Circuits and Systems
Mekelweg 4,
2628 CD Delft
The Netherlands
https://sps.ewi.tudelft.nl/

SPS-2024-5494079

M.Sc. Thesis

Multi-feature-based Automatic Targetless Camera-LiDAR Extrinsic Calibration

Xi Chen B.Sc.

Abstract

In autonomous driving, environmental perception, crucial for navigation and decision-making, depends on integrating data from multiple sensors like cameras and LiDAR. Camera-LiDAR fusion combines detailed imagery with precise depth, improving environmental awareness. Effective data fusion requires accurate extrinsic calibration to align camera and LiDAR data under one coordinate system. We aim to calibrate the camera and LiDAR extrinsic automatically and without specific targets. Targetless, non-automated calibration methods are time-consuming and labor-intensive. Existing advanced methods have proven that automatic calibration methods based on edge features are effective, and most focus on the extraction and matching of single features. The proposed method matches 2D edges from LiDAR's multi-attribute density map with image-derived intensity gradient and semantic edges, facilitating 2D-2D edge registration. We innovate by incorporating semantic feature and addressing random initial setting through the PnP problem of centroid pairs, enhancing the convergence of the objective function. We introduce a weighted multi-frame averaging technique, considering frame correlation and semantic importance, for smoother calibration. Tested on the KITTI dataset, it surpasses four current methods in single-frame tests and shows more robustness in multi-frame tests than MulFEAT. Our algorithm leverages semantic information for extrinsic calibration, striking a balance between network complexity and robustness. Future enhancements may include using machine learning to convert sparse matrices to dense formats for improved optimization efficiency.

Multi-feature-based Automatic Targetless Camera-LiDAR Extrinsic Calibration

Thesis

submitted in partial fulfillment of the
requirements for the degree of

Master of Science

in

Electrical Engineering

by

Xi Chen B.Sc.
born in Hebei, China

This work was performed in:

Circuits and Systems Group
Department of Microelectronics
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Delft University of Technology

Delft University of Technology
Department of
Microelectronics

The undersigned hereby certify that they have read and recommend to the Faculty of Electrical Engineering, Mathematics and Computer Science for acceptance a thesis entitled **"Multi-feature-based Automatic Targetless Camera-LiDAR Extrinsic Calibration"** by **Xi Chen B.Sc.** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: January 18, 2024

Chairman: _____
dr. R.T. Rajan

Advisor: _____
dr. R.T. Rajan

Committee Members: _____
prof.dr.ir. A.J. van der Veen

_____
dr. H. Caesar

_____
ir. K. Zhang

Abstract

In autonomous driving, environmental perception, crucial for navigation and decision-making, depends on integrating data from multiple sensors like cameras and LiDAR. Camera-LiDAR fusion combines detailed imagery with precise depth, improving environmental awareness. Effective data fusion requires accurate extrinsic calibration to align camera and LiDAR data under one coordinate system. We aim to calibrate the camera and LiDAR extrinsic automatically and without specific targets. Targetless, non-automated calibration methods are time-consuming and labor-intensive. Existing advanced methods have proven that automatic calibration methods based on edge features are effective, and most focus on the extraction and matching of single features. The proposed method matches 2D edges from LiDAR's multi-attribute density map with image-derived intensity gradient and semantic edges, facilitating 2D-2D edge registration. We innovate by incorporating semantic feature and addressing random initial setting through the PnP problem of centroid pairs, enhancing the convergence of the objective function. We introduce a weighted multi-frame averaging technique, considering frame correlation and semantic importance, for smoother calibration. Tested on the KITTI dataset, it surpasses four current methods in single-frame tests and shows more robustness in multi-frame tests than MulFEAT. Our algorithm leverages semantic information for extrinsic calibration, striking a balance between network complexity and robustness. Future enhancements may include using machine learning to convert sparse matrices to dense formats for improved optimization efficiency.

Acknowledgments

This thesis represents the conclusion of a long and challenging journey, one that I could only undertake thanks to the unwavering support I received from many. Hereby I extend my deepest appreciation to everyone who has assisted and stood by me throughout my master's studies and research.

First and foremost, I am immensely grateful to my supervisor, dr. R.T. Rajan, and my day-to-day guide, Kaiwen Zhang, for their invaluable guidance and mentorship during these intense months of work. This project was fraught with challenges, and there were moments of self-doubt, frustration, and setbacks. I am profoundly thankful to my supervisor for their continuous encouragement and relentless support during the dissertation phase, always available in my times of need. My health issues occasionally hindered my progress, yet my supervisor showed immense understanding and consideration. Fear of negative feedback often loomed over me, but thankfully, my supervisor's tolerance and constructive approach always guided me. In every interaction, regardless of my progress, they provided motivating feedback, sparking inspiration and driving the advancement of my project. My gratitude also goes to Kaiwen for his extensive assistance with the technical aspects of this project. Whenever I faced logical or computational dilemmas, Kaiwen encouraged me to arrive at solutions independently. Even amidst her busy schedule, he consistently made time for our meetings, for which I am eternally thankful. I also would like to express my sincere thanks to my committee members, prof.dr.ir. A.J. van der Veen, and dr. H. Caesar, and to the professors and friends who attended my graduation defense. Your insights, advice, and discussions regarding this thesis are invaluable. This defense marks a significant milestone in my life, one that I will cherish forever.

With profound gratitude, I acknowledge the unwavering financial support and boundless encouragement from my parents. In the past two years or so, I have seen mountains, rivers, and stars, walked through many landscapes, and finally realized my childhood dream. I am also immensely thankful for my dearest friend, Fei. Despite the physical distance of thousands of miles and a challenging seven-hour time difference, our bond remains strong. We share a deep understanding and provide each other with invaluable emotional support.

In conclusion, my heart is filled with profound gratitude for all the support and kindness I have received. Each individual I've mentioned, and many unmentioned, has played an integral role in my journey, shaping my experiences and contributing to my growth.

Xi Chen B.Sc.
Delft, The Netherlands
January 18, 2024

Contents

List of Figures

List of Tables

List of Notations

a	Scalar
$\mathbf{a}$	Column vector
$\mathbf{A}$	Matrix
$\mathbf{A}_{N\times M}$	Matrix with N rows and M columns
$\mathbf{A}_{ij}$	The entry in the i-th row and j-th column of matrix $\mathbf{A}$
$\mathbf{B}_{N\times M\times K}$	K dimensional matrix with N rows and M columns
$\mathbf{B}_{ijk}$	The entry in row i and column j in the k-th dimension of matrix $\mathbf{B}$
$\hat{\mathbf{a}}$	Estimate of vector $\mathbf{a}$
$\hat{\mathbf{A}}$	Estimate of matrix $\mathbf{A}$
$\mathcal{A}$	Set
$f(\cdot)$	Objective function
$p(\theta\|\mathbf{X})$	Posterior
$p(\mathbf{X}\|\theta)$	Likelihood function
$\mathbf{U}$	Identity matrix (Unit matrix)
$\mathbf{p}^c$	Homogeneous coordinate vector of a point under c coordinate system
$\mathbf{p}_i^c$	Homogeneous coordinate vector of i-th point under c coordinate system
$\mathbf{p}^L$	Homogeneous coordinates vector of a point in LiDAR coordinates
$\mathbf{p}^C$	Homogeneous coordinates vector of a point in camera coordinates
$\mathbf{p}^I$	Homogeneous coordinates vector of a point on the camera image plane
$\mathbf{P}^c$	Point coordinate matrix under c coordinate system
$\mathbf{P}_k^c$	Point coordinate matrix of k-th frame under c coordinate
$\mathbf{P}^L$	Matrix of homogeneous coordinates vectors in the LiDAR coordinate
$\mathbf{P}^C$	Matrix of homogeneous coordinates vectors in the camera coordinate
$\mathbf{P}^I$	Matrix of homogeneous coordinates vectors on the camera image plane
$\mathbf{P}^L$	Matrix of homogeneous coordinates vectors in the LiDAR coordinate
$\mathbf{P}^C$	Matrix of homogeneous coordinates vectors in the camera coordinate
$\mathbf{P}^I$	Matrix of homogeneous coordinates vectors on the camera image plane
$\mathbf{K}$	Camera intrinsic matrix
${}^a_b(\cdot)$	Operation from b coordinate system to a coordinate system
${}^C_L\mathbf{T}$	Extrinsic matrix from LiDAR coordinate to camera coordinate
$\mathbf{R}$	Rotation matrix
$\mathbf{t}$	Translation vector
$\boldsymbol{\theta}$	...
$\mathbf{I}^{gray}$	Grayscale image matrix
$\mathbf{I}^{rgb}$	RGB image matrix
$\mathbf{I}^{gray}_{H\times W}$	Grayscale image matrix with height H and width W
$\mathbf{I}^{rgb}_{H\times W}$	RGB image matrix with height H and width W
$\mathbf{E}^I$	Edge feature matrix of image
$\mathbf{E}^I(u,v)$	Edge intensity of the pixel coordinates (u,v) in the image
$\mathbf{E}_k^I$	Edge feature matrix of the k-th frame image
$\mathbf{E}_k^I(u,v)$	Edge intensity of the pixel coordinates (u,v) in the k-th image

$\mathcal{C}$	Image set
$\mathcal{L}$	LiDAR point cloud set
$\mathcal{M}$	...
$\mathcal{F}^C$	Feature set ...
$\mathcal{F}^L$	Feature set ...
r	Reflectivity
d	Depth
ξ	Distance discontinuity
$\boldsymbol{\psi}$	Attributes of a LiDAR point
$\mathbf{\Psi}$	Sparse projection matrix
$\mathbf{\Phi}$	Densely completed projection matrix
$\mathbf{E}^L$	Edge feature matrix of one frame LiDAR point cloud
$\mathbf{E}_k^L$	Edge feature matrix of the k-th frame LiDAR point cloud
$\mathbf{E}_k^L(u,v)$	Edge intensity of the pixel coordinates (u,v) in the k-th projected image

1 Introduction

Over the past decade, autonomous driving technology has evolved and gained tremendous attention from researchers. One important module of autonomous driving is environment perception [8], [9], [10], which involves critical tasks like object detection, semantic recognition, depth complementation, and prediction [11]. These tasks are essential for an autonomous driving system to accurately understand its surroundings, and they heavily rely on data gathered from various sensors mounted on the vehicle, as shown in Figure 1.1. Under this circumstance, sensor fusion techniques allow combining data from multiple sensors for more accurate, reliable and comprehensive environment sensing [12]. This amalgamation of data from different sources, such as cameras, LiDAR (Light Detection and Ranging), Radar (Radio Detection and Ranging), and ultrasonic sensors, allows the system to compensate for the limitations of individual sensors. For instance, while cameras provide detailed visual information, they can be affected by lighting conditions or obstructions. LiDAR sensors offer precise distance measurements but can struggle in adverse weather conditions. By fusing data from these diverse sources, an autonomous vehicle can form a more robust and detailed understanding of its surroundings [13], which is critical for safe navigation and decision-making.

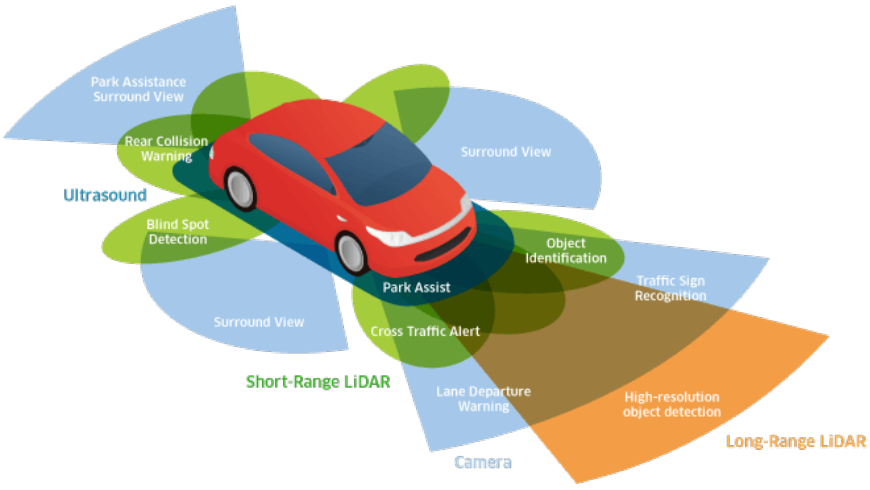


Figure 1.1: Sensor fusion application scenarios. [1]

Multi-sensor fusion techniques, renowned for their ability to provide a more robust and synergistic perception of the environment, have found extensive application in

various industries [14]. These techniques bring together data from different sensors, such as cameras, LiDAR, and IMUs (Inertial Measurement Units) to create a comprehensive understanding of the surroundings. However, implementing these techniques involves overcoming several challenges. Extrinsic calibration is one of the critical issues in multi-sensor fusion systems, for example, Camera-LiDAR calibration, Camera-IMU calibration, and LiDAR-IMU calibration. The characteristics and application scenarios

Table 1.1: Characteristics of common vehicle-mounted sensors.

Category	Advantages	Limatations	Main applications
Camera	High resolution; Color and texture recognition;	Limited field-of-view; Sensitive to light and weather (fog);	Object detection and tracking; (e.g. pedestrian and traffic signs)
Mmw-Radar	Wide range; Strong penetration; Not easily affected by light and weather;	Not sensitive to horizontal objects; Unimageable and only 2D detection;	Adaptive Cruise Control (ACC); Anti-collision system;
LiDAR	Wide field-of-view and range; Not easily affected by light;	Low resolution; High cost; 3D object detection; Sensitive to rain and small interference;	Object detection and tracking; Simultaneous localization and mapping
IMU	Velocity and acceleration provided; Accurate location; High working frequency;	Unable to recognize objects; Unable to provide surroundings perception;	Navigation and path planning; Simultaneous localization and mapping;
Ultrasound	Strong detection ability at close range; Low cost and small size; Not easily affected by light and weather;	Limited range detection; Limited field-of-view; Slow response; Sensitive to object materials;	Blind spot monitoring; Parking assist;

of some of the main in-vehicle sensors: camera, radar, LIDAR, ultrasound, IMU/GPS are presented in Table 1.1. In multi-sensor autonomous driving systems, cameras and LiDAR sensors can provide complementary information to enhance the capabilities of various applications. Cameras offer color and texture information, while LiDAR provides accurate depth information. By calibrating these sensors together, a more comprehensive and accurate perception of the environment can be achieved, which is crucial for tasks such as object detection, tracking, and scene understanding in autonomous vehicles and robotics. Camera-LiDAR calibration helps in associating the data from both sensors correctly. Matching 2D image features with their corresponding 3D points in the LiDAR data can be challenging without proper calibration. Accurate data association is essential for tasks like Simultaneous Localization And Mapping (SLAM) [15]. Calibration can also mitigate the limitations and inaccuracies of individual sensors. For example, camera distortion and lens effects can be corrected, and LiDAR alignment issues can be resolved, which can increase the overall robustness and accuracy of the sensor suite [16]. To sum up, an accurate extrinsic calibration is a prerequisite for many multi-sensor fusion tasks.

1.1 Sensor Calibration

Sensor calibration involves both temporal calibration (clock synchronization) and spatial calibration. Usually, the system needs to be time-calibrated before spatial calibration can be completed. Sensor spatial calibration can be divided into intrinsic calibration and extrinsic calibration in a multi-sensor system, both stand for different functions and are critical for accurate perception and navigation [17].

1.1.1 Intrinsic Calibration

In the camera-LiDAR calibration system, intrinsic calibration determines the internal optical characteristics of a camera sensor, and these characteristics include the focal length, optical center, and lens distortion coefficients of the camera or optical sensors. These parameters are inherent to the sensor and are independent of the position or orientation in which it is placed. Through intrinsic calibration, we can correct image distortions due to the optical nature of the sensor and ensure the accuracy of the measured data.

1.1.2 Camera-LiDAR Extrinsic Calibration

Extrinsic calibration establishes the spatial transformation between different sensors or between a sensor and the world, which is critical for sensor fusion, where data from different sensors need to be combined to create a coherent model of the environment. This usually includes the position and orientation of the sensor relative to a known reference frame. The camera-LiDAR extrinsic calibration problem involves determining the spatial relationship between the camera sensor system and the LiDAR sensor system. The transformation of the LiDAR coordinate system to the camera coordinate system can be achieved by a combination of rotations and translations, which describe the orientation and position of the LiDAR coordinate system with respect to the camera coordinate system.

