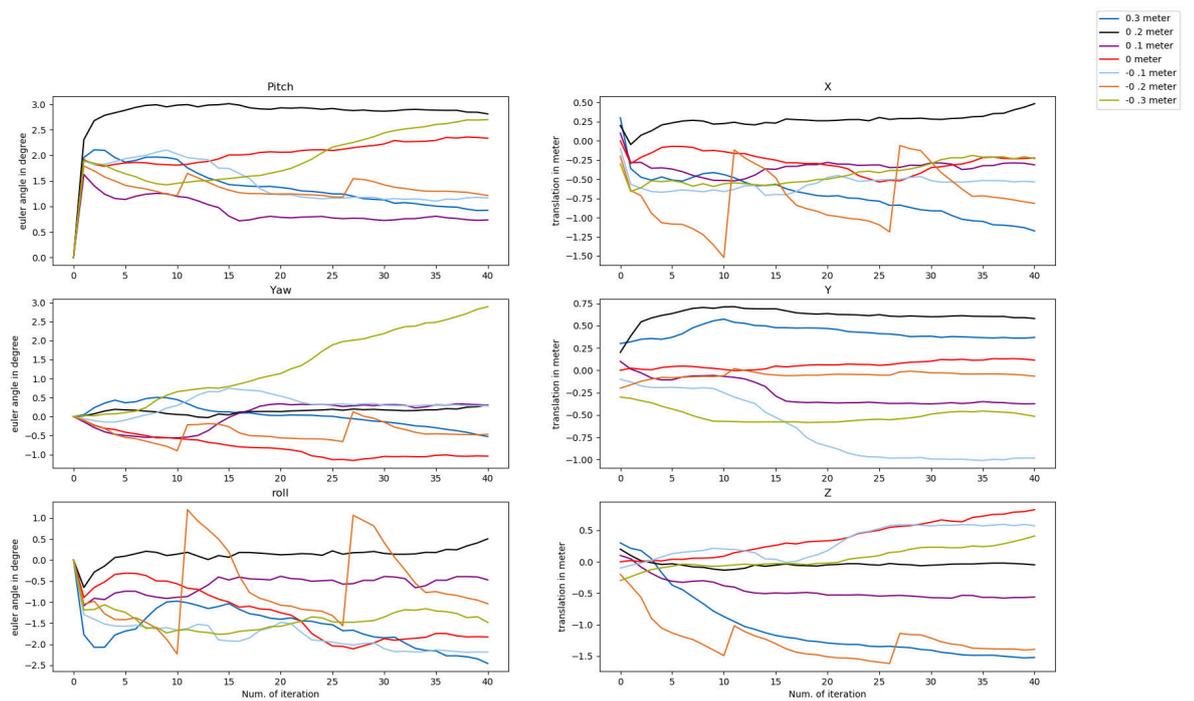
**Figure 7.24:** Qualitative results of initial rotation shifts with  $-1.2^\circ$ . We can see that our model rotates the point clouds on the ground too much. In the image on the right, we can see that the point cloud on the ground has been deformed, and the part that was originally connected to the left bracket of the gantry has been separated. This is also consistent to the results that the **PYR-XYZ** loss is different to other cases.

### 7.2.2 Translation Limit Test

In the following we show our limit tests with translation offsets. We add offsets to the initial translation vector in three directions within the range of  $[-0.3,0.3]$  meters. Fig. 7.25 shows the normalized optimization cost and Fig. 7.26 represents the PYR-XYZ loss.



**Figure 7.25:** Convergence of normalized optimization cost. The X axis represents the number of iterations. The Y axis represents the normalized cost. We tested the optimization performance under level of  $[-0.3,-0.2,-0.1,0,0.1,0.2,0.3]$  meters of translation offsets. It can be seen from the figure that our model can converge within 40 iterations. It is worth noting that in the case of  $-0.2$  m, the value of the cost function has two obvious oscillations during the optimization process.

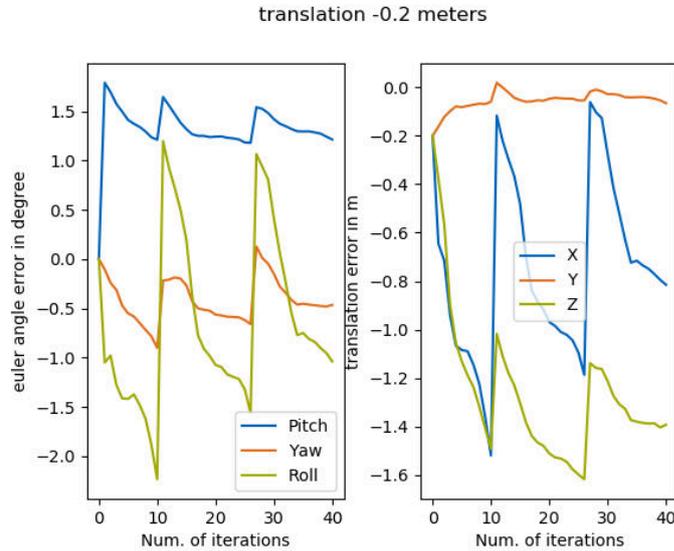


**Figure 7.26:** Convergence of **PYR-XYZ loss** during the optimization. The X axis represents the number of iterations. The Y axis represents the corresponding difference to the manual calibration result. The results are presented in six dimensions separately: Pitch, Roll, Yaw, X, Y, and Z. Again, our model rarely converges to the manual calibration result. The strong oscillations in the case of -0.2 m are also observed here, which is consistent with the situation in the cost function. To be more specific, the oscillation in Roll and X directions is most conspicuous.

### Qualitative Result and Analysis

Here we present the calibration results and their projections for an initial offset of  $-0.2$  m. Moreover, we analyze the cause for the oscillations in the calibration process.

#### Case 1: $-0.2$ meter initial rotation shift:

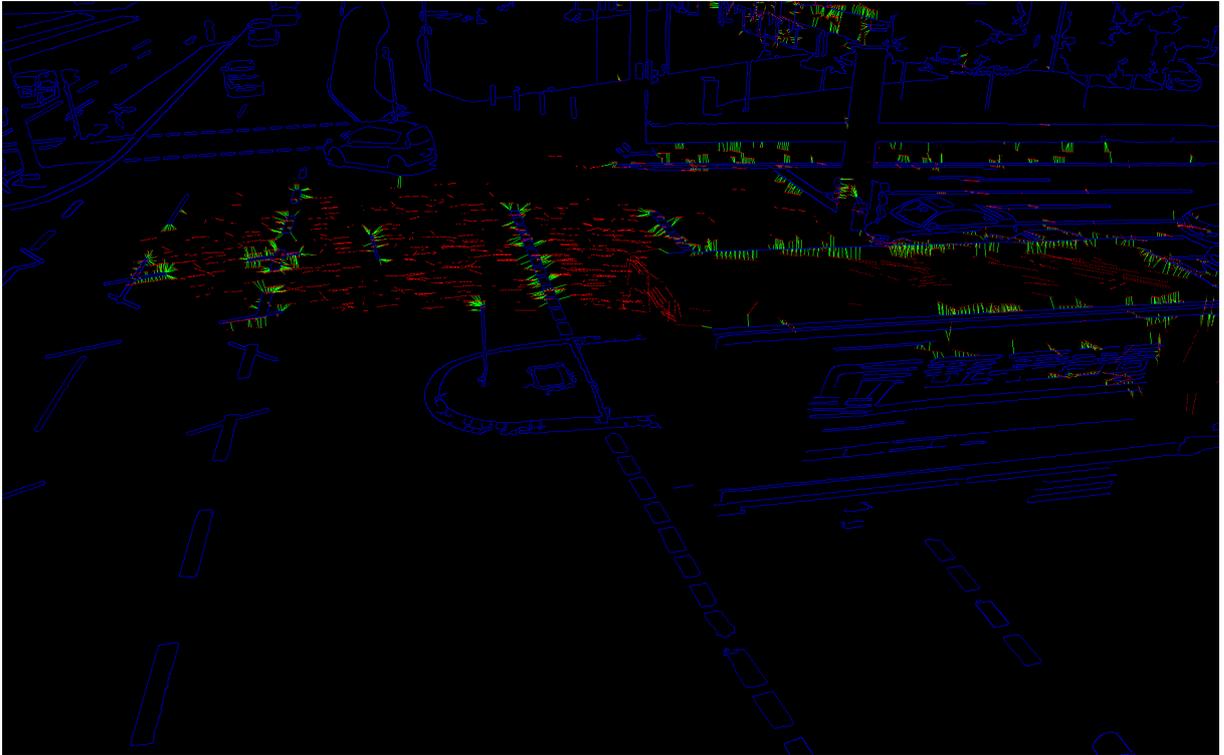


**Figure 7.27: PYR-XYZ loss** of initial rotation shifts with  $-0.2$  meter. The X axis represents the number of iterations. The Y axis represents the corresponding difference to the manual calibration result. Vigorous oscillations can be observed in any direction except the Y axis. There is almost no difference between the initial result and the calibration result. In another word, the extrinsic didn't changed much after calibration.



**Figure 7.28:** Qualitative results of initial rotation shifts with  $-0.2$  meter. Left: **initial extrinsic**. Right: **optimization result**. Comparing the projections before and after calibration, it can be seen that there is almost no difference. This is consistent with the results of PYR-XYZ loss. It can be seen from the figure that the cause of this result is the truck in the lower right. The large number of 3D points on the trailer of the truck should be the cause.

In order to further find out the impact of the truck on the calibration, we checked the edge pairs generated during calibration. Figure 7.29 is the corresponding residual map.



**Figure 7.29:** Residual map of the calibration in the case of **-0.2 meter** initial offset. The trailer has a huge reflective area and its 3D point are projected almost exactly into the corresponding area in the image, which creates a large number of edge pairs. It can be seen from the cost value in 7.27 that there are huge fluctuations in the optimization process, indicating that although the optimization algorithm successfully jumped out of this local minimal, it was pulled back again to the initial value because of the large number of edge pairs established by the truck.

### 7.2.3 Short Summary

In general, in this limit test the model is working well with the initial offsets within the range of  $[-1.2, 0.6]^\circ$  and  $[-0.3, 0.3]$  meters.

The effect of the truck on the calibration emphasized the importance of the distribution of edge pairs for the calibration. High-density point clouds are always prone to generate more edge pairs, and therefore have greater weight for the optimizer. This also indicates that it is necessary to do enough preprocessing on the point cloud. Hence we employed automatic point cloud preprocessing modules like scattering, upsampling, filtering to enable our model to calibrate with live data.

## 7.3 Runtime Measurements

A runtime test is also conducted to our automatic calibration model. The test scenery is the calibration between S110 Ouster south LiDAR and S110 Basler south2 camera. We enable all the modules and test them within 40 iterations.