In autonomous vehicles, intrinsic calibration ensures that the data from each sensor is accurate and can be trusted for detailed analysis. Extrinsic calibration allows the vehicle to integrate this data into a unified model of its surroundings. For instance, the vehicle can overlay the precise distance measurements from a LiDAR sensor onto the visual data from cameras to understand the environment in detail and make informed navigation decisions.

1.2 Motivation

The motivation for Camera-LiDAR extrinsic calibration lies in the need to accurately align and synchronize data from cameras and LiDAR sensors in a multi-sensor setup. This calibration is crucial for applications such as autonomous vehicles, robotics, and augmented reality, where precise spatial alignment between the camera and LiDAR

is essential. By determining the accurate transformation between the coordinate systems of the camera and LiDAR, one can seamlessly fuse information from both sensors, enabling more effective perception, object recognition, and scene understanding. Calibration ensures that the data from each sensor is correctly mapped to a common reference frame, facilitating coherent and accurate analysis in various applications.

Calibration plays a key role in ensuring that the data from each sensor is accurately projected onto a unified reference frame. This alignment is critical for conducting a coherent and precise analysis across a spectrum of applications. In autonomous driving, for example, the synergy of camera and LiDAR data facilitates advanced navigation capabilities, allowing for the accurate detection and classification of various road elements, obstacles, and dynamic entities like pedestrians and other vehicles. In robotics, this calibration is integral for tasks requiring spatial awareness and interaction with complex environments. Similarly, in augmented reality applications, aligning real-world and virtual elements seamlessly is imperative for creating immersive and realistic experiences.

Furthermore, the process of Camera-LiDAR extrinsic calibration involves sophisticated algorithms and precise measurements. It typically includes the use of calibration patterns or environments and advanced software to calculate the relative positions and orientations of the sensors. This calibration process not only compensates for physical misalignments but also accounts for the intrinsic characteristics of each sensor type, such as the camera's lens distortions and the LiDAR's range accuracy.

Traditional methods of Camera-LiDAR calibration involving calibration plates and manual selection of features have long been the standard approach in the field. These methods, while effective, are often labor-intensive, time-consuming, and reliant on controlled laboratory environments. However, recent advancements in Camera-LiDAR calibration have sparked a paradigm shift, with a growing emphasis on exploring automatic, automatic, and targetless calibration approaches that offer several advantages over their traditional counterparts.

1.3 Related Works

In this section, we delve into the evolution of extrinsic calibration between a camera and LiDAR, along with a review of relevant research. The prevailing extrinsic cali-

Table 1.2: Categories of Camera-LiDAR extrinsic calibration methods.

Category	**Target-based**	**Targetless**
Manual	*Existing* calibration reference target; *Artificial* feature selection.	*Non-existent* calibration reference target; *Artificial* feature selection.
Automatic	*Existing* calibration reference target; *Algorithmic* feature extraction.	*Non-existent* calibration reference target; *Algorithmic* feature extraction.

bration techniques predominantly center on one-shot extrinsic estimation. Table 1.2 shows the categories of Camera-LiDAR extrinsic calibration methods. According to [3], these methods can be broadly classified into two categories: target-based and targetless calibration, contingent on the presence of an extrinsic target as the reference

point. Additionally, they can be further categorized as automatic or manual calibration, depending on whether features are extracted automatically.

1.3.1 Target-based Calibration Methods

The manual target-based extrinsic parameter calibration approach requires engineers to manually specify the correspondence between the LiDAR point cloud and the camera image.

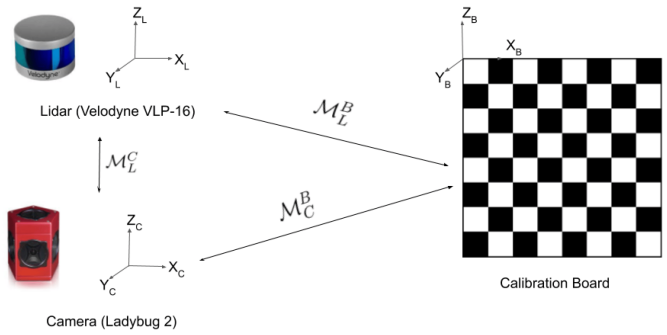


Figure 1.2: Principle of extrinsic calibration using the chessboard as a reference target. [2]

This process can be aided by calibration boards, an example is a checkerboard as shown in Figure 1.2, which enforces geometric constraints between 3D points in the point cloud and corresponding image pixels. This enables accurate estimation of the extrinsic parameters. Zhang and Pless [18] first proposed a spatial calibration framework between a camera and LiDAR. This method uses a chessboard as the calibration target and imposes geometric constraints on the relative pose according to the position of the visible laser point on the chessboard to estimate the extrinsic parameters. Unnikrishnan and Hebert [19] also used a checkerboard as a target, manually selecting 3D laser points along the edges of the checkerboard pattern, and aligning these points with the checkerboard plane in the camera image. These specified calibration targets impose geometric constraints between corresponding 3D points in point clouds and pixels in images, which enable the agent to estimate extrinsic parameters.

Manual target-based calibration methods require not only extrinsic targets but also a manual selection of 'feature points'. An automatic target-based calibration method with targets was proposed by Geiger et al. [20] applying a corner point detection algorithm on checkerboards as alignment feature points, avoiding manually selecting feature points. Nunez et al. [21] modified on [18], using IMU (Inertial Measurement Unit) sensor to decrease the number of points needed for a robust calibration. The laser points are moved in three different planes to form a set of virtual 3D points, which are verified by the powerful RANSAC planar analysis. Toth et al. [22] used a spherical target for automatic extrinsic parameter calibration, estimating the extrinsic parameters through geometric constraints on the same sphere center.

1.3.2 Automatic Targetless Calibration Methods

The accuracy of traditional target-based extrinsic calibration depends heavily on the quality and precise placement of the targets used in the process. Calibration involves the physical setting of the target and possible multiple measurements for verification, which is very time-consuming. At the same time, this process requires the experimenter to have experience in operating the sensor and certain domain knowledge to ensure that the calibration points are accurately set and measured. Hence, More and more scholars are turning to targetless extrinsic parameter calibration. Scaramuzza et al. [23] manually select pairs of points between 3D LiDAR points and image pixels and then estimate extrinsic parameters by solving a PnP (Perspective-from-n-Points) problem. However, this manual targetless approach still does not get rid of manual participation, and the selection of feature points is subjective.

In recent years, automatic and targetless methods have gradually emerged, which no longer rely on extrinsic reference targets. According to [3], automatic targetedless methods can be divided into four categories based on the way information is extracted from the environment, information theory-based methods, feature-based methods (or appearance-based methods), ego-motion-based methods, and deep-learning-based methods. In the following subsections, We will focus on discussing the different types of automatic and targetless methods.

1.3.2.1 Information-based Methods

The method based on information theory estimates extrinsic parameters by maximizing the similarity transformation between the LiDAR sensor and the camera. The basic principle is as follows:

$$\hat{\mathbf{T}} = \arg\max_{\mathbf{T}} MI((proj_{\mathbf{T}}(\mathbf{P}^L), \mathbf{I}) \tag{1.1}$$

where $\mathbf{P}^L$ denotes the 3D points generated by the LiDAR, $\mathbf{I}$ denotes the camera image, $proj_{\mathbf{T}}$ describe the project function from the set of 3D points to the image w.r.t. the extrinsic matrix $\mathbf{T}$, and $MI(\cdot)$ denotes the corresponding information metric that measures the similarity between two data distributions.

Pandey et al. [24] estimate the extrinsic parameters by maximizing the mutual information between the point cloud and image features, where the marginal and joint probabilities of two sensor features can be obtained from the normalized marginal and joint histograms of the reflectivity and grayscale intensity values of the points co-observed by the laser scanner and camera. Similarly, Taylor and Nieto use the normalized mutual information (NMI) between the surface normal of point cloud and gray-scale intensity of image [25], and they further combined this method with the NMI of reflectivity and gray-scale intensity [26]. On this basis, Irie et al. [27] additionally used the statistical characteristics of point cloud depth discontinuity and image edge, and used bagged least-squares mutual information (BLSMI) instead of NMI as a measure.

1.3.2.2 Motion-Based Methods

Ego-motion based methods exploit the motion transformation of sensors mounted on the traveling vehicle to estimate the extrinsic parameter. Hand-eye calibration problem is a classic approach to solving extrinsic calibration, the transformation between the camera and the gripper is calculated by solving the equation $AX = XB$, as illustrated in Figure 1.3.

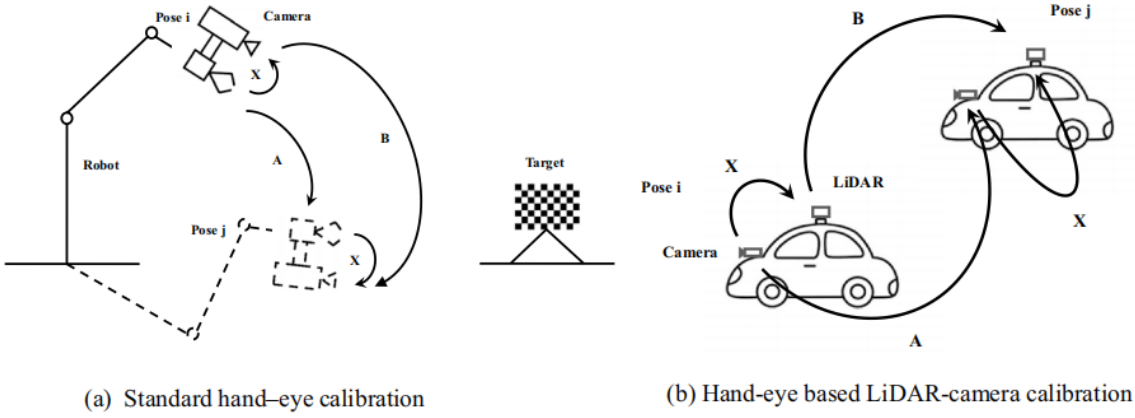


Figure 1.3: Hand-eye calibration physical model. [3]

Taylor and Nieto [28] use iterative closed points (ICP) and structure-from-motion (SfM) algorithms to estimate the motion of LiDAR and camera respectively, and solve the extrinsic transformation by observing the same motion of both sensors. In their further work, [29], they use visual odometry to obtain the camera's motion and combine scene vision on the motion-based calibration framework. Meanwhile, they designed the hand-eye equations from a probabilistic approach to include the uncertainty in each sensor reading, which allows for the simultaneous calibration of all sensors and helps to estimate the uncertainty of the final calibration. Hand-eye-based methods do not require initial calibration parameters and overlapping fields of view. However, the accuracy of motion estimation for the sensors affects the performance of ego-motion-based methods.

1.3.2.3 Feature-Based Methods

Feature-based automatic targetless calibration methods aim to extract and match common features of LiDAR point cloud and camera images. These features usually represent stable geometric or semantic elements in the surrounding environment, such as lines, corners, edges, and more semantic elements. González-Aguilera et al. use the Förstner operator [30] to extract and match corner and circular points from a camera image and point cloud projected image. Scale-invariant feature transform (SIFT) is a popular operator for detecting and matching local features in images. Point features extracted by SIFT are invariant to image translation, scaling, and rotation. Böhm

and Becker [31] use SIFT to define descriptor similarity, and estimate the extrinsic parameters when maximizing such similarity. However, SIFT and SURF rely on texture information for feature extraction, and these methods may have difficulty finding matching features in environments lacking texture (e.g., smooth walls).

Levison and Thrun [32] extracted 'edges' and aligned them through a defined spatial geometrical relation, where edges are recognized from the points by calculating the differences in depth between neighboring points and filtering out points whose values of differences are below a preset threshold. Castorena et al. [33] further extended the edge extraction by first generating a dense depth map by upsampling the point cloud, and then extracting the edges by calculating gradient changes in depth. The challenge of edge alignment is that the characteristics of each sensor modality (such as sampling mode and measured information) are significantly different, and the extracted edge emphasis will also be different. Continuous edge extraction of point clouds is difficult to implement in 3D. Zhang et al. [7] converted the 3D-2D edge alignment problem to 2D-2D edge alignment, projected the point cloud scan onto the surface of the cylinder and expanded it, and added a multi-feature density map to optimize the edge extraction process.

Another efficient way is by matching the semantic information of the scene, nevertheless, the extraction of semantic features is more difficult than the extraction of geometric features and is hard to implement with simple mathematical operators. Zhu et al. [34] employ the Pyramid Scene Parsing Network (PSPNet) to semantically segment each camera frame and use it to construct an optimization objective for extrinsic parameter estimation. The strength of the semantic edges is emphasized by applying a distance transform to the mask and an inverse distance transform to the background. Smooth and continuous cost functions can be obtained when the point cloud falls into different positions of the mask. However, poor segmentation quality or coarse LiDAR scan may degrade the calibration accuracy. Liu et al.[35] applied pre-trained SPVNA and pre-trained SDCNet for semantic segmentation of point clouds and images respectively, and designed a differentiable semantic alignment loss using gGradient-based optimization to estimate the extrinsic parameters. This requires that both networks need to share the same labels, and the semantic alignment loss is formulated to be insensitive to translation errors due to the limitations of pre-trianed models. Luo et al. [36] apply the Segmented Anything Model (SAM) on the images to optimize the extrinsic parameters by maximizing the consistency of the points projected within each image mask, where the point cloud consistency is described by a combination of intensities, normal vectors, and classes derived from certain segmentation methods.

1.3.2.4 Deep-Learning-based Methods

Recently, deep learning has made breakthroughs in automatic feature engineering and achieved excellent performance on multiple tasks. Some use deep learning networks to effectively extract semantic features in images and point clouds for registration, such as SOIC [34], SemAlign [37], and SemCal [35], some are end-to-end models such as RegNet [38], CalibNet [39], LCCNet [40] and DeepI2P [41]. However, the development

and computing costs of end-to-end deep learning models are extremely high, require a large amount of data training, and are often difficult to implement in actual projects.

1.3.3 Summary

Traditional methods of Camera-LiDAR calibration involving calibration plates and manual selection of features have long been the standard approach in the field. These methods, while effective, are often labor-intensive, time-consuming, and reliant on controlled laboratory environments. However, recent advancements in Camera-LiDAR calibration have sparked a paradigm shift, with a growing emphasis on exploring automatic, automatic, and targetless calibration approaches that offer several advantages over their traditional counterparts. Feature extraction based on geometric information and feature extraction based on semantics each have their own merits. Geometric information can be described by intuitive mathematical formulas, and the implementation is relatively simple. Semantic information usually relies on small and medium-sized networks. Using pre-trained networks can avoid using a large amount of data for training. Although semantic information can identify categories in the scene, the edges of the mask are often ambiguous. Our approach combines semantic and geometric features, similar to [36], where the gradient edges of the image are blended with semantic edges to match the blended features of the point cloud.

1.4 Objectives and Improvements

In this study, we mainly focus on the development and analysis of autonomous, targetless extrinsic calibration algorithms for camera-LiDAR systems. We explore various related algorithms and present an enhanced version that draws inspiration from the recent work [7], incorporating several improvements. Our experimental validation and performance assessments are carried out using the well-regarded and widely recognized KITTI dataset, renowned for its relevance and authority in the field. The main contributions of this paper are as follows:

- An easy-to-implement edge-based feature registration algorithm is proposed.
- Involving semantic edges using an advanced SMA model on image edge extraction.
- Edges based on distance discontinuity and edges based on projection map completion are simultaneously considered in point cloud edge extraction.
- Using fast density completion on point cloud projection map completion.
- Solving the PnP problem using semantic-based prime thus avoiding random initial settings on extrinsic parameters estimation.
- Multi-frame weighted average and the scene similarity is proposed and explored.

1.5 The Outline of Thesis

The outline of the thesis is shown below:

- Chapter 1: Introduction
- Chapter 2: Problem Formulation: Camera-LiDAR extrinsic calibration framework
- Chapter 3: Proposed methods
- Chapter 4: Experiments and results
- Chapter 5: Conclusions and future work

In the opening chapter of this thesis, we delve into the research context and underscore the significance of calibrating extrinsic parameters between cameras and LiDAR. This sets the stage for a comprehensive exploration of the topic. Then we present a comparative analysis of existing literature on extrinsic parameter calibration. Here, we scrutinize both non-automatic and automatic methods, as well as targeted and untargeted approaches. The emphasis, however, is placed on automatic untargeted algorithms, discussing their nuances in detail. Chapter 2 is dedicated to explicating the sensor coordinate systems and the mathematical framework underpinning the extrinsic reference calibration of the camera and LiDAR. This chapter serves as the bedrock for the development and explanation of the algorithm proposed in Chapter 3. Chapter 4 involves rigorous experimentation and analysis using the KITTI dataset. This empirical investigation aims to validate the theoretical models and algorithms discussed in previous chapters. In the final chapter, a macro-analysis of the whole work in terms of future improvements will be reflected.

Problem Formulation: Camera-LiDAR Extrinsic Calibration Framework

2

This chapter introduces the fundamental concepts and methodologies that underpin the integration of camera and LiDAR systems. We begin by exploring the basic principles of camera coordinates and LiDAR set-up parameters, laying the groundwork for understanding how these two distinct technologies can complement each other. The camera's role in capturing high-resolution imagery and LiDAR's proficiency in generating precise 3D spatial data is pivotal in creating a comprehensive perception system.

The intricacies of imaging and the mechanics of 3D space rotation and translation are then introduced to grasp how images and spatial data are transformed and interpreted in a unified system. The ability to convert and understand these transformations is central to effectively merging the visual and spatial data. We then transition to exploring general 3D-3D and 3D-2D transformations, which are key to aligning and integrating data from different sensor modalities.

Lastly, the chapter focuses on the specific process of projecting LiDAR data onto camera imagery. This projection is a critical step in sensor fusion, as it allows for the direct combination of LiDAR's spatial measurements with the camera's visual data. The content discussed in this chapter is foundational to developing a camera-LiDAR calibration algorithm.

2.1 Camera Coordinate System and Imaging Principle

Before discussing the LiDAR to image projection, we first introduce the coordinate system conversion from the camera coordinate to the image plane. Suppose there is a point $[x^C, y^C, z^C]^T$ in space, which is the same as the presentation in the camera coordinate system, where Z is the vertical distance from the point to the camera optical center (origin). Let the intersection of this point with the image plane be point $[x^I, y^I]^T$. The simulation of the projection of a point in space to the image plane is shown in Figure 2.1, from which, we can find the relation between a camera point and its imaging point:

$$x^I = \frac{f_x x^C}{z^C},\ y^I = \frac{f_y y^C}{z^C} \tag{2.1}$$

However, the real center of the image coordinate system is in the upper left corner of the image, while Equation 2.1 assumes that the origin is at the center of the image. To deal with the offset, let the coordinates of the pixel corresponding to the optical center on the image be (c_x, c_y), which is also referred to as the principal point offset.

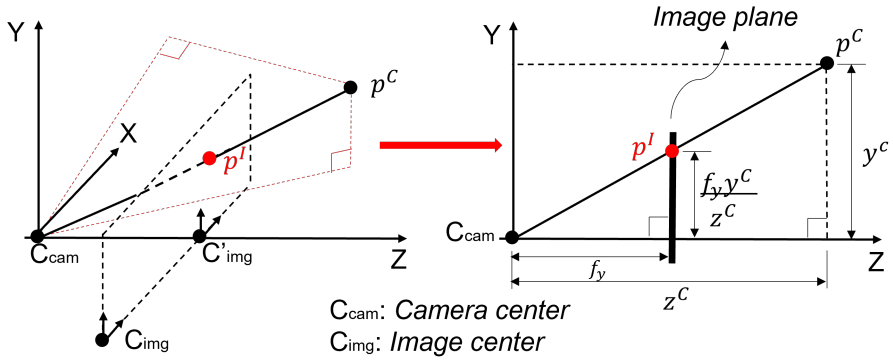


Figure 2.1: The general imaging principle of projecting a point in space from the camera coordinate system to the image plane

Then, the modified formula of Equation 2.1 is given by

$$x^I = \frac{f_x x^C}{z^C} + c_x, \; y^I = \frac{f_y y^C}{z^C} + c_y \tag{2.2}$$

which is equivalent to

$$z^C \begin{bmatrix} x^I \\ y^I \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x^C \\ y^C \\ z^C \end{bmatrix} \tag{2.3}$$

where $[x^I, y^I]^T$ is the coordinate of the projected on the image, and $\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$ is usually referred to as the intrinsic parameter matrix.

2.2 LiDAR Coordinate System and Point Cloud Scanning Model

Point cloud refers to a data set of spatial points scanned by a 3D LiDAR device. Each point cloud contains 3D coordinates (x, y, z) and reflection intensity (r), where reflection intensity is related to the surface material of the target object, the laser incident angle, the laser wavelength, and the energy density of the LiDAR.

Generating a 3D point cloud image requires only the point cloud coordinates vector $[x^L, y^L, z^L]^T$ of each scanned point. Using function `points3d` from `Mayavi` to generate an interactive point cloud panorama, the effect is as shown below

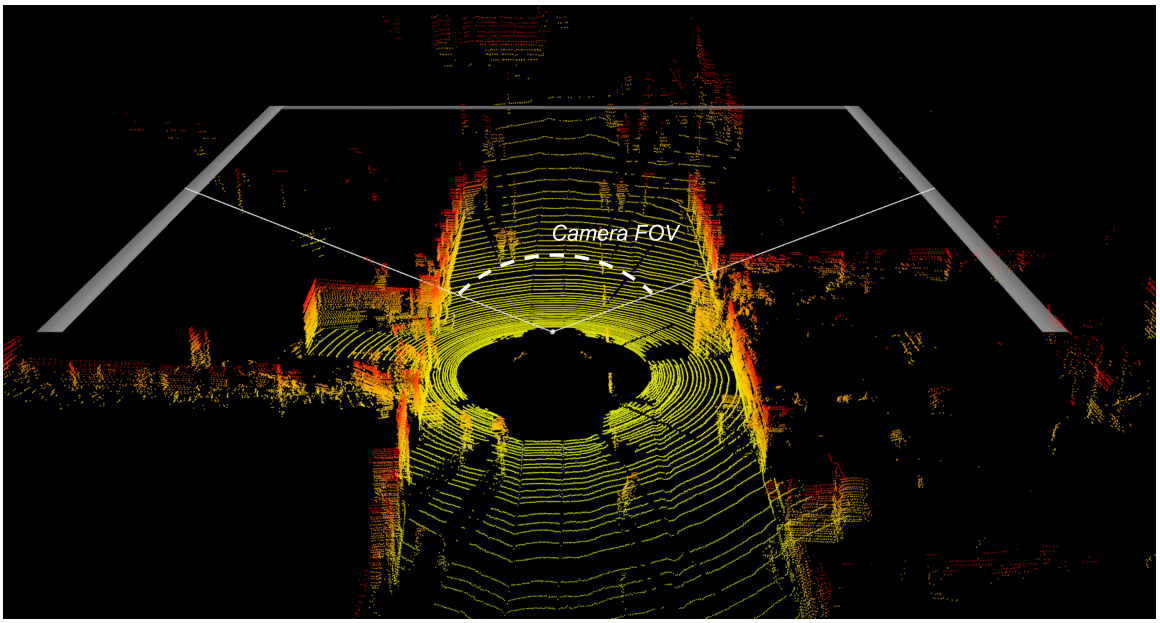


Figure 2.2: LiDAR point cloud panorama view

LIDAR devices emit a series of laser pulses and calculate the position and distance of an object based on sending laser pulses and measuring the time it takes for those pulses to reflect back from the object. LIDAR systems include rotating components, and the rotating components of LIDAR allow the laser to emit pulses at different angles, thus enabling it to scan the entire surrounding environment. LiDAR set-up has several important parameters that affect the sparsity and distribution of point clouds. Table 2.1 explains these device parameters. The number of channels and angular resolution

Table 2.1: Important LiDAR parameters and their physical meaning.

Device parameters	**Description**
Number of Channels	The number of laser beams in vertical direction
Vertical Resolution	The angular resolution in the vertical direction
Horizontal Resolution	The angular resolution in the horizontal direction
Horizontal Field of View (FoV)	The total angular coverage in the horizontal direction
Vertical Field of View (FoV)	The total angular coverage in the vertical direction
Range	The furthest observable distance of LiDAR

affect the sparseness of the point cloud, and the FOV and range affect the observation range of the LiDAR. Figure 2.3 shows the LiDAR scanning model.

As mentioned before, LiDAR point cloud data have an uneven spatial distribution. Besides, due to the self-occlusion of three-dimensional objects in the scene, LiDAR can only detect a part of the surface of the three-dimensional object, which also leads to the fact that point cloud data can only represent a part of the geometric information of the object.

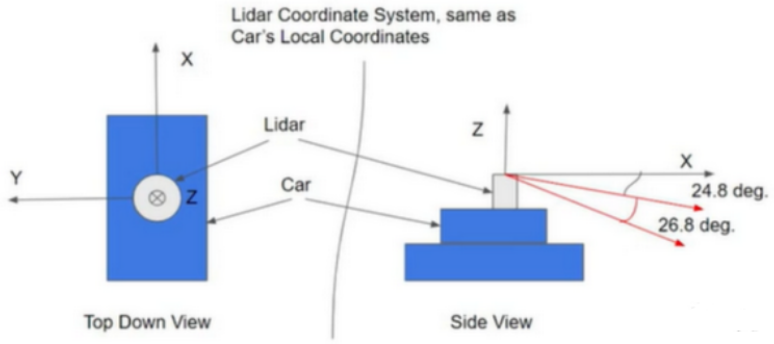


Figure 2.3: LiDAR scanning model.

Although the point cloud map is not as visual as the image, the scanned points can roughly record the size and shape of the obstacle and the surroundings and terrain. This characteristic can be used to cluster or remove certain characteristic surfaces through the characteristics of point cloud distribution.

2.3 6-DOF Rigid Body Transformation in 3D space

The coordinates transformation between two different three-dimensional space coordinate systems usually entails a combination of three-dimensional rotation and translation, which is exactly a 6-DoF (Six degrees of freedom) problem. Describing the translation is straightforward and requires only a vector, denoted as $\mathbf{t} = [x, y, z]^T$. However, describing the rotation in three-dimensional space is more complicated. There are several ways to express a rotation in three-dimensional space, including Euler angles, axis-angle representation (rotation vector), quaternion, and rotation matrix. Among them, the rotation matrix $\mathbf{R} \in \mathbb{R}^{3\times3}$ can be uniquely converted from other representation ways. Each representation has its advantages and disadvantages, depending on the specific application and requirements. In this section, we will discuss several mathematical ways of describing rotations and their conversion to the rotation matrix. In particular, some important properties of the rotation matrix will be introduced. Finally, the transformation of points in 3D space will be explained to help us formulate extrinsic parameter problems.

2.3.1 Euler Angles and Rotation Matrix

A concise way to express rotations is by decomposing them into three fundamental rotations, as illustrated in Equations 2.4–2.6. These equations represent rotations around the x-, y-, and z-axes within the right-handed Cartesian coordinate system, respectively.

$$\mathcal{R}_x(\gamma) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{bmatrix} \tag{2.4}$$

$$\mathcal{R}_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \tag{2.5}$$

$$\mathcal{R}_z(\alpha) = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.6}$$

where γ, β, and α are also called *roll*, *pitch* and *yaw*. Figure 2.4 shows the geometric interpretation of the 6 degrees of freedom problem.

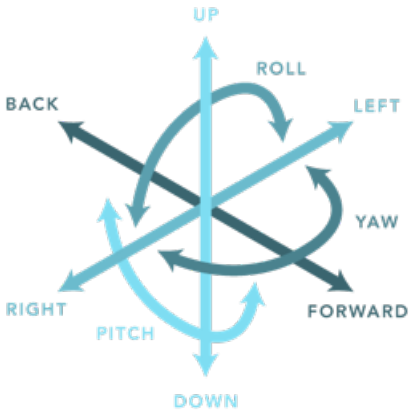


Figure 2.4: Axis representation of rotation and translation in 6-DOF rigid body transformation.

Any 3-dimensional rotation matrix $\mathbf{R} \in \mathbb{R}^{3\times3}$ can be used with these three angles to characterize, and can be expressed as the product of the three basic rotation matrices in Equation 2.7. That is, first rotate around the z-axis by α degrees, then rotate around the y-axis by β degrees, and finally rotate around the x-axis by γ degrees.

$$\begin{aligned} \mathbf{R}(\alpha,\beta,\gamma) &= \mathcal{R}_z(\alpha)\mathcal{R}_y(\beta)\mathcal{R}_x(\gamma) \\ &= \begin{bmatrix} \cos\alpha\cos\beta & \cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma & \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma \\ \sin\alpha\cos\beta & \sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma \\ -\sin\beta & \cos\beta\sin\gamma & \cos\beta\cos\gamma \end{bmatrix} \end{aligned} \tag{2.7}$$

where $\alpha, \beta, \gamma \in [0, \pi)$. This rotation representation is also called intrinsic rotation whose Tait–Bryan angles are α, β, γ, about axes z, y, x, respectively. Similarly, we have extrinsic rotation as:

$$\begin{aligned} \mathbf{R}(\alpha,\beta,\gamma) &= \mathcal{R}_x(\gamma)\mathcal{R}_y(\beta)\mathcal{R}_z(\alpha) \\ &= \begin{bmatrix} \cos\beta\cos\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma & \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma \\ \cos\beta\sin\gamma & \sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & \cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma \\ -\sin\beta & \sin\alpha\cos\beta & \cos\alpha\cos\beta \end{bmatrix} \end{aligned} \tag{2.8}$$

which represents an extrinsic rotation whose Euler angles are α, β, γ, about axes x, y, z, respectively. Equation 2.8 is actually equivalent to Equation 2.7. The order of matrices product is from left to right and not commutative.

For the Euler angle representation using the *z-y-x* rotation order, the transformation from the rotation matrix to the Euler angle can be calculated by Equation 2.9 to 2.12.

$$\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{2.9}$$

where r_{ij} denotes the element of the rotation matrix, where i denotes the row number and j denotes the column number. Each row and column represents the rotation component on different axes.

$$\alpha = \arctan(r_{21}, r_{11}) \tag{2.10}$$

$$\beta = \arcsin(-r_{31}, \sqrt{r_{31}^2 + r_{33}^2}) \tag{2.11}$$

$$\gamma = \arctan(r_{32}, r_{33}) \tag{2.12}$$

It should be noted that when using Euler angles to represent rotation, when β angle approaches ± 90 degrees, there may be situations where the pose cannot be uniquely determined. In this case, only the sum or difference of α and γ can be found. Around this angle, the rotation around the *yaw* axis and the rotation around the horizontal axis (*roll*) will be coupled, resulting in the loss of the ability to control independently. This situation is called Gimbal Lock.

2.3.2 Axis-Angle Representation and Quaternion

Rotation can also be determined by a unit vector $\mathbf{e} = [e_x, e_y, e_z]^T$ that represents the direction of the axis of rotation, and an angle of rotation around this axis θ. By Rodrigues' rotation formula, the rotation matrix determined by the axis and angle is given by

$$\begin{aligned} &\mathbf{R}(\mathbf{e}, \theta) \\ &= \begin{bmatrix} \cos\theta + (1-\cos\theta)e_x^2 & (1-\cos\theta)e_x e_y - (\sin\theta)e_z & (1-\cos\theta)e_x e_z + (\sin\theta)e_y \\ (1-\cos\theta)e_y e_x + (\sin\theta)e_z & \cos\theta + (1-\cos\theta)e_y^2 & (1-\cos\theta)e_y e_z + (\sin\theta)e_x \\ (1-\cos\theta)e_z e_x + (\sin\theta)e_y & (1-\cos\theta)e_z e_y + (\sin\theta)e_x & \cos\theta + (1-\cos\theta)e_z^2 \end{bmatrix} \\ &= \exp\left(\theta \begin{bmatrix} 0 & -e_z & e_y \\ e_z & 0 & -e_x \\ -e_y & e_x & 0 \end{bmatrix}\right) \end{aligned} \tag{2.13}$$

Although the axis-angle representation of rotation only requires one axis and the angle of rotation around that axis, there are actually four parameters involved in forming the rotation matrix. Similarly, a rotation can also be defined using a quaternion. A quaternion is defined by

$$\mathbf{q} = q_0 + q_1 i + q_2 j + q_3 k \tag{2.14}$$

where q_0 is a real value, and q_1, q_2, q_3 are complex values. The corresponding rotation matrix is

$$\mathbf{R}(\mathbf{q}) = \begin{bmatrix} 1-2(q_2^2+q_3^2) & 2(q_1 q_2 - q_0 q_3) & 2(q_1 q_3 + q_0 q_2) \\ 2(q_1 q_2 + q_0 q_3) & 1-2(q_1^2+q_3^2) & 2(q_2 q_3 - q_0 q_1) \\ 2(q_1 q_3 - q_0 q_2) & 2(q_2 q_3 - q_0 q_1) & 1-2(q_1^2+q_2^2) \end{bmatrix} \tag{2.15}$$

2.3.3 Properties of Rotation Matrix

Rotation matrix has several important properties that are crucial to understanding and applying rotation operations in 3D space. For $\mathbf{R} \in \mathbb{R}^{3\times3}$, The following are some basic properties of rotation matrices:

Property 1: The rotation matrix has orthogonality, and its row vectors and column vectors are unit vectors and are orthogonal to each other. The orthogonal property maintains the length and angle of vectors and does not change the relative relationship of vectors in space.

$$\mathbf{R}^T\mathbf{R} = \mathbf{U} \tag{2.16}$$

Property 2: The determinant of the rotation matrix is ±1, which ensures that the volume expansion factor of the matrix is 1 and does not change the spatial volume.

$$\det|\mathbf{R}| = \pm1 \tag{2.17}$$

Property 3: The inverse of a rotation matrix is equal to its transpose. The inverse matrix realizes the undoing of the rotation and returns to the state before the rotation.

$$\mathbf{R}^T = \mathbf{R}^{-1} \tag{2.18}$$

Measuring the rotation error between two rotations, represented by matrix $\mathbf{R}_1$ and $\mathbf{R}_2$ is not a simple subtraction. Consider one point is rotated through $\mathbf{R}_1$, and the other point is rotated through $\mathbf{R}_2$. The former needs to reach the same state as the latter point after undergoing a rotation $\Delta\mathbf{R}$. Utilizing Equations 2.16 and 2.18, the 'difference' between two rotations $\mathbf{R}_1$ and $\mathbf{R}_2$ can be determined by

$$\Delta\mathbf{R} = \mathbf{R}_1^{-1}\mathbf{R}_2 = \mathbf{R}_1^T\mathbf{R}_2 \tag{2.19}$$

where $\Delta\mathbf{R}$ serves as a metric for measuring the error between the two rotations.