The calibration model was implemented on a Razer Blade 14 2021 gaming Laptop, equipped



**Figure 7.30:** Scenario for runtime tests. In this scene, there are a lot of shadows on the ground that fall in the area that will affect the calibration results. There are houses in the distance (the background), which will also affect the calibration results. Therefore, this scenario is very suitable for testing the time consumption of all modules.

with a AMD Ryzen 9 5900HX with Radeon Graphics 3.30 GHz central processing unit (CPU), 16 GB of RAM, and an Ubuntu 20.04 operating system.

The results of the runtime test show that our automatic calibration model is capable of calibrating infrastructure sensors in about 148 seconds. In complex scenarios, the total cal-

<b>Module</b>	<b>Runtime [in ms]</b>
Background cropping	71.134
Shadow Filtering	19.660
Canny Edge Detection	84.740
Point Cloud cropping	1.496
Point Cloud Scattering	3,782.460
Point Cloud Outlier Removal	1,361.700
Point Cloud Upsampling	7,399.190
Point Cloud Voxeliastion	16,012.800
Rough Calibration	20,901.200
Optimization	97,983.700
<b>Total</b>	<b>147,618.000</b>

**Table 7.1:** Runtime test result. Here the calibration process from s110 Ouster north LiDAR to s110 Basler south2 camera was tested. We applied scattering 100 times, a search radius of 0.01 m for upsampling, and used a nearest neighbour threshold of 50 for outlier removal.

ibration time with all modules enabled and 40 iterations should not exceed 3 minutes. Our model scattered the point cloud by a factor of 100 in 3,782.46 ms (i.e. 3.7 sec.). What is not mentioned in the table is that the upsampling module generated 3,655,880 new points (initially 736,999 points were in the cloud), which is an upsampling factor of about six. The upsampling module only took 7.39 seconds.

# Chapter 8

## Conclusion

There is no doubt that autonomous driving will become one of the main development directions of the automotive industry in the future, and it will profoundly change our way of travel and even our work and life. *AUTOTech.agil's* goal is to research the flow of information between vehicles and infrastructure along the A9 highway, to create a digital twin of the traffic situation, and to develop value-added services, which will promote the development of autonomous driving.

The calibration between LiDARs and cameras is the most fundamental part of the whole project. Our thesis mainly focused on two calibration tasks:

- Calibration between HD map and infrastructure LiDARs
- Automatic Calibration between infrastructure LiDARs and cameras

One part of our work contains the calibration of LiDARs on the s110 sensor station to the HD map. Since the data in the HD map is saved in the form of keypoints, this calibration process cannot be automated. Calibration targets in the scene must be manually selected to find their keypoint recorded in the HD map file. Furthermore, the corresponding keypoints in the point cloud are obtained manually. We establish the correspondences, using the PnP [28] and the SVD [6] algorithm to calculate the extrinsic parameters. The experimental results prove that the manual calibration method has high accuracy for HD map and LiDAR calibration, but the process is cumbersome.

In the second part of our work we performed automatic calibration of LiDAR and camera sensors. This is the most important part of our thesis. We take the voxel-based automatic calibration method published by Liu et al. [91] from Hong Kong University as a basis and improve it for real-time calibration in outdoor scenes. To enable this model to be used for real-time calibration in outdoor traffic scenarios, we add three main modules to this model.

The first is a module that runs in a loop and acquires point clouds as well as images in real-time. The original model can only be used for the calibration of pre-processed point clouds and images locally, while our model can obtain and parse raw data by subscribing to ROS topics that are published by the sensors in real-time. Additionally, our models run the whole day (24/7) and perform calibration regularly (e.g., every 30 minutes).

The second module is the image pre-processing module. In outdoor calibration, the ranges of the cameras and LiDARs are not the same. In many cases, LiDAR cannot capture the point cloud in the distant view, which is caused by the limitation of hardware. However, in the 3D-2D calibration, the edges of the background in the camera image will still establish correspondences with the LiDAR edges. We therefore employ a template-based background calibration method, which can remove the background extremely fast and efficiently. Another

problem in image processing is the effect of shadows. In reality, the Canny edge detector will also extract the edges of shadows on the ground, and these edges will sometimes establish wrong correspondences with the point cloud. We filter the shadows by adjusting the gray scale value of pixels whose value is lower than a reference pixel on the ground.

The last and most important module is the point cloud preprocessing module. The raw point cloud published by the sensor cannot be directly used for calibration because it is obtained by a single scan of a 64 lanes LiDAR which is very sparse and it is not able to generate enough voxels. We employ a series of methods such as registration, scattering, upsampling to increase the density and textures of the point cloud. On the other hand, raw point clouds contain a lot of noise and outliers, which also needs to be filtered.

Our model is sufficiently robust in the limit test and is able to converge to excellent calibration results in most of the time. What's more, based on the basis of manual calibration, our automatic calibration model can further optimize the extrinsic. However, we also found flaws in the model during testing. The calibration quality of voxel-based methods highly depends on the established edge pairs. The quality of the point cloud, the complexity of the image, and the initial extrinsic will obviously affect the calibration results. Overall, our model is capable of real-time calibration of infrastructure LiDARs and cameras and there is room for further improvement which will be mentioned in the next chapter.

# Chapter 9

## Future Work

As the most fundamental part in the *AUTOTech.agil* project, there are still insufficiencies in our work. Possible improvement in the future is listed below:

- Our automatic calibration model can only be used to calibrate extrinsic parameters. But in real-world applications, intrinsic parameters also need to be calibrated. If the intrinsic parameter is poor, our extrinsic calibration results will also be adversely affected. Therefore, it is necessary to automatically optimize both intrinsic and extrinsic parameters at the same time.
- During our experiments, we found that the quality of the calibration is directly related to the distribution of edge pairs. In order to obtain a sufficient number of evenly distributed edge pairs, we often need to debug the parameters. A possible improvement is to add an automatic detection mechanism to perform parameter tuning and regenerate edges when the edge pairs are not well distributed.
- We employed the adaptive voxelization method [51] in our model to reduce the workload of parameter searching and speed up the establishment of voxels. However, the adaptive voxelization did not perform well in the limit test, so we did not enable this module in the limit test in Chapter 7. In the future, we can further optimize this method so that it can be used for calibration of real-time data.
- In order to make the automatic calibration model applicable to complex outdoor scenes, we added many modules. These modules introduce many hyperparameters, which greatly increase the workload of debugging. These modules can be optimized in the future to reduce the number of hyperparameters.
- Our model performs global optimization. The disadvantage of global optimization is that it will be biased towards the area with more denser point clouds. In these dense areas more edges can be extracted. In sparse areas the optimization thus contributes to a larger error value in the cost function. A possible improvement is to add weights to punish areas with too many edge pairs.



## Bibliography

- [1] Abdel-Aziz, Y., Karara, H., and Hauck, M. “Direct Linear Transformation from Comparator Coordinates into Object Space Coordinates in Close-Range Photogrammetry\*”. In: *Photogrammetric Engineering & Remote Sensing* 81.2 (2015), pp. 103–107. ISSN: 0099-1112. DOI: <https://doi.org/10.14358/PERS.81.2.103>. URL: <https://www.sciencedirect.com/science/article/pii/S0099111215303086>.
- [2] AG, ams-OSRAM. *Flash vs. Scan*. visited on 11/1/2022. URL: <https://ams.com/en/lidar>.
- [3] AG, ams-OSRAM. *VCSEL*. visited on 11/1/2023. URL: <https://ams.com/zh/vcsell>.
- [4] al., X. Y. et. *A survey of downsampling and upsampling methods for 3D Point Cloud Processing*. visited on 12/1/2022. URL: [https://www.youtube.com/watch?v=SCsIA6\\_uM58](https://www.youtube.com/watch?v=SCsIA6_uM58).
- [5] Alexa, M., Behr, J., Cohen-Or, D., Fleishman, S., Levin, D., and Silva, C. T. “Computing and rendering point set surfaces”. In: *IEEE Transactions on visualization and computer graphics* 9.1 (2003), pp. 3–15.
- [6] Arun, K. S., Huang, T. S., and Blostein, S. D. “Least-Squares Fitting of Two 3-D Point Sets”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-9.5* (1987), pp. 698–700. DOI: 10.1109/TPAMI.1987.4767965.
- [7] Bentley, J. L. “Multidimensional Binary Search Trees Used for Associative Searching”. In: *Commun. ACM* 18.9 (Sept. 1975), pp. 509–517. ISSN: 0001-0782. DOI: 10.1145/361002.361007. URL: <https://doi.org/10.1145/361002.361007>.
- [8] Besl, P. J. and McKay, N. D. “A Method for Registration of 3-D Shapes”. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 14 (1992), pp. 239–256.
- [9] Brian Robinson, I. *Calibration LiDAR camera Manual*. visited on 13/1/2022. URL: [https://github.com/Livox-SDK/livox\\_camera\\_lidar\\_calibration](https://github.com/Livox-SDK/livox_camera_lidar_calibration).
- [10] Bruckner, M. *OpenDRIVE parser*. visited on 13/1/2022. URL: <https://github.com/Brucknem/OpenDRIVE/>.
- [11] Bueno, M., González-Jorge, H., Martínez-Sánchez, J., and Lorenzo, H. “Automatic point cloud coarse registration using geometric keypoint descriptors for indoor scenes”. In: *Automation in Construction* 81 (2017), pp. 134–148. ISSN: 0926-5805. DOI: <https://doi.org/10.1016/j.autcon.2017.06.016>. URL: <https://www.sciencedirect.com/science/article/pii/S0926580517305265>.
- [12] Caesar, H., Bankiti, V., Lang, A. H., Vora, S., Liong, V. E., Xu, Q., Krishnan, A., Pan, Y., Baldan, G., and Beijbom, O. “nuScenes: A multimodal dataset for autonomous driving”. In: *CVPR*. 2020.
- [13] Canny, J. “A Computational Approach to Edge Detection”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-8.6* (1986), pp. 679–698. DOI: 10.1109/TPAMI.1986.4767851.