2.3.4 General Conversion Between 3D Coordinate Systems

The transformation of the 3D coordinate system is a 6-degree-of-freedom problem, represented by rotation and translation. With a clear definition of rotation and relation with Euler angles, transforming a three-dimensional point (x, y, z) from coordinate system A to coordinate system B is given by

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}^B = \mathbf{R} \begin{bmatrix} x \\ y \\ z \end{bmatrix}^A + \mathbf{t} \tag{2.20}$$

where $\mathbf{R}$ is rotation matrix, and $\mathbf{t}$ is translation matrix.

2.4 LiDAR to Camera Image Projection

Sections 2.1 and 2.2 have established that camera images and LiDAR point cloud data are captured in distinct coordinate systems. Consequently, when integrating these images with point clouds for sensor fusion, it is essential to perform a coordinate transformation, which is Camera-LiDAR calibration. From Figure 2.5 we can see that:

Camera image coordinates:

- The coordinate values in an image are always positive.
- The origin is located in the upper left-hand corner of the image.
- The coordinates take integer values only.

LiDAR point cloud coordinates:

- The coordinate values in the point cloud can be positive or negative.
- The origin is located in the center of the 3D point cloud space (LiDAR coordinate system).
- The coordinates can take any real numbered values.

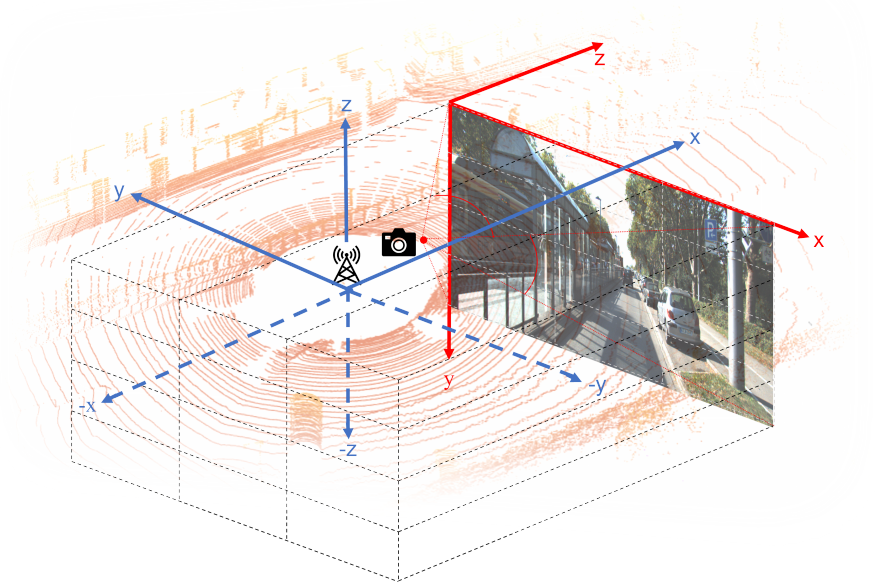


Figure 2.5: Camera and LiDAR coordinate system

Now we consider the situation that the coordinates of a point in the world coordinate system are different from its representation in the camera coordinate system. In such cases, it's essential to first transform the other coordinates into the camera coordinate system before projecting the point onto the image plane. From Equation 2.3, we already

have the conversion formula from the camera to the image plane, based on this, the general coordinate system conversion formula from the world to the image plane is as follows

$$z^C \begin{bmatrix} x^I \\ y^I \end{bmatrix} = \mathbf{K}[{}^C_W\mathbf{R} \begin{bmatrix} x^W \\ y^W \\ z^W \end{bmatrix} + {}^C_W\mathbf{t}] \tag{2.21}$$

where $[x^W, y^W, z^W]^T$ is a vector of coordinates of a point in the world coordinate system. For the sake of clarity, $(\cdot)^b_a$ refers to operation $(\cdot)$ from a to b, this definition is also adopted in the following sections. ${}^C_W\mathbf{R} \in \mathbb{R}^{3\times3}$ is the relative rotation between the world coordinate system and the camera coordinate system, and accordingly ${}^C_W\mathbf{t} \in \mathbb{R}^{1\times3}$ is the relative translation vector between the world coordinate system and the camera coordinate system.

Rewrite Equation 2.21 using the homogeneous coordinates form, we have

$$z^C \begin{bmatrix} x^I \\ y^I \\ 1 \end{bmatrix} = \mathbf{K} \left[{}^C_W\mathbf{R}|{}^C_W\mathbf{t}\right] \begin{bmatrix} x^W \\ y^W \\ z^W \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{xx} & r_{xy} & r_{xz} & t_x \\ r_{yx} & r_{yy} & r_{yz} & t_y \\ r_{zx} & r_{zy} & r_{zz} & t_z \end{bmatrix} \begin{bmatrix} x^W \\ y^W \\ z^W \\ 1 \end{bmatrix} \tag{2.22}$$

Define the notation of used calibration parameters for convenience as follows:

- $\mathbf{P}^{(i)}_{rect} \in \mathbb{R}^{3\times4}$ ..... Projection matrix after rectification
- $\mathbf{R}^{(0)}_{rect} \in \mathbb{R}^{3\times3}$ ..... Rectifying rotation matrix of camera 0
- ${}^C_L\mathbf{T} \in \mathbb{R}^{4\times4}$ ..... Transformation matrix from LiDAR to camera 0
- ${}^C_L\mathbf{R} \in \mathbb{R}^{3\times3}$ ..... Rotation matrix from LiDAR to camera 0
- ${}^C_L\mathbf{t} \in \mathbb{R}^{1\times3}$ ..... Translation matrix from LiDAR to camera 0

where $i \in 0, 1, 2, 3$ is the camera index. For binocular cameras or multiple cameras, the camera index is used to number the cameras. All these parameters above are necessary for calibrating the cameras intrinsically and intrinsically.

An example of a camera calibration file is shown in Figure 2.6.

```
P0: 7.215377000000e+02 0.000000000000e+00 6.095593000000e+02 0.000000000000e+00 0.000000000000e+00 7.215377000000e+02
1.728540000000e+02 0.000000000000e+00 0.000000000000e+00 0.000000000000e+00 1.000000000000e+00 0.000000000000e+00
P1: 7.215377000000e+02 0.000000000000e+00 6.095593000000e+02 -3.875744000000e+02 0.000000000000e+00 7.215377000000e+02
1.728540000000e+02 0.000000000000e+00 0.000000000000e+00 0.000000000000e+00 1.000000000000e+00 0.000000000000e+00
P2: 7.215377000000e+02 0.000000000000e+00 6.095593000000e+02 4.485728000000e+01 0.000000000000e+00 7.215377000000e+02
1.728540000000e+02 2.163791000000e-01 0.000000000000e+00 0.000000000000e+00 1.000000000000e+00 2.745884000000e-03
P3: 7.215377000000e+02 0.000000000000e+00 6.095593000000e+02 -3.395242000000e+02 0.000000000000e+00 7.215377000000e+02
1.728540000000e+02 2.199936000000e+00 0.000000000000e+00 0.000000000000e+00 1.000000000000e+00 2.729905000000e-03
R0_rect: 9.999239000000e-01 9.837760000000e-03 -7.445048000000e-03 -9.869795000000e-03 9.999421000000e-01 -4.278459000000e-03
7.402527000000e-03 4.351614000000e-03 9.999631000000e-01
Tr_velo_to_cam: 7.533745000000e-03 -9.999714000000e-01 -6.166020000000e-04 -4.069766000000e-03 1.480249000000e-02 7.280733000000e-04
-9.998902000000e-01 -7.631618000000e-02 9.998621000000e-01 7.523790000000e-03 1.480755000000e-02 -2.717806000000e-01
Tr_imu_to_velo: 9.999976000000e-01 7.553071000000e-04 -2.035826000000e-03 -8.086759000000e-01 -7.854027000000e-04 9.998898000000e-01
-1.482298000000e-02 3.195559000000e-01 2.024406000000e-03 1.482454000000e-02 9.998881000000e-01 -7.997231000000e-01
```

Figure 2.6: Example of a calibration configuration file.

where the calibration parameter matrices for each frame are recorded in row-major order [6].

`P0` $\sim$ `P3` stores the entries of the rectifying projection matrices of camera 0, 1, 2, and 3, respectively. The projection matrix after rectification $\mathbf{P}_{rect}^{(i)}$, also referred to as the intrinsic parameter matrix, is determined only by the camera intrinsic matrix, which is given by

$$\mathbf{P}_{rect}^{(i)} = \begin{bmatrix} f_u^{(i)} & 0 & c_u^{(i)} & -f_u^{(i)} b_x^{(i)} \\ 0 & f_v^{(i)} & c_v^{(i)} & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \ i = 0, 1, 2, 3 \tag{2.23}$$

where $b_x^{(i)}$ denotes the baseline (in meters) concerning reference camera 0, $f_u^{(i)}$ and $f_v^{(i)}$ are the focal lengths of the camera, and $c_u^{(i)}$ and $c_v^{(i)}$ are the principal point offsets. The principal axis of the camera is the line perpendicular to the image plane and passing through the vacuum, and its focal point with the image plane is called the principal point. The principal point offset is the position of the principal point relative to the image plane.

`R0_rect` stores the entries of the rectifying rotation matrix of camera 0, denoted as $\mathbf{R}_{rect}^{(0)}$, which enables to make the image planes coplanar.

`Tr_velo_to_cam` stores the entries for the concatenation of the rotation matrix and translation matrix from LiDAR to camera 0. To built the transformation matrix ${}_L^C\mathbf{T}$, use the following equation

$${}_L^C\mathbf{T} = \begin{bmatrix} {}_L^C\mathbf{R} & {}_L^C\mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \tag{2.24}$$

where the rotation matrix ${}_L^C\mathbf{R}$ and the translation matrix ${}_L^C\mathbf{T}$ are together called the extrinsic parameter matrix. They together describe how to transform the point from the LiDAR coordinate system to the camera coordinate system.

LiDAR point cloud to image projection involves not only LiDAR-to-camera coordinate conversion, but also camera-to-camera calibrations. This process is divided into the following procedures:

Using homogeneous coordinates, several transformations such as translation and rotation can be easily implemented by cascading several matrix-vector multiplications. Therefore, denote the point cloud coordinates as $\mathbf{p}^L = [x^L, y^L, z^L, 1]^T$. First, a transformation from LiDAR coordinates to reference camera coordinates is performed by

$$\mathbf{x} = {}_L^C\mathbf{T}\ \mathbf{p}^L \tag{2.25}$$

Then perform camera rectification based on the reference camera (camera 0), we have

$$\mathbf{x}' = \mathbf{R}_{rect}^{(0)}\ {}_L^C\mathbf{T}\ \mathbf{p}^L \tag{2.26}$$

where $\mathbf{R}_{rect}^{(0)}$ is expanded into a 4×4 matrix by appending a fourth zero-row and column and setting $\mathbf{R}_{rect}^{(0)}(4,4) = 1$. Distortion may be introduced into the image due to inconsistencies in the focal length of the lens over its diameter, thus the purpose of this step is to make the projected image coplanar with the reference camera image.

Finally, we project the point $\mathbf{p}^L$ in the LiDAR coordinate system to the point $\mathbf{p}^I = [x^C, y^C, z^C]^T$ on the image plane of the i-th camera by

$$\mathbf{p}^C = \mathbf{P}^{(i)}_{rect}\ \mathbf{R}^{(0)}_{rect}\ {}^C_L\mathbf{T}\ \mathbf{p}^L \tag{2.27}$$

where $\mathbf{P}^{(i)}_{rect}$ implements the projection of a point in the coordinate system of the reference camera (camera 0) onto the image plane of the i-th camera.

Normalizing the left side of Equation 2.27, and denote $\mathbf{p}^I = [\frac{x^C}{z^C}, \frac{y^C}{z^C}, 1]^T$, we have

$$z^C\mathbf{p}^I = \mathbf{P}^{(i)}_{rect}\ \mathbf{R}^{(0)}_{rect}\ {}^C_L\mathbf{T}\ \mathbf{p}^L \tag{2.28}$$

Considering that we use image data from the left color camera (camera 2), the projection equation is then given by

$$z^C\mathbf{p}^I = \mathbf{P}^{(2)}_{rect}\ \mathbf{R}^{(0)}_{rect}\ {}^C_L\mathbf{T}\ \mathbf{p}^L \tag{2.29}$$

where $\mathbf{P}^{(2)}_{rect}$ and $\mathbf{R}^{(0)}_{rect}$ can be viewed as equal as the camera intrinsic $\mathbf{K}$, ${}^C_L\mathbf{T}$ is a transformation matrix composed of extrinsic parameters of the system.

2.5 Problem Statement

In Section 2.3, we discussed the projection of a point $\mathbf{p}^L$ in the LiDAR coordinate system to the camera image plane. We now consider that in a pair of an image and a LiDAR point cloud, there is a number of LiDAR points whose coordinate vectors are stacked horizontally into a point set matrix for the certain frame, denoted as $\mathbf{P}^L$. Following Equation 2.29, for each point $\mathbf{p}^L_i$ in a certain frame, the projection equation is given by

$$\mathbf{p}^C_i = \begin{bmatrix} x^C_i \\ y^C_i \\ z^C_i \end{bmatrix} = \mathbf{K}^C_L\mathbf{T}\mathbf{p}^L_i = \mathbf{K}^C_L\mathbf{T}\begin{bmatrix} x^L_i \\ y^L_i \\ z^L_i \\ 1 \end{bmatrix},\ \mathbf{p}^I_i = \begin{bmatrix} \frac{x^C_i}{z^C_i} \\ \frac{y^C_i}{z^C_i} \\ \frac{y^C_i}{z^C_i} \end{bmatrix} \tag{2.30}$$

where the subscript of $\mathbf{p}^L_i$ and $\mathbf{p}^C_i$ represents the i-th LiDAR point of a certain frame and the corresponding projected point, respectively.

Then we apply Equation 2.30 on the point set matrix $\mathbf{P}^L$, we have

$$\mathbf{P}^C = \begin{bmatrix} x^C_0 x^C_1 \cdots x^C_n \\ y^C_0 y^C_1 \cdots y^C_n \\ z^C_0 z^C_1 \cdots z^C_n \end{bmatrix} = \mathbf{K}^C_L\mathbf{T}\mathbf{P}^L = \mathbf{K}^C_L\mathbf{T}\begin{bmatrix} x^L_0 x^L_1 \cdots x^L_n \\ y^L_0 y^L_1 \cdots y^L_n \\ z^L_0 z^L_1 \cdots z^L_n \\ 1\ 1\ \cdots\ 1 \end{bmatrix},\ \mathbf{P}^I = \begin{bmatrix} \frac{x^C_0}{z^C_0} \frac{x^C_1}{z^C_1} \cdots \frac{x^C_n}{z^C_n} \\ \frac{y^C_0}{z^C_0} \frac{y^C_1}{z^C_1} \cdots \frac{y^C_n}{z^C_n} \end{bmatrix} \tag{2.31}$$

If we have N frames, then for a certain frame, Equation 2.31 becomes

$$\mathbf{P}^C_k = \begin{bmatrix} x^C_{k,0} x^C_{k,1} \cdots x^C_{k,n} \\ y^C_{k,0} y^C_{k,1} \cdots y^C_{k,n} \\ z^C_{k,0} z^C_{k,1} \cdots z^C_{k,n} \end{bmatrix} = \mathbf{K}^C_L\mathbf{T}\mathbf{P}^L_k = \mathbf{K}^C_L\mathbf{T}\begin{bmatrix} x^L_{k,0} x^L_{k,1} \cdots x^L_{k,n} \\ y^L_{k,0} y^L_{k,1} \cdots y^L_{k,n} \\ z^L_{k,0} z^L_{k,1} \cdots z^L_{k,n} \\ 1\ 1\ \cdots\ 1 \end{bmatrix},\ \mathbf{P}^I_k = \begin{bmatrix} \frac{x^C_{k,0}}{z^C_{k,0}} \frac{x^C_{k,1}}{z^C_{k,1}} \cdots \frac{x^C_{k,n}}{z^C_{k,n}} \\ \frac{y^C_{k,0}}{z^C_{k,0}} \frac{y^C_{k,1}}{z^C_{k,1}} \cdots \frac{y^C_{k,n}}{z^C_{k,n}} \end{bmatrix} \tag{2.32}$$

For convenience of expression, we rewrite Equation 2.24 as

$$ {}^{C}_{L}\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \tag{2.33} $$

where $\mathbf{t} = [t_x, t_y, t_z]^T$ is the translation vector, and $\mathbf{R}$ is the rotation matrix. The 3×3 rotation matrix can represent a unique rotation, but there are redundant degrees of freedom, which can be represented by three-dimensional Euler angles through certain linear transformation $\mathbf{R} = f(\phi_x, \phi_y, \phi_z)$. For the sake of clarity, we define the extrinsic parameters $\boldsymbol{\theta} = (\phi_x, \phi_y, \phi_z, t_x, t_y, t_z)$, ${}^{C}_{L}\mathbf{T}$ is only related to $\boldsymbol{\theta}$ with $\mathbf{K}$ known, thus we can rewrite Equation 2.30 as

$$ \mathbf{p}^{C}_{i} = T(\boldsymbol{\theta})\mathbf{p}^{L}_{i} \tag{2.34} $$

where $T(\cdot)$ is a function that uniquely parameterized by $\boldsymbol{\theta}$.

The proposed framework is dedicated to sensory setups with global-shutter cameras and 3D LiDARs that are rigidly attached. We assume that the camera is internally calibrated and these parameters are fixed during the calibration. The extrinsic calibration can be determined by maximizing the registration function between the camera and LiDAR:

$$ \hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} \mathcal{F}(\mathbf{P}^{C}_{k}, \mathbf{P}^{L}_{k} | \boldsymbol{\theta}) \tag{2.35} $$

which is equivalent to

$$ \{\hat{{}^{C}_{L}\mathbf{T}}\} = \arg\max_{{}^{C}_{L}\mathbf{T}} \mathcal{F}(\mathbf{P}^{C}_{k}, \mathbf{P}^{L}_{k} | {}^{C}_{L}\mathbf{T}) \tag{2.36} $$

According to Equation 2.24 and the properties of the rotation matrix, we can rewrite Equation 2.36 as follows

$$ \{\hat{\mathbf{R}}, \hat{\mathbf{t}}\} = \arg\max_{\mathbf{R},\mathbf{t}} \mathcal{F}(\mathbf{P}^{C}_{k}, \mathbf{P}^{L}_{k} | \mathbf{R}, \mathbf{t}), \ s.t.\ \mathbf{R}^T\mathbf{R} = \mathbf{U} \tag{2.37} $$

where $\mathbf{U}$ is the identity matrix, and the constraint is imposed by the property of the rotation matrix we introduced in Section 2.3.3.

2.6 Summary

In this section, we present in detail the physical model of the camera-LiDAR system and the framework of the calibration problem. As a prerequisite for the extrinsic calibration problem, the LiDAR-to-camera image transformation undergoes two steps, one is the 3D-2D imaging process from the camera system to the image plane, which is involved by the camera intrinsic, and the other is the 3D-3D transformation of the LiDAR coordinate system to the camera coordinate system, i.e., the extrinsic parameters that we estimate. The transformation or projection from the 3D to the 2D coordinate system is described by a combination of rotation and translation. Due to the special properties of the rotation matrix, we use the rotation matrix to describe the rotation in the projection of a 3D point to a 2D plane. The great advantage of the rotation matrix is its unique mathematical representation, however, the rotation matrix has nine entries,

while describing rotations in space requires only a minimum of three variables, the Euler angles. Given that rotation matrices and rotation angles can be transformed into each other, the calibration of the camera-LiDAR extrinsic calibration is a 6-DoF problem, and our goal is to find the optimal parameters to make the correct correspondence between the two types of data (images and point clouds).

3 Proposed Methods

Mis-calibration of the camera coordinate system and the LiDAR coordinate system will result in unachievable cooperative perception for autonomous driving. As we discussed in Chapter 1, a more efficient and automatic target-less epistemic calibration method is our goal, past edge-feature-based camera-LiDAR extrinsic calibration methods mostly ignore semantic features. We bring a new improvement that can be achieved by combining traditional edge extraction and semantic edge extraction.

The foundational mathematical model for the transformation of coordinates between camera and LiDAR systems was outlined in the preceding chapter. Building upon this, this chapter introduces our proposed algorithm for extrinsic calibration of camera-LiDAR systems, which is centered around the principle of edge alignment.

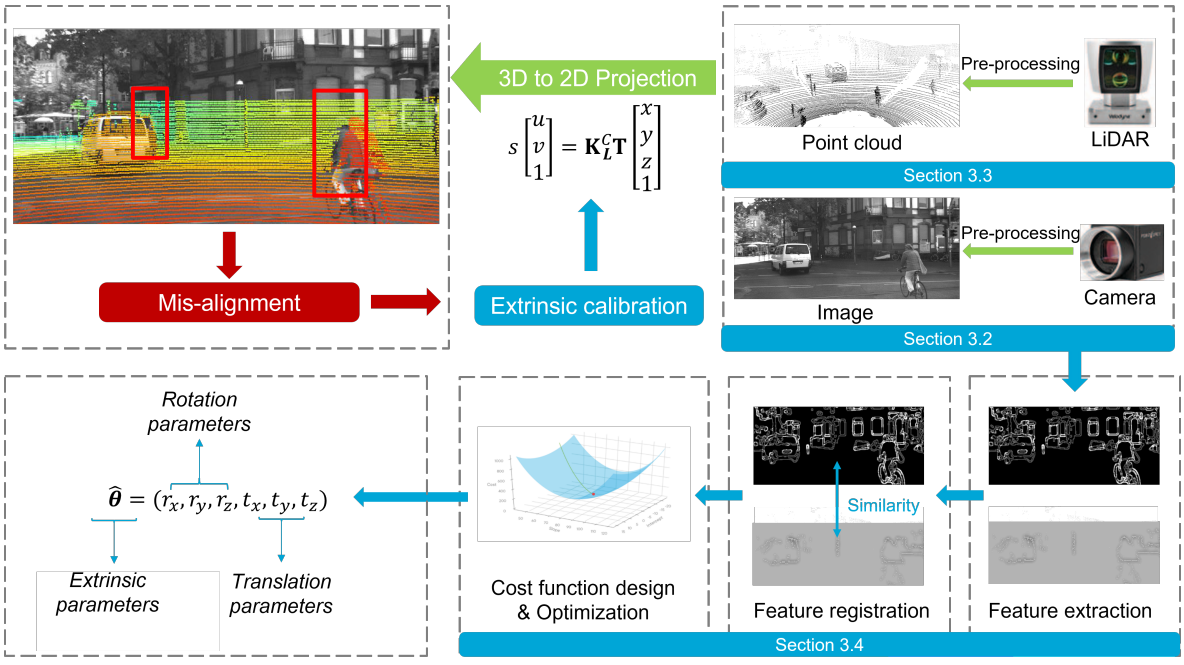


Figure 3.1: A Camera-LiDAR calibration framework.

Figure 3.1 explains a general framework of extrinsic parameter estimation, including the processing of LiDAR point cloud and camera image, feature extraction, feature registration and parameter optimization. In this chapter, we will introduce the processing techniques of images and point clouds, as well as the technical details of edge extraction and extrinsic parameters estimation.

3.1 One-shot Estimation

In this section, the one-shot estimation method is introduced. Figure 3.2 shows the simplified pipeline.

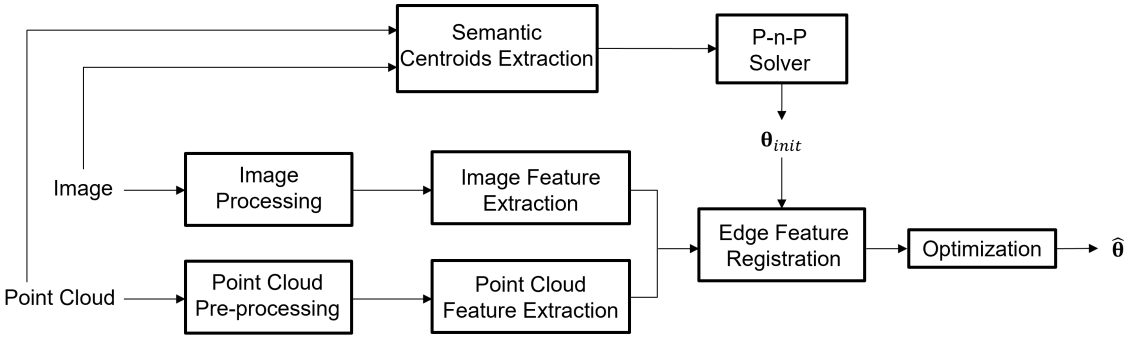


Figure 3.2: One-shot estimation pipeline.

For the input images-points pairs, we perform image processing on the images to provide high-quality images for subsequent image edge extraction. It is tricky for raw point clouds to perform edge extraction directly, so we need to preprocess the point clouds and mine the hidden edge information based on different attributes of the point clouds. In the edge extraction part, we carry out edge detection in the 2D plane and design the cost function for optimization. For the initial setting of extrinsic parameters, we use semantic networks to extract the semantic center of mass and perform 3D-2D relational alignment to achieve coarse calibration and use this result as the initial input.

3.2 Image Processing

Image quality and image edge extraction are affected by several factors, such as noise, contrast, brightness and exposure. Noise introduces errors in edge extraction near noisy points, compromising accuracy. Additionally, edges may be influenced by shadows or highlights caused by lighting. Overexposure impacts image contrast, and contrast variations can lead to insufficiently visible edges or incorrect identification of non-edges, all of which can make edge extraction challenging. Therefore, before we extract the edge of the image, exploiting image equalization and other image processing techniques can help improve edge extraction results. Here we will explore various image processing techniques designed to improve image quality, ultimately leading to enhanced edge extraction.

Histogram Equalization (HE) is the most common image enhancement technique used to improve the contrast and visibility of images. The first step is to compute the histogram $H(i)$ of the grayscale levels in the image:

$$H(i) = \sum_{x=1}^{M} \sum_{y=1}^{N} \delta(I(x,y), i) \tag{3.1}$$

where $M \times N$ is the size of the image (with M rows and N columns), $I(x, y)$ is the intensity of the pixel at position (x, y), and $\delta(a, b)$ is the Kronecker delta function, which is 1 if $a = b$ and 0 otherwise.

The Cumulative Distribution Function (CDF) of the histogram is calculated as follows

$$CDF(i) = \sum_{j=0}^{i} H(j) \tag{3.2}$$

where i ranges from 0 to 255 in an 8-bit image.

The CDF is then normalized by the total number of pixels N in the image

$$CDF_{norm}(i) = \frac{CDF(i)}{N} \tag{3.3}$$

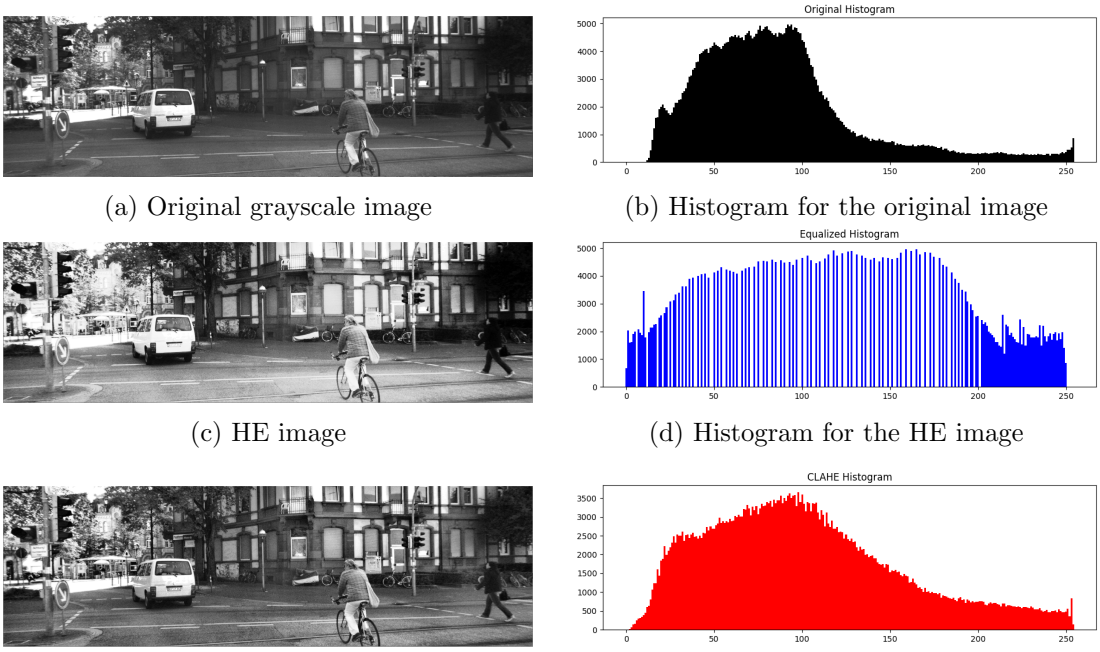
Mapping the pixel intensities of the original image to new intensity levels is done through Equation 3.4, which results in a more uniform distribution of intensity levels. This mapping is done so that the histogram of the output image is as close to a uniform distribution as possible, thus enhancing the overall contrast of the image.

$$i' = round((L - 1) \cdot CDF_{norm}(i)) \tag{3.4}$$

where each pixel intensity i in the original image is assigned to a new intensity i'. L is the number of possible intensity levels, $L = 256$ for an 8-bit image.

In the histogram equalization introduced earlier, the global image is directly equalized, which is global histogram equalization, without taking into account the local image area (local region). However, adaptive histogram equalization (AHE) allows the construction of a mapping function using only the histogram distribution within a window of local regions. AHE calculates the CDF for each local region and normalizes it as in HE. The intensity of each color block is mapped using the respective normalized CDF, which can enhance the local contrast.

Contrast Limited Adaptive Histogram Equalization (CLAHE) further optimizes AHE. In CLAHE, each local histogram is clipped at a predefined threshold before the CDF is computed. This threshold is usually a percentage of the maximum value of the histogram. The excess counts in the clipped histogram are then redistributed evenly across all intensity levels.



(a) Original grayscale image (b) Histogram for the original image

(c) HE image (d) Histogram for the HE image

(e) CLAHE image (f) Histogram for the CLAHE image

Figure 3.3: Comparison of different image equalization methods on a grayscale image.

We perform HE and CLAHE on the original gray-scale image, and the comparison on their histogram is shown in Figure 3.3. We can find that both equalization methods change the contrast of the image, and HE makes the grayscale distribution uniform globally. We magnify the local details, as depicted in Figure 3.4. It is evident that

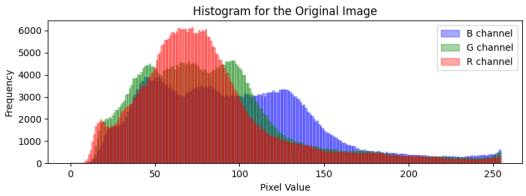
(a) (b) (c)

Figure 3.4: Comparison of different image equalization methods on local areas of the grayscale image

CLAHE exhibits better equalization performance compared to HE when processing local details.