- [14] Chai, T. and Draxler, R. “Root mean square error (RMSE) or mean absolute error (MAE)?” In: *Geosci. Model Dev.* 7 (Jan. 2014). DOI: 10.5194/gmdd-7-1525-2014.
- [15] Chen, H. “Pose determination from line-to-plane correspondences: existence condition and closed-form solutions”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 13.6 (1991), pp. 530–541. DOI: 10.1109/34.87340.
- [16] Creß, C., Zimmer, W., Strand, L., Fortkord, M., Dai, S., Lakshminarasimhan, V., and Knoll, A. “A9-Dataset: Multi-Sensor Infrastructure-Based Dataset for Mobility Research”. In: *2022 IEEE Intelligent Vehicles Symposium (IV)*. 2022, pp. 965–970. DOI: 10.1109/IV51971.2022.9827401.
- [17] Dahlström, T. *How Accurate Are HD Maps for Autonomous Driving and ADAS Simulation?* visited on 13/1/2022. URL: <https://atlatec.de/en/blog/how-accurate-are-hd-maps-for-autonomous-driving-and-adas-simulation/>.
- [18] Dai, A., Chang, A. X., Savva, M., Halber, M., Funkhouser, T., and Nießner, M. *ScanNet: Richly-annotated 3D Reconstructions of Indoor Scenes*. 2017. DOI: 10.48550/ARXIV.1702.04405. URL: <https://arxiv.org/abs/1702.04405>.
- [19] Daniel. *Point cloud upsampling*. visited on 13/1/2022. URL: <https://github.com/danielTobon43/upsamplingCloudP>.
- [20] Davis, N. *An Introduction to Laser Diodes*. visited on 11/1/2022. URL: <https://www.allaboutcircuits.com/technical-articles/an-introduction-to-laser-diodes/>.
- [21] Dhall, A., Chelani, K., Radhakrishnan, V., and Krishna, K. M. *LiDAR-Camera Calibration using 3D-3D Point correspondences*. 2017. DOI: 10.48550/ARXIV.1705.09785. URL: <https://arxiv.org/abs/1705.09785>.
- [22] Dhall, A., Chelani, K., Radhakrishnan, V., and Krishna, K. M. “LiDAR-Camera Calibration using 3D-3D Point correspondences”. In: *ArXiv abs/1705.09785* (2017).
- [23] Dhome, M., Richetin, M., Lapreste, J.-T., and Rives, G. “Determination of the attitude of 3D objects from a single perspective view”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 11.12 (1989), pp. 1265–1278. DOI: 10.1109/34.41365.
- [24] Dinesh, C., Cheung, G., and Bajić, I. V. “3D Point Cloud Super-Resolution via Graph Total Variation on Surface Normals”. In: *2019 IEEE International Conference on Image Processing (ICIP)*. 2019, pp. 4390–4394. DOI: 10.1109/ICIP.2019.8803560.
- [25] e.V., A. *ASAM OpenDRIVE*. visited on 13/1/2022. URL: <https://www.asam.net/standards/detail/opendrive/>.
- [26] EMG(emapgo.com.cn). *feature map*. visited on 13/1/2022. URL: <https://new.qq.com/rain/a/20200727A0DKJA00>.
- [27] Fang, C., Ding, S., Dong, Z., Li, H., Zhu, S., and Tan, P. “Single-Shot is Enough: Panoramic Infrastructure Based Calibration of Multiple Cameras and 3D LiDARs”. In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2021, pp. 8890–8897. DOI: 10.1109/IROS51168.2021.9636767.
- [28] Fischler, M. A. and Bolles, R. C. “Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography”. In: *Commun. ACM* 24.6 (June 1981), pp. 381–395. ISSN: 0001-0782. DOI: 10.1145/358669.358692. URL: <https://doi.org/10.1145/358669.358692>.

- [29] Fischler, M. A. and Bolles, R. C. "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography". In: *Readings in Computer Vision*. Ed. by Fischler, M. A. and Firschein, O. San Francisco (CA): Morgan Kaufmann, 1987, pp. 726–740. ISBN: 978-0-08-051581-6. DOI: <https://doi.org/10.1016/B978-0-08-051581-6.50070-2>. URL: <https://www.sciencedirect.com/science/article/pii/B9780080515816500702>.
- [30] Fix, E. and Hodges, J. *Discriminatory Analysis: Nonparametric Discrimination: Consistency Properties*. USAF School of Aviation Medicine, 1951. URL: <https://books.google.de/books?id=4XwytAEACAAJ>.
- [31] Geiger, A., Lenz, P., Stiller, C., and Urtasun, R. "Vision meets Robotics: The KITTI Dataset". In: *International Journal of Robotics Research (IJRR)* (2013).
- [32] Ghimire, S. *Voxelization of point cloud*. visited on 11/1/2022. URL: <https://medium.com/@ghimire.aiesecer/voxel-fitting-on-lidar-point-cloud-713b2c7b1f3d>.
- [33] GmbH, S. S. *LiDAR*. visited on 11/1/2023. URL: <https://www.ssla.co.uk/lidar1>.
- [34] Google. *Google map*. visited on 13/1/2022. URL: <https://www.google.com/maps>.
- [35] He, X., Zemel, R., and Carreira-Perpinan, M. "Multiscale conditional random fields for image labeling". In: *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004*. Vol. 2. 2004, pp. II–II. DOI: 10.1109/CVPR.2004.1315232.
- [36] Headquarters, C. *What is LiDAR?* visited on 11/1/2023. URL: <https://www.synopsys.com/glossary/what-is-lidar.html>.
- [37] Huang, H., Li, D., Zhang, H., Ascher, U., and Cohen-Or, D. "Consolidation of Unorganized Point Clouds for Surface Reconstruction". In: *ACM Trans. Graph.* 28.5 (Dec. 2009), pp. 1–7. ISSN: 0730-0301. DOI: 10.1145/1618452.1618522. URL: <https://doi.org/10.1145/1618452.1618522>.
- [38] Huang, X., Mei, G., Zhang, J., and Abbas, R. *A comprehensive survey on point cloud registration*. Mar. 2021.
- [39] Inc., M. A. *nuScenes detection task*. visited on 12/1/2022. URL: <https://www.nuscenes.org/object-detection?externalData=all&mapData=all&modalities=Any>.
- [40] Institute, O. R. *OXFORD ROBOTCAR DATASET*. visited on 12/1/2022. URL: <https://robotcar-dataset.robots.ox.ac.uk/>.
- [41] JavaTpoint. *KNN*. visited on 11/1/2022. URL: <https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning>.
- [42] Kalman, R. E. "A New Approach to Linear Filtering and Prediction Problems". In: *Transactions of the ASME—Journal of Basic Engineering* 82.Series D (1960), pp. 35–45.
- [43] Laurmaa, V., Picasso, M., and Steiner, G. "An octree-based adaptive semi-Lagrangian VOF approach for simulating the displacement of free surfaces". In: *Computers & Fluids* 131 (Mar. 2016). DOI: 10.1016/j.compfluid.2016.03.005.
- [44] Lepetit, V., Moreno-Noguer, F., and Fua, P. "EPnP: An accurate O(n) solution to the PnP problem". In: *International Journal of Computer Vision* 81 (Feb. 2009). DOI: 10.1007/s11263-008-0152-6.
- [45] Li, X., Xiao, Y., Wang, B., Ren, H., Zhang, Y., and Ji, J. "Automatic targetless LiDAR–camera calibration: a survey". In: *Artificial Intelligence Review* (Nov. 2022), pp. 1–39. DOI: 10.1007/s10462-022-10317-y.