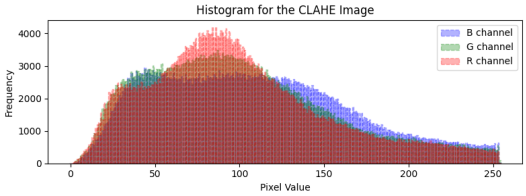
The previous results are based on grayscale image processing. If we perform CLAHE on the color RGB image, we need to perform CLAHE on the grayscale images of the three color channels respectively. The results are shown in the Figure 3.5. Noise filtering in image processing is essential for enhancing image quality and preparing images for

(a) Original RGB image



(b) Histogram for the original RGB image

(c) CLAHE image



(d) Histogram for the CLAHE image

Figure 3.5: Comparison between CLAHE on the RGB image and the original RGB image.

further processing like edge detection or object recognition. Two popular methods for noise filtering are Gaussian filtering and Bilateral Filtering.

Bilateral filtering is a filtering technique used for image denoising. It preserves the edge information of the image while reducing the effect of noise.

$$I'(x,y) = \frac{1}{W} \sum_{(i,j)\in\text{neighborhood}} I(i,j) \cdot w(i,j,x,y) \tag{3.5}$$

where $I'(x,y)$ is the pixel value in the output image. W is the normalized sum of weights. $I(i,j)$ represents the coordinates of pixels in the neighborhood. $w(i,j,x,y)$ is the pixel intensity weight function, considering both intensity similarity and spatial distance.

Gaussian filtering uses a Gaussian function (normal distribution) as a filter kernel to perform a convolution operation on an image. Its purpose is to smooth the image and reduce the noise in the image while blurring the details of the image.

$$H(x,y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \tag{3.6}$$

$$I'(x,y) = \sum_i \sum_j I(x-i, y-j) \cdot H(i,j) \tag{3.7}$$

where $H(x,y)$ is a 2D Gaussian filter kernel whose size is controlled by sigma. $I(x,y)$ is the pixel value of the original image and $I'(x,y)$ is the pixel value of the filtered image. The Gaussian filter kernel performs a convolution operation with the image, that is, the filter kernel is slid over the image and a weighted average is applied to the pixel values at each position.

3.3 Point Cloud Pre-Processing

LiDAR data comprises multiple frames of raw point clouds, denoted as $\mathcal{L} = \{\mathbf{P}_k^L | k = 0, 1, \cdots, N\}$. In contrast to images, which only possess grayscale values for each point, each point in $\mathbf{P}_k^L$ is characterized by multiple 'values' or attributes. Unlike traditional imaging where pixel values represent intensity, LiDAR points convey richer information about the environment. The point cloud data obtained through LiDAR scanning not only provides the three-dimensional coordinates (x, y, z) of a point in 3D space, but also provides the reflection intensity value r of the point, which we have already mentioned in Section 3.3.

One important index is depth information, denoted as d representing the distance from the LiDAR sensor to a point in the environment, which can be calculated by

$$d = \sqrt{(x^L)^2 + (y^L)^2} \tag{3.8}$$

which is the 2D Euclidean distance of a point in the LiDAR coordinate system.

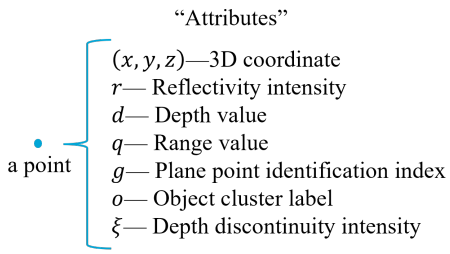


Figure 3.6: Point attributes of a LiDAR point

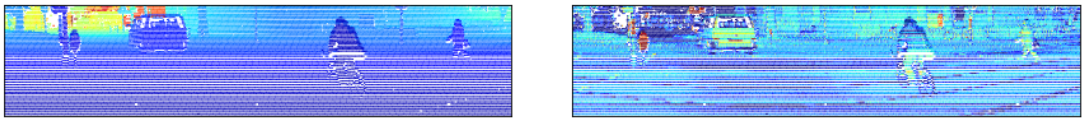
In addition to these basic attributes, some 'hidden' attributes can be calculated by different methods depending on the configuration of the LiDAR system and the specific characteristics of the acquired data. Figure 3.6 shows the point attributes of a LiDAR point. For example, the RANSAC planar segmentation algorithm can divide points into planar or non-planar points, instructed by g, the DBSCAN clustering algorithm can derive the class centroid for each point, denoted as o.

Point cloud preprocessing will take into account the characteristics of these attributes, threshold filtering, RANSAC plane segmentation and DBSCAN classification methods will be introduced in the following subsections.

3.3.1 Threshold Filtering

Threshold filtering stands as a straightforward and efficacious methodology for preprocessing point cloud data. Implementing a height threshold facilitates the exclusion of points below a designated elevation, which is particularly advantageous for the elimination of ground-level or low-lying objects from the dataset. Adjusting the threshold for reflectivity value enables the selective retention or removal of points based on their reflective intensity, a feature that is critical for distinguishing reflective objects such

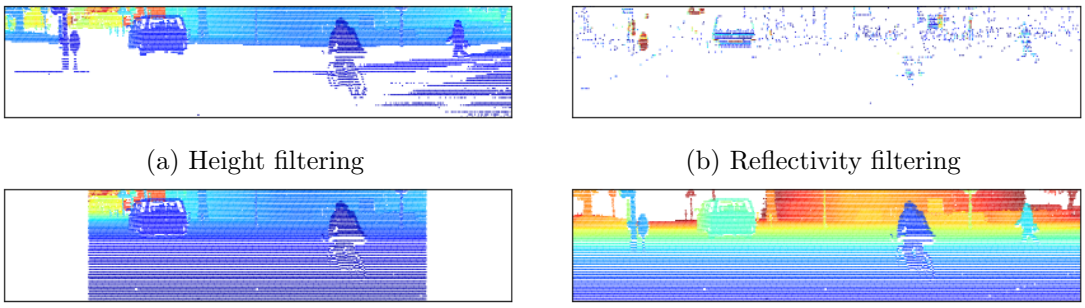
as traffic signals or automotive safety markings. Given the camera's limited field of view, it is efficient to excise LiDAR points that fall beyond this visual range, including areas that are not within the direct line of sight, like the rear angles obscured from the camera's perspective. Furthermore, setting a distance range threshold is imperative, as points situated too proximate or too distant may escape detection within the camera's imagery. Employing such a threshold aids in discarding points that lie outside the valid operational range, thereby ensuring that the processed data remains relevant and accurate for subsequent image analysis tasks. The number of point clouds processed through threshold filtering will be greatly reduced, which will help improve the speed of subsequent point cloud processing.



(a) un-processing depth projection map (b) un-processing reflectivity projection map

Figure 3.7: Un-processed LiDAR point cloud pano-view display

Figure 3.7a and 3.7b show the depth projection map and reflectivity projection map of all point clouds in one frame.



(a) Height filtering (b) Reflectivity filtering

(c) FoV filtering (d) Range filtering

Figure 3.8: Thereshoding filtering of LiDAR point cloud

Figure 3.8 demonstrates the effect of threshold filtering on different attributes. Height filtering can effectively remove part of the ground, and reflection value filtering can retain reflective objects such as signal signs and poles. Angle filtering filters horizontal angles of view beyond the camera's front view. Range filtering can filter out objects that are unclear (too close or too far) captured by the camera. These threshold filtering methods can be applied before or after the RANSAC and DBSCAN algorithms to reduce the amount of data calculation or clean the data again.

3.3.2 RANSAC Plane Segmentation

Point cloud data is large and sparse, and there is noise, which the noise can be removed by clustering the main objects, however, in the point cloud data, the ground points account for a large part of the data, which will greatly affect the efficiency of the clustering algorithm. When the ground points are projected onto the image plane, they also occupy a large area of the image. In Fig.3.7a and 3.7b, the projection maps based on the depth d and reflectivity intensity r of the points are drawn, respectively. In the reflectivity projection map, the ground lane lines can be differentiated by the discontinuity of reflectivity. In the depth projection map, there are a large number of redundant ground points, and there are no effective features to be extracted. This section describes the use of the RANSAC (Random Sample Consensus) algorithm to separate point cloud ground points and non-ground points.

The principle of the RANSAC (Random Sample Consensus) algorithm in extracting planes in a laser point cloud is based on iteratively selecting random samples to estimate a planar model and verifying how many points in the entire dataset match this estimated model. Algorithm 1 explains the basic steps and principles of the RANSAC algorithm for extracting planes from a laser point cloud.

Algorithm 1 RANSAC plane segmentation [42]

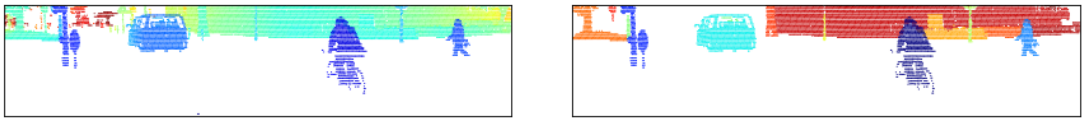
Input: Point cloud $\mathbf{P}^L \in \mathbb{R}^{4\times N}$ with N points, number of iterations $numIter$, initial number of inliers $numInliers$, threshold th

Create a all-zero vector $\mathbf{l}$ with length equal to $\mathbf{P}^L$
for $i < numIter$ **do**
  $maybeInliers$:= Randomly selected n points from $\mathbf{P}^L$
  $maybeModel$:= Model parameters $Ax + By + Cz + D = 0$ fitted to $maybeInliers$
  for $n < N$ **do**
    Calculate $dist_n = \frac{|Ax_n+By_n+Cz_n+D|}{\sqrt{A^2+B^2+C^2}}$ for each point
    if $dist_n < th$ **then**
      $\mathbf{l}_n \leftarrow 1$
    end if
  end for
  if $\sum_n \mathbf{l}_n > numInliers$ **then**
    Update model parameters A, B, C, D
    Update $numInliers := \sum_n \mathbf{l}_n$
  end if
end for

Output: Label sequence $\mathbf{l} \in \mathbb{R}^{1\times N}$

The algorithm randomly selects sample points (For planes, $n = 3$ points are usually selected). These points are used to define a potential plane model $Ax+By+Cz+D = 0$. Once the model is built, the algorithm checks how many points in the point cloud fit the flat model. That is, the distance $dist_n$ from each point to the plane is calculated and compared with a predetermined threshold. Points whose distance is smaller than the threshold are considered as inliers of the plane model. This process was repeated several

times and each time a new sample of points was randomly selected and a new model was built In all iterations, the model with the largest number of interior points was considered the best estimate. This model is used to separate ground and non-ground points in the point cloud.



(a) RANSAC FILTERING (b) DBSCAN CLUSTERING

Figure 3.9: Processed LiDAR point cloud pano-view map.

An example of the results of the RANSAC algorithm is shown in Figure3.9a, where the ground is filtered out in the depth map. Large areas of the ground are effectively removed while objects are retained. The strength of the RANSAC algorithm lies in its robustness to outliers. Even if the data contains a large number of points that do not fit the model (e.g. noise or other non-planar structures), RANSAC can efficiently identify the planes that best fit the dataset. This makes it very effective in processing real LiDAR data, especially in applications such as autonomous driving, robot navigation, and 3D reconstruction.

3.3.3 DBSCAN Object Clustering

Density-Based Spatial Clustering with Applications to Noise (DBSCAN) is a non-parametric algorithm for density-based clustering, which, given a set of points in a given space, groups together tightly-packed points (points with many nearby neighbors) and labels as outliers those points that are individually located in a low-density region (where their nearest neighbors are too far away.) DBSCAN is one of the most common and most frequently cited clustering algorithms [43].

DBSCAN requires two parameters: *eps* and the minimum number of points required to form a dense region $minPts$. It starts with an arbitrary starting point that has not been visited. This point's *eps*-neighborhood is retrieved, and if it contains sufficiently many points, a cluster is started. Otherwise, the point is labeled as noise. Note that this point might later be found in a sufficiently sized *eps*-environment of a different point and hence be made part of a cluster.

If a point is found to be a dense part of a cluster, its *eps*-neighborhood is also part of that cluster. Hence, all points that are found within the *eps*-neighborhood are added, as is their own *eps*-neighborhood when they are also dense. This process continues until the density-connected cluster is completely found. Then, a new unvisited point is retrieved and processed, leading to the discovery of a further cluster or noise. Through DBSCAN clustering on point cloud data, points with the same local density can be classified into the same category. The main purpose of object clustering is to separate objects from each other and find the boundaries of objects. The algorithm principle of DBSCAN clustering is shown in Algorithm 2.

Algorithm 2 DBSCAN object clustering [43]

Input: Point cloud $\mathbf{P}^L \in \mathbb{R}^{4\times N}$ with N points, minimum number of points to form a cluster $minPts$, cluster radius value eps

Create a cluster counter $C := 0$
Create a all-zero vector $\mathbf{L}$ with length equal to $\mathbf{P}^L$
Mark all points unvisited: $\mathbf{L} \leftarrow -1$
for each point $\mathbf{p}$ **in** $\mathbf{P}^L$ **do**
  if $\mathbf{L}_p! = -1$ **then**
    Neighbors set $\mathcal{N}$:= total number of points in the eps-range sphere given the center point $\mathbf{p}$
    if $\mathcal{N} < minPts$ **then**
      $\mathbf{L}_p := Noise$
    else if $\mathcal{N} > minPts$ **then**
      $C := C + 1$
      Update cluster label $\mathbf{L}_p \leftarrow C$
      Create a seed Set $\mathcal{S} := \mathcal{N} \setminus \{\mathbf{p}\}$
      for each point $\mathbf{q}$ in $\mathcal{S}$ **do**
        if $\mathbf{L}_q = noise$ **then**
          $\mathbf{L}_q \leftarrow C$
        else if $\mathbf{L}_q! = -1$ **then**
          $\mathbf{L}_q \leftarrow C$
        end if
        Neighbors set $\mathcal{N}$:= total number of points in the eps-range sphere given the center point $\mathbf{q}$
        if $\mathcal{N} > minPts$ **then**
          Add new neighbors to seed set $\mathcal{S} := \mathcal{S} \cup \mathcal{N}$
        end if
      end for
    end if
  end if
end for

Output: Cluster labels vector $\mathbf{L} \in \mathbb{R}^{1\times N}$

Figure 3.9b shows an example of the depth projection map after performing DBSCAN clustering. Compared with 3.9a, small objects in the distance are clustered out, while main object points are remained and clustered.

DBSCAN algorithm is sensitive to the amount of data. The time complexity of DBSCAN is usually $O(n \log n)$, where n is the number of data points. As the amount of data increases, the algorithm requires more time to calculate the neighborhood of each point, thus affecting the performance of the algorithm. In practice, using the RANSAC algorithm can separate ground points, and the amount of point cloud data will be greatly reduced, which can effectively improve the speed of the DBSCAN algorithm. In addition, since the edges that can be extracted from distant objects are too small or fuzzy, in the actual clustering process, we set a range threshold that filters out distant object points, and clusters only a limited range of objects.

3.4 Edge Extraction

In this section, we describe the process and important algorithms for edge extraction on images and edge extraction on point clouds. Firstly we will introduce the edge extraction operators and their principles, which we use for edge extraction in both images and point clouds. In the edge extraction part of the image, we introduce the advanced Segment Anything Model (SAM) semantic model to assist in extracting the image edges. Edge extraction for point clouds is more complex, the edges of a point cloud can be considered to be composed of two parts, 3D edge points generated by distance discontinuities and 2D edge extraction by projection map complementation, the latter being parameterized by extrinsic parameters. We apply a fast projection map complementation method to complement the projection maps with different features separately and fuse all the edges as the edge features of the point cloud.

3.4.1 Edge Operator

The Sobel operator and the Canny operator are two common edge extraction operations in image processing. The Sobel operator detects changes in the spatial brightness of an image by calculating the vertical gradient and horizontal gradient of the image pixel points, and regions of rapidly changing brightness, which usually correspond to edges. The 3×3 horizontal and vertical Sobel kernels are defined as

$$\begin{aligned} \mathbf{h}_x &= \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \\ \mathbf{h}_y &= \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \end{aligned} \tag{3.9}$$

where $\mathbf{h}_x$ and $\mathbf{h}_y$ represent the horizontal and vertical Sobel kernels, respectively. The Sobel kernel is applied to the surrounding neighborhood of each pixel of an image to calculate the gradient of that pixel in the horizontal and vertical directions. The Sobel edge detection algorithm is illustrated in Algorithm 3.

Algorithm 3 Sobel Operator

Input: Image $\mathbf{I}$

1: Calculate the horizontal gradient: $\mathbf{G}_x \leftarrow \mathbf{h}_x * \mathbf{I}$
2: Calculate the vertical gradient: $\mathbf{G}_y \leftarrow \mathbf{h}_y * \mathbf{I}$
3: Calculate an approximation of the gradient: $\mathbf{G} \leftarrow \sqrt{\mathbf{G}_x^2 + \mathbf{G}_y^2}$
4: Calculate the direction of the gradient: $\mathbf{\Theta} \leftarrow \arctan(\mathbf{G}_y, \mathbf{G}_x)$

Output: Edge map $\mathbf{G}$ (with same dimension as $\mathbf{I}$)

The Canny edge detector is a complicated and multi-step operator, employing techniques like Non-Maximum Suppression and Hysteresis Thresholding. The detailed procedures are explained in Algorithm 4. These methods are applied to further refine the

gradient of edges, effectively isolating the most probable edges with enhanced precision and accuracy.

Algorithm 4 Canny Operator

Input: Image $\mathbf{I}$, lower threshold $maxVal$, higher threshold $minVal$

Calculate the horizontal gradient: $\mathbf{G}_x \leftarrow \mathbf{h}_x * \mathbf{I}$
Calculate the vertical gradient: $\mathbf{G}_y \leftarrow \mathbf{h}_y * \mathbf{I}$
Calculate an approximation of the gradient: $\mathbf{G} \leftarrow \sqrt{\mathbf{G}_x^2 + \mathbf{G}_y^2}$
Calculate the direction of the gradient: $\mathbf{\Theta} \leftarrow \arctan(\mathbf{G}_y, \mathbf{G}_x)$
for each pixel (i, j) in $\mathbf{I}$ **do**
  if $\mathbf{G}(i, j) > \max \mathbf{G}_{neighbor_A}, \mathbf{G}_{neighbor_A}$ **then**
    $\mathbf{G}(i, j) \leftarrow \mathbf{G}(i, j)$
  else
    $\mathbf{G}(i, j) \leftarrow 0$
  end if
end for
for each pixel (i, j) in $\mathbf{I}$ **do**
  if $\mathbf{G}(i, j) > maxVal$ **then**
    $\mathbf{G}(i, j) \leftarrow 1$
  else if $\mathbf{G}(i, j) < minVal$ **then**
    $\mathbf{G}(i, j) \leftarrow 0$
  else
    if $\sum_{i-1}^{i+1} \sum_{j-1}^{j+1} \mathbf{G}(i, j) > 0$ **then**
      $\mathbf{G}(i, j) \leftarrow 1$
    else
      $\mathbf{G}(i, j) \leftarrow 0$
    end if
  end if
end for

Output: Edge map $\mathbf{G}$ (with same dimension as $\mathbf{I}$)

Non-maximum suppression evaluates each pixel to check if its gradient is the maximum when compared to the gradient of neighboring pixels along its gradient direction. This process ensures that only the most prominent gradient values are preserved, effectively highlighting the sharpest and most significant edges in the image.

Hysteresis threshold preserves strong edges and their connected edges through two thresholds and discards weak edge intensity gradients. Any edge greater than $maxVal$ is considered a strong edge, while edges below $minVal$ are defined as non-edges. Those lying between these two thresholds are classified as edge or non-edge based on their connectivity. If they are connected to "strong edge" pixels, they are considered part of the edge.

The Sobel operator is simple and fast to implement, with a small amount of calculation. Although the Canny edge detection algorithm is relatively complex, it is effective in suppressing noise and redefining "edge" and "non-edge". In our edge detection step, we

mix the two types of edges, superimposing the results of the Canny operator onto the Sobel operator can strengthen those "strong edges" while retaining the "weak edges". The edge detector is defined in Algorithm 5.

Algorithm 5 Edge Detector

Input: Image $\mathbf{I}$, edge operator mod

Calculate the Sobel edge: $\mathbf{G}_{Sobel} = Sobel\_Operator(\mathbf{I})$
Calculate the Canny edge: $\mathbf{G}_{Canny} = Canny\_Operator(\mathbf{I})$
if $mod = sobel$ **then**
    $\mathbf{G} \leftarrow \mathbf{G}_{Sobel}$
else if $mod = canny$ **then**
    $\mathbf{G} \leftarrow \mathbf{G}_{Canny}$
else if $mod = mix$ **then**
    $\mathbf{G} \leftarrow 0.5\mathbf{G}_{Sobel} + 0.5\mathbf{G}_{Canny}$
end if

Output: Edge map $\mathbf{G}$ (with same dimension as $\mathbf{I}$)

3.4.2 Image Edge Extraction

Camera data set $\mathcal{C}$ and LiDAR data set $\mathcal{L}$ representations are different. When two sensors partially share the same field of view, we can calibrate both sensors by extracting the edges of objects in the view and aligning the edges in the same view. Therefore feature extraction is required before aligning the two modalities. Camera data consists of multiple frames of images, $\mathcal{C} = \{\mathbf{I}_k | k = 0, 1, \cdots, N\}$, and $\mathbf{I}_k(i, j)$ is the pixel value of the image, where the subscript k represents the index of the image frame. The SMA pre-trained model is first applied to the entire image to get a new segmented image with several masks of differentiated objects, which are annotated as $\mathcal{C}_{SMA} = \{\mathbf{M}_k | k = 0, 1, \cdots, N\}$.

Figure 3.10 illustrates the procedure to obtain the edge intensity of the k-th frame image $\mathbf{E}_k^I$. The pixel value $\mathbf{E}_k^I(i, j)$ indicates the edge strength (intensity) of the corresponding image pixel.

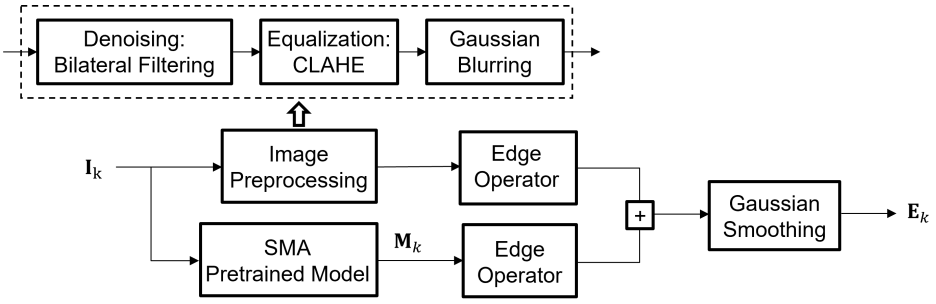


Figure 3.10: Image edge feature extraction pipeline.

We have introduced several image processing techniques in Section 3.2. Before edge extraction on the original image matrix $\mathbf{I}$, we first preprocess the image to improve

the image quality as shown by the steps inside the dotted box. Next, we perform edge extraction on the enhanced image and the segmented semantic mask, respectively. Before Gaussian smoothing, we blend two types of edge matrix. The significance of Gaussian smoothing is to make the distribution of intensity values of edges from 'strong edges' to 'non-edges' smoother.

(a) Image edge without Gaussian smoothing.

(b) Image edge with Gaussian smoothing.

Figure 3.11: Image edge w and w/o Gaussian smoothing.

Figure 3.11 shows the comparison between the extracted image edge and the smoothed result. The smoothed result reflects a more uniform appearance than the non-smoothed result.

3.4.3 Point Cloud Edge Extraction

The edge features of the point cloud can be exploited from its discontinuous depth as described in [32]. For a certain point $\{x_n^L, y_n^L, z_n^L\}$. Basically, the discontinuity intensity ξ can be calculated by

$$\xi_n = max(q_{n-1} - q_n, q_{n+1} - q_n, 0) \tag{3.10}$$

where $q = \sqrt{(x_n^L)^2 + (y_n^L)^2 + (z_n^L)^2}$ is the range value.

In each scan, LiDAR emits a set of laser beams vertically along a specific number of lines, scanning the surroundings. Significant differences in depth observed at the boundaries between objects or between an object and its background indicate a higher probability of being an edge. Using this property, neighboring points can be used to calculate the depth discontinuity values along the emitted beams and as a method of extracting edge strength.

Figure 3.12: Projection map of depth discontinuity.

The projection map depicting the extracted depth discontinuity is illustrated in Figure 3.12. In this representation, the value assigned to each pixel correlates with the intensity of the edge discontinuity. The larger the gray value (the brighter the pixel), the stronger the edge, and vice versa. For enhanced visual clarity, the contrast within the figure has been meticulously adjusted, ensuring that the contours of the edges are distinctly visible.

Another way is to extract edges from the projected density map, this approach is parameterized by $\boldsymbol{\theta}$. The density map $\boldsymbol{\Phi}$ can be obtained by performing a complex morphological algorithm [4] on the sparse projection map Ψ. The density completion problem of sparse projection images can be expressed as follows

$$\min \left\| \hat{f}(\boldsymbol{\Psi}) - f(\boldsymbol{\Psi}) \right\|_F^2 = 0 \tag{3.11}$$

where $\boldsymbol{\Psi}$ is the sparse projection map of the point cloud, $f(\boldsymbol{\Psi}) = \boldsymbol{\Phi}$ is the ideal density map. To find $\hat{f}$ that approximates the mapping from sparse projected map to dense map, realize $\hat{f}$ via a series of morphological operations described in Figure 3.13.

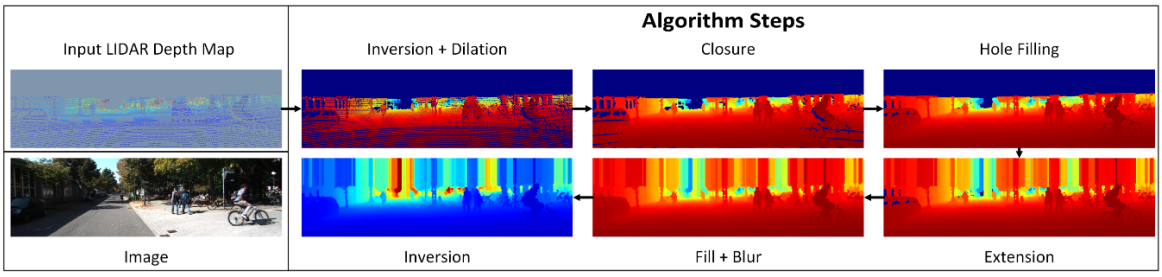


Figure 3.13: The flowchart of a fast method to complete sparse projection images. [4]

Taking the densification of depth maps as an example, in the first step we perform grayscale inversion. Applying a dilation operation on the original depth map will cause a larger distance to cover a smaller distance, resulting in the loss of edge information of closer objects. To solve this problem, the effective (non-null) pixel depth is linearly inverted. This inversion allows the algorithm to preserve closer edges when applying

the dilation operation. Next, we perform kernel dilation, filling empty pixels closest to valid pixels first, since these pixels are most likely to share close depth values with the valid depth. After the initial expansion step, there are still many holes with empty values in the projection map. We then complete small hole closure using morphological close operation. As can be seen from Figure 3.13, some small to medium-sized holes in the projection map were not filled by the first two dilation operations. To fill these small holes, an empty pixel mask is first computed, followed by a dilation operation. This operation only fills empty pixels while leaving the previously calculated valid pixels unchanged. The final fill step takes care of larger holes in the projection map that are not completely filled from the previous steps. Large holes are filled by masked dilations while leaving valid pixels unchanged.

After applying the previous steps, a blurring operation is performed on the resulting density map, aiming to remove outliers and noise while maintaining local edges. With all steps done, the final step is the inverse of the first step, depth Inversion, reverting from the inverted depth values back to the original depth encoding.

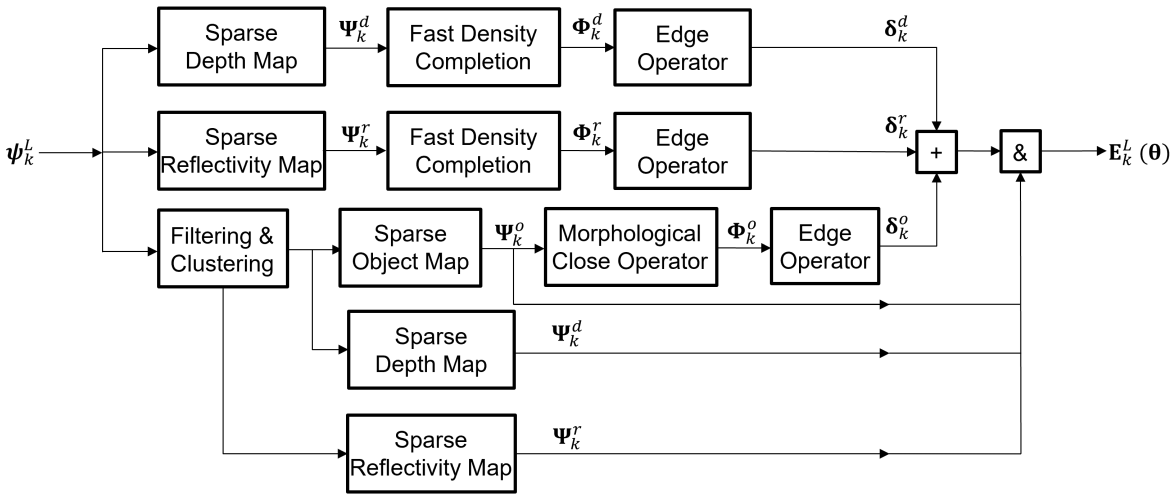


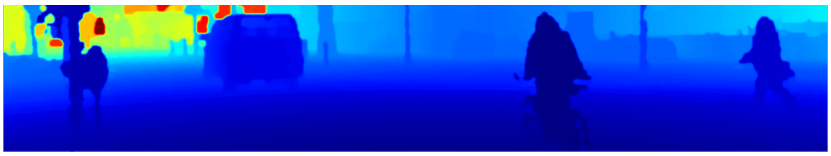
Figure 3.14: Point cloud edge extraction from density map.

Similarly, we use the same algorithm to complete the reflectivity sparse projection map. For the reflectivity density map, the boundaries between different low-reflective objects cannot be obtained from the difference in reflection intensity, so we filter out the low-reflectivity parts and thus highlight the boundary between highly reflective objects and low-reflective objects.

In Section 3.3.3 we introduced the method of using DBSCAN clustering. By projecting the filtered and clustered point cloud, we can obtain a sparse object projection map. For the object projection map, we only need to use the closing operation to fill the gaps between each cluster projection. The closing operation is effective in filling small holes and cracks within an object without significantly changing the size of the object.

In this way, the edge extraction of the 3D environment becomes the edge extraction of the 2D image. Figure 3.14 illustrates the edge extraction procedures from the dense

map. By performing density complementation on the depth sparse projection map and reflectivity sparse projection map, we can extract the edges $\boldsymbol{\delta}^d$ and $\boldsymbol{\delta}^r$ using the edge extraction operator, and then perform morphological closure operations on the filtered and clustered point cloud projections before edge extraction to obtain the edges $\boldsymbol{\delta}^o$. The edge matrices have the same dimensions as the camera image. The preservation of the real existing projected points can be achieved by performing the "with operation" between the obtained edge images and the points in the projection map. Finally, we have the point cloud edge matrix $\mathbf{E}^L(\boldsymbol{\theta})$ under a certain extrinsic parameter $\boldsymbol{\theta}$.



(a) Depth dense completion map.

(b) Reflectivity dense completion map.

(c) Object dense map.

Figure 3.15: Density completion map of different projection images.

In Figure 3.15, the depth map provides depth visual differentiation, the reflection map provides the boundaries between highly reflective objects and low reflective objects, and the object map enhances the edges of dense clusters. In this way, the edge extraction of the 3D environment becomes the edge extraction of the 2D image. For each projection map of given extrinsic parameters, the density map can be obtained through density completion for edge extraction, as shown in Figure 3.16.

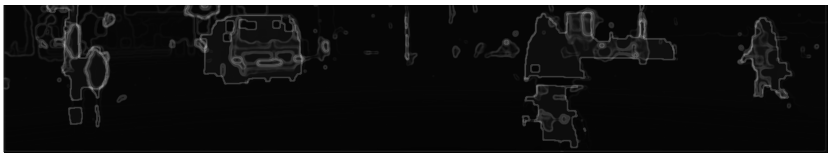


Figure 3.16: Extracted point cloud mixed attribute edge projection map.

3.4.4 Feature Registration and Optimization

When the point cloud is projected onto the image plane, set the value of each projected point pixel to the depth discontinuity, we get the depth discontinuity image denoted as $\mathbf{D}^{L\to I} = \mathbf{D}^{L}(\boldsymbol{\theta})$.