- [46] Li, Z., Du, Y., Zhu, M., Zhou, S., and Zhang, L. “A Survey of 3D Object Detection Algorithms for Intelligent Vehicles Development”. In: *Artif. Life Robot.* 27.1 (Feb. 2022), pp. 115–122. ISSN: 1433-5298. DOI: 10.1007/s10015-021-00711-0. URL: <https://doi.org/10.1007/s10015-021-00711-0>.
- [47] Library, P. C. *PCD*. visited on 11/1/2022. URL: [https://pointclouds.org/documentation/tutorials/pcd\\_file\\_format.html](https://pointclouds.org/documentation/tutorials/pcd_file_format.html).
- [48] Lipman, Y., Cohen-Or, D., Levin, D., and Tal-Ezer, H. “Parameterization-Free Projection for Geometry Reconstruction”. In: *ACM Trans. Graph.* 26.3 (July 2007), 22–es. ISSN: 0730-0301. DOI: 10.1145/1276377.1276405. URL: <https://doi.org/10.1145/1276377.1276405>.
- [49] Liu, R., Wang, J., and Zhang, B. “High Definition Map for Automated Driving: Overview and Analysis”. In: *Journal of Navigation* 73.2 (2020), pp. 324–341. DOI: 10.1017/S0373463319000638.
- [50] Liu, X., Yuan, C., and Zhang, F. “Targetless Extrinsic Calibration of Multiple Small FoV LiDARs and Cameras Using Adaptive Voxelization”. In: *IEEE Transactions on Instrumentation and Measurement* 71 (2022), pp. 1–12. DOI: 10.1109/TIM.2022.3176889.
- [51] Liu, X., Yuan, C., and Zhang, F. “Targetless Extrinsic Calibration of Multiple Small FoV LiDARs and Cameras Using Adaptive Voxelization”. In: *IEEE Transactions on Instrumentation and Measurement* 71 (2022), pp. 1–12. DOI: 10.1109/TIM.2022.3176889.
- [52] Liu, Y., Huang, T., and Faugeras, O. “Determination of camera location from 2-D to 3-D line and point correspondences”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 12.1 (1990), pp. 28–37. DOI: 10.1109/34.41381.
- [53] LLC, 2.-2. W. *waymo*. visited on 12/1/2022. URL: <https://waymo.com/open/about/>.
- [54] Ma, T., Liu, Z., Yan, G., and Li, Y. *CRLF: Automatic Calibration and Refinement based on Line Feature for LiDAR and Camera in Road Scenes*. 2021. DOI: 10.48550/ARXIV.2103.04558. URL: <https://arxiv.org/abs/2103.04558>.
- [55] Maddern, W., Pascoe, G., Linegar, C., and Newman, P. “1 Year, 1000 Km: The Oxford RobotCar Dataset”. In: *Int. J. Rob. Res.* 36.1 (Jan. 2017), pp. 3–15. ISSN: 0278-3649. DOI: 10.1177/0278364916679498. URL: <https://doi.org/10.1177/0278364916679498>.
- [56] Matthews, K. *What are HD maps, and how will they get us closer to autonomous cars?* visited on 13/1/2022. URL: <https://iot.eetimes.com/what-are-hd-maps-and-how-will-they-get-us-closer-to-autonomous-cars/>.
- [57] Meagher, D. “Octree Encoding: A New Technique for the Representation, Manipulation and Display of Arbitrary 3-D Objects by Computer”. In: (Oct. 1980).
- [58] Mellado, N., Aiger, D., and Mitra, N. J. “Super 4PCS Fast Global Pointcloud Registration via Smart Indexing”. In: *Computer Graphics Forum* 33 (2014).
- [59] Menze, M. and Geiger, A. “Object Scene Flow for Autonomous Vehicles”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015.
- [60] Muñoz-Bañón, M. Á., Candelas, F. A., and Torres, F. “Targetless Camera-LiDAR Calibration in Unstructured Environments”. In: *IEEE Access* 8 (2020), pp. 143692–143705. DOI: 10.1109/ACCESS.2020.3014121.
- [61] Naik, S., Mudenagudi, U., Tabib, R., and Jamadandi, A. “FeatureNet: Upsampling of Point Cloud and It’s Associated Features”. In: *SIGGRAPH Asia 2020 Posters*. 2020, pp. 1–2.
- [62] Odunlade, E. *What is LiDAR and How does it Work*. visited on 11/1/2023. URL: <https://circuitdigest.com/article/what-is-lidar-and-how-does-lidar-works/>.

- [63] OpenDRIVE, A. *OpenDRIVE 1.6*. visited on 13/1/2022. URL: [https://releases.asam.net/OpenDRIVE/1.6.0/ASAM\\_OpenDRIVE\\_BS\\_V1-6-0.html](https://releases.asam.net/OpenDRIVE/1.6.0/ASAM_OpenDRIVE_BS_V1-6-0.html).
- [64] Pokojski, W. and Pokojska, P. “Voronoi diagrams – inventor, method, applications”. In: *Polish Cartographical Review* 50 (Sept. 2018), pp. 141–150. DOI: 10.2478/pcr-2018-0009.
- [65] Preiner, R., Mattausch, O., Arikan, M., Pajarola, R., and Wimmer, M. “Continuous Projection for Fast L1 Reconstruction”. In: *ACM Trans. Graph.* 33.4 (July 2014). ISSN: 0730-0301. DOI: 10.1145/2601097.2601172. URL: <https://doi.org/10.1145/2601097.2601172>.
- [66] Providentia, R. *Add value from providentia++*. visited on 12/1/2022. URL: <https://innovation-mobility.com/mobilitaet-services-verkehr-neue-dienste/>.
- [67] Quan, L. and Lan, Z. “Linear N-point camera pose determination”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 21.8 (1999), pp. 774–780. DOI: 10.1109/34.784291.
- [68] Ranftl, R., Lasinger, K., Hafner, D., Schindler, K., and Koltun, V. *Towards Robust Monocular Depth Estimation: Mixing Datasets for Zero-shot Cross-dataset Transfer*. 2019. DOI: 10.48550/ARXIV.1907.01341. URL: <https://arxiv.org/abs/1907.01341>.
- [69] Robotics, O. *Robot Operating System*. visited on 14/1/2022. URL: <https://www.ros.org/>.
- [70] Sayyah, K., Efimov, O., Patterson, P., Schaffner, J., White, C., Seurin, J.-F., Xu, G., and Miglo, A. “Two-dimensional pseudo-random optical phased array based on tandem optical injection locking of vertical cavity surface emitting lasers”. In: *Opt. Express* 23.15 (July 2015), pp. 19405–19416. DOI: 10.1364/OE.23.019405. URL: <https://opg.optica.org/oe/abstract.cfm?URI=oe-23-15-19405>.
- [71] Scale AI, I. and Technology, H. *PandaSet by Hesai and Scale AI*. visited on 12/1/2022. URL: <https://pandaset.org/>.
- [72] Sitnik, R. and Karaszewski, M. “Optimized point cloud triangulation for 3D scanning systems”. In: *Machine Graphics and Vision* 17.4 (2008), pp. 349–371.
- [73] Song, S., Lichtenberg, S. P., and Xiao, J. “SUN RGB-D: A RGB-D scene understanding benchmark suite”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 567–576. DOI: 10.1109/CVPR.2015.7298655.
- [74] Stanford University, P. U. and Munich, T. U. of. *ScanNet Benchmark*. visited on 12/1/2022. URL: [https://kaldir.vc.in.tum.de/scannet\\_benchmark/](https://kaldir.vc.in.tum.de/scannet_benchmark/).
- [75] Stanford University, P. U. and Munich, T. U. of. *ScanNet: Richly-annotated 3D Reconstructions of Indoor Scenes*. visited on 12/1/2022. URL: <http://www.scan-net.org/>.
- [76] Technology (KIT), K. I. of and Chicago (TTI-C), T. T. I. at. *The KITTI Vision Benchmark for object detection*. visited on 12/1/2022. URL: [https://www.cvlibs.net/datasets/kitti/eval\\_object.php?obj\\_benchmark=3d](https://www.cvlibs.net/datasets/kitti/eval_object.php?obj_benchmark=3d).
- [77] Technology (KIT), K. I. of and Chicago (TTI-C), T. T. I. at. *The KITTI Vision Benchmark Suite*. visited on 12/1/2022. URL: <https://www.cvlibs.net/datasets/kitti/>.
- [78] Teniente A., E., Aviles, T., and Andrade Cetto, J. “Registration of 3D Points Clouds for Urban Robot Mapping”. In: (Jan. 2008), p. 8.
- [79] Terzakis, G. and Lourakis, M. I. A. “A Consistently Fast and Globally Optimal Solution to the Perspective-n-Point Problem”. In: *European Conference on Computer Vision*. 2020.

- [80] Triggs, B., McLauchlan, P. F., Hartley, R. I., and Fitzgibbon, A. W. “Bundle Adjustment — A Modern Synthesis”. In: *Vision Algorithms: Theory and Practice*. Ed. by Triggs, B., Zisserman, A., and Szeliski, R. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 298–372. ISBN: 978-3-540-44480-0.
- [81] TUM. *Research for automated driving with external infrastructure*. visited on 12/1/2022. URL: <https://innovation-mobility.com/en/#aktuelles>.
- [82] University, P. *SUN RGB-D: A RGB-D Scene Understanding Benchmark Suite*. visited on 12/1/2022. URL: <https://rgbd.cs.princeton.edu/>.
- [83] Vardhan, H. *What are HD Maps?* visited on 13/1/2022. URL: <https://www.geospatialworld.net/article/hd-maps-autonomous-vehicles/>.
- [84] Wang, W., Nobuhara, S., Nakamura, R., and Sakurada, K. *SOIC: Semantic Online Initialization and Calibration for LiDAR and Camera*. 2020. DOI: 10.48550/ARXIV.2003.04260. URL: <https://arxiv.org/abs/2003.04260>.
- [85] Wang, W., Sakurada, K., and Kawaguchi, N. “Reflectance Intensity Assisted Automatic and Accurate Extrinsic Calibration of 3D LiDAR and Panoramic Camera Using a Printed Chessboard”. In: *Remote Sensing 9.8* (Aug. 2017), p. 851. DOI: 10.3390/rs9080851. URL: <https://doi.org/10.3390%2Frs9080851>.
- [86] *Waymo Open Dataset: An autonomous driving dataset*. 2019.
- [87] Xu, C., Zhang, L., Cheng, L., and Koch, R. “Pose Estimation from Line Correspondences: A Complete Analysis and a Series of Solutions”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.6 (2017), pp. 1209–1222. DOI: 10.1109/TPAMI.2016.2582162.
- [88] Yifan, W., Wu, S., Huang, H., Cohen-Or, D., and Sorkine-Hornung, O. “Patch-based progressive 3d point set upsampling”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 5958–5967.
- [89] Yu, C., Gao, C., Wang, J., Yu, G., Shen, C., and Sang, N. *BiSeNet V2: Bilateral Network with Guided Aggregation for Real-time Semantic Segmentation*. 2020. DOI: 10.48550/ARXIV.2004.02147. URL: <https://arxiv.org/abs/2004.02147>.
- [90] Yu, L., Li, X., Fu, C.-W., Cohen-Or, D., and Heng, P.-A. “PU-Net: Point Cloud Upsampling Network”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2018.
- [91] Yuan, C., Liu, X., Hong, X., and Zhang, F. *Pixel-level Extrinsic Self Calibration of High Resolution LiDAR and Camera in Targetless Environments*. 2021. DOI: 10.48550/ARXIV.2103.01627. URL: <https://arxiv.org/abs/2103.01627>.
- [92] Zeng, G., Li, H., Wang, X., and Li, N. “Point cloud up-sampling network with multi-level spatial local feature aggregation”. In: *Computers & Electrical Engineering* 94 (2021), p. 107337.
- [93] Zhang, Q. and Pless, R. “Extrinsic calibration of a camera and laser range finder (improves camera calibration)”. In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*. Vol. 3. 2004, 2301–2306 vol.3. DOI: 10.1109/IROS.2004.1389752.
- [94] Zhang, X., Zhu, S., Guo, S., Li, J., and Liu, H. “Line-based Automatic Extrinsic Calibration of LiDAR and Camera”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. 2021, pp. 9347–9353. DOI: 10.1109/ICRA48506.2021.9561216.

- [95] Zhang, Y., Zhao, W., Sun, B., Zhang, Y., and Wen, W. “Point Cloud Upsampling Algorithm: A Systematic Review”. In: *Algorithms* 15.4 (2022). ISSN: 1999-4893. DOI: 10.3390/a15040124. URL: <https://www.mdpi.com/1999-4893/15/4/124>.
- [96] Zhou, L., Yang, Y., Abello, M., and Kaess, M. “A Robust and Efficient Algorithm for the PnL Problem Using Algebraic Distance to Approximate the Reprojection Distance”. In: *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*. AAAI’19/IAAI’19/EAAI’19. Honolulu, Hawaii, USA: AAAI Press, 2019. ISBN: 978-1-57735-809-1. DOI: 10.1609/aaai.v33i01.33019307. URL: <https://doi.org/10.1609/aaai.v33i01.33019307>.
- [97] Zhu, Y., Li, C., and Zhang, Y. “Online Camera-LiDAR Calibration with Sensor Semantic Information”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 2020, pp. 4970–4976. DOI: 10.1109/ICRA40945.2020.9196627.
- [98] Zimmer, W. *A9 calibration data*. visited on 13/1/2022. URL: <https://docs.google.com/spreadsheets/d/1-tcaWvmRHG-uq1bSNnlr-oBMRsOntseNETTbmK2AWM/edit#gid=0>.
- [99] Zimmer, W. *calibration target*. visited on 12/1/2022. URL: [https://gitlab.lrz.de/providentiaplusplus/toolchain/-/tree/dev/bohan/101/camera-lidar-calibration/package/calibration\\_camera\\_lidar\\_manual](https://gitlab.lrz.de/providentiaplusplus/toolchain/-/tree/dev/bohan/101/camera-lidar-calibration/package/calibration_camera_lidar_manual).
- [100] Zimmer, W. *LiDAR data provided by Walter Zimmer*. unpublished document.
- [101] Zimmer, W. *Point cloud registration*. visited on 13/1/2022. URL: [https://gitlab.lrz.de/providentiaplusplus/toolchain/-/tree/dev/57/3d-object-detection-LiDAR-unsupervised/package/point\\_cloud\\_registration](https://gitlab.lrz.de/providentiaplusplus/toolchain/-/tree/dev/57/3d-object-detection-LiDAR-unsupervised/package/point_cloud_registration).