At the same time, depth and reflectivity can be used as weights on depth discontinuity image $\mathbf{D}^{L}(\boldsymbol{\theta})$. We linearly add up the normalized reflectivity and inverse depth $w^{L} = f(1/d^{L}, r^{L})$, then we can get the weight image $\mathbf{W}^{L}(\boldsymbol{\theta})$ by the same way we get $\mathbf{D}^{L}(\boldsymbol{\theta})$. The point of adding weights is that we place more trust in close 'edges' and highly reflective 'edges'.

Define the edge registration score (ERS) to evaluate the alignment score between the edge of the point cloud projected onto the image plane and the edge of the camera image:

$$J(\boldsymbol{\theta}) = \sum_{(i,j)} \{\mathbf{W}^{L}(\boldsymbol{\theta})\mathbf{D}^{L}(\boldsymbol{\theta})\mathbf{E}^{I} + \mathbf{E}^{L}(\boldsymbol{\theta})\mathbf{E}^{I}\} \tag{3.12}$$

which is the objective function $\mathcal{F}(\cdot)$ in equation (2.35) and (2.36). The larger the ERS, the more valid edges are aligned.

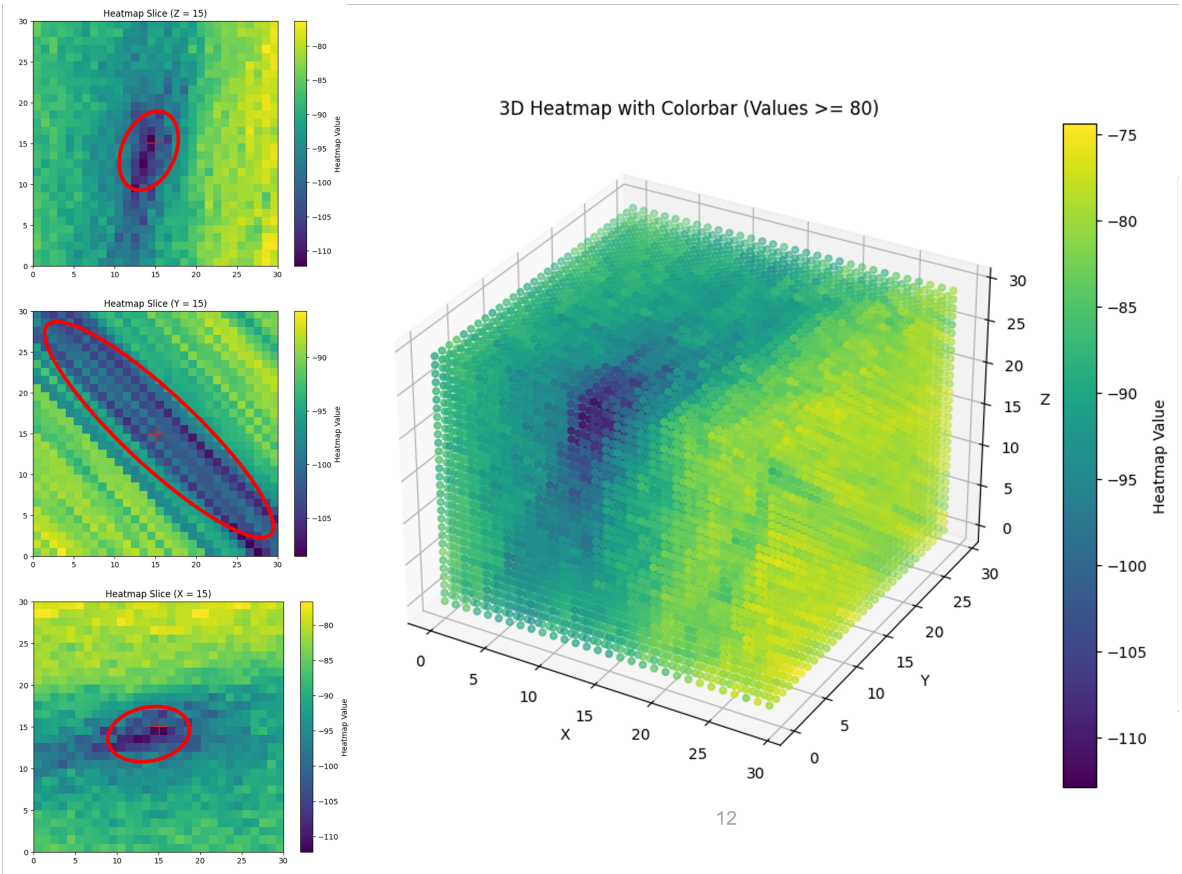


Figure 3.17: Cost function heatmap on varied rotation angles.

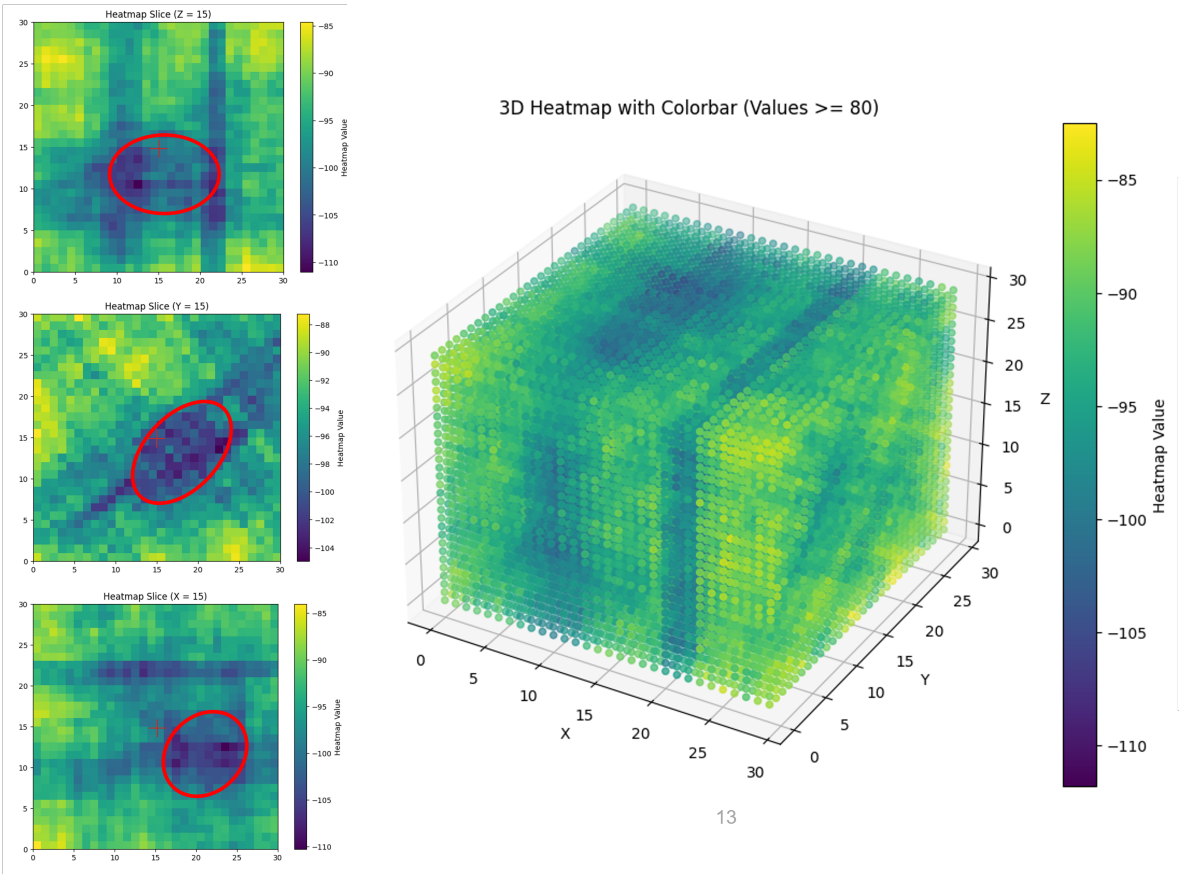


Figure 3.18: Cost function heatmap on varied translation vector.

A simple and easy-to-implement optimization method is the Barzilai and Borwein gradient ascent method [44]. Define the numerical gradient of the registration function:

$$G(\boldsymbol{\theta}) = \frac{J(\boldsymbol{\theta} + \boldsymbol{\epsilon}) - J(\boldsymbol{\theta} - \boldsymbol{\epsilon})}{2\boldsymbol{\epsilon}} \tag{3.13}$$

The update function is defined as:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha_k \frac{G(\boldsymbol{\theta}_k)}{||G(\boldsymbol{\theta}_k)||} \tag{3.14}$$

where the subscript k here denote the k-th iteration.

The Nelder-Mead method, also known as the simplex method, is an iterative numerical optimization algorithm used for unconstrained optimization problems. Proposed by John Nelder and Roger Mead in 1965 [45], this algorithm falls under the category of direct search methods, meaning it does not require derivative information (gradient or Hessian matrix) and relies solely on the evaluation of the objective function.

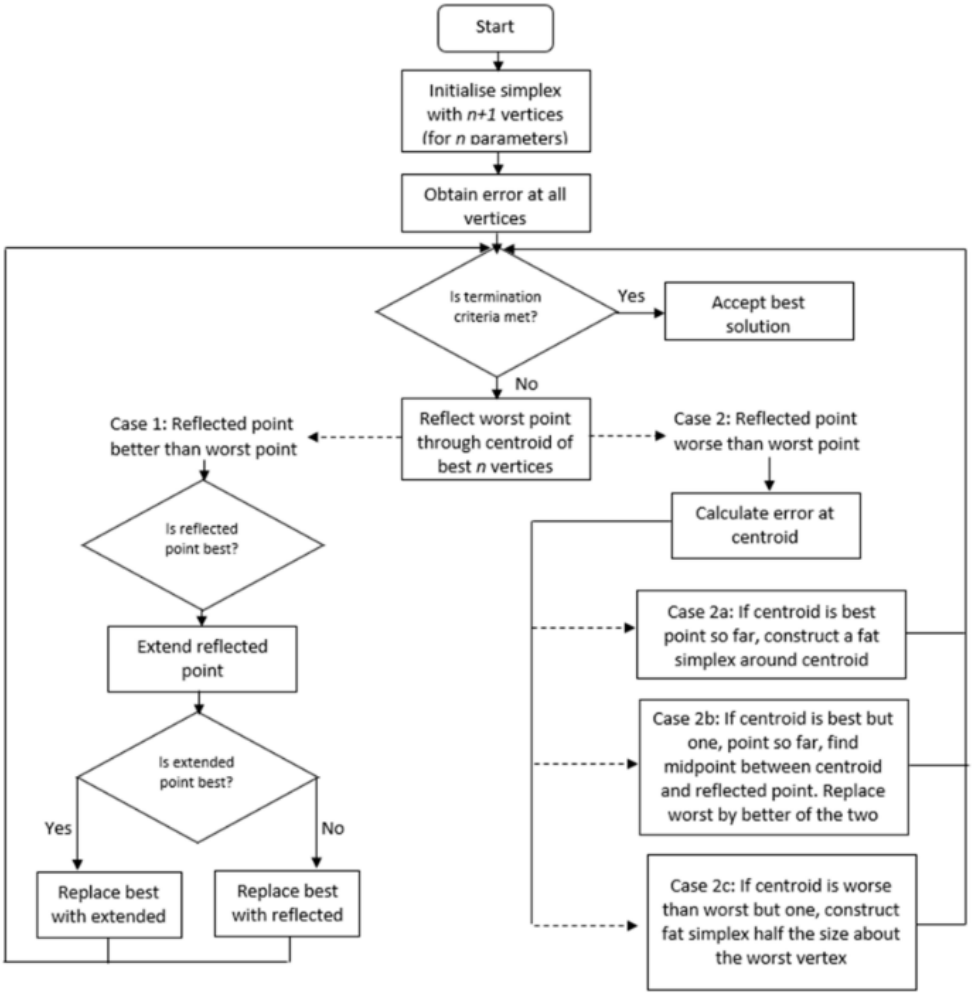


Figure 3.19: The flowchart of the Nelder-Mead optimization algorithm. [5]

Figure 3.19 illustrates the Flowchart for the Nelder-Mead optimization algorithm. The downhill simplex method starts with an initial simplex. For an N-dimensional optimization problem, an $(N+1)$-dimensional simplex is initially constructed, the function values of the vertices of the simplex are computed, and then the vertex function values are updated and new vertices and simplexes are constructed step by step until the convergence condition is reached.

The Nelder-Mead method is a derivative-free optimization method suitable for functions whose derivatives are difficult to calculate or do not exist, as in our case, while the Barzilai and Borwein method is more suitable for objectives that are smooth and whose derivatives are existing and easy to calculate. In this case, the Nelder-Mead method may be more stable when iterating irregular or noisy functions. The Barzilai and Borwein method may encounter difficulties when dealing with highly nonlinear or multimodal problems.

3.5 Initial Setting and Coarse Estimation

Previous research did not place significant emphasis on setting initial extrinsic parameters. Often, they relied on random initial estimates or configurations grounded in engineering experience. While this might not pose a significant challenge for engineers with industrial expertise, minimizing human intervention enhances the automation of the algorithm. Inspired by a calibration method based on semantic segmentation, a coarse estimation method is designed to obtain the initial value of extrinsic parameters [46].

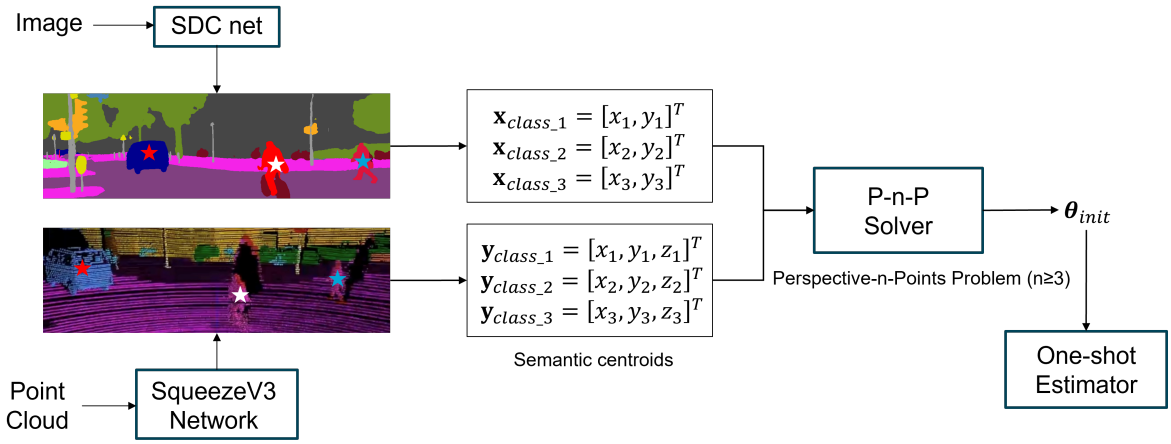


Figure 3.20: Coarse estimation framework.

Figure 3.20 illustrates the semantic-based coarse extrinsic estimation pipeline. To achieve semantic recognition in images, we employ the SDC Net along with the pre-trained model available in [47]. For semantic recognition in point clouds, we opt for the pre-trained SqueezeV3 network [48] to classify the semantic clusters.

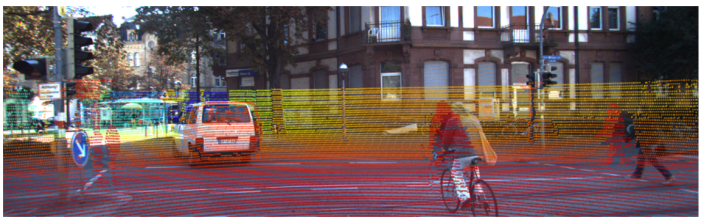


Figure 3.21: Coarse estimation projection map.

Although the projection shown in Figure 3.21 is obviously not the correct parameter, The coarse extrinsic parameter estimation method utilizing semantic segmentation centroid pairing, swiftly guides us to the proximity of the optimal value for fine-tuning.

3.6 Multi-frame Estimation Proposals

In Section 3.2, we discussed the issue of color histograms. Typically, histograms serve as a representation of the distribution of pixel intensity or color values within an image. The similarity ratio, derived from the comparison of two histograms, provides insight into the degree of similarity or dissimilarity in pixel intensity distributions between the two images. To illustrate this, we randomly selected three sets of images from a sequence of consecutive frames. Each set consists of five consecutive pictures, and their corresponding color histograms are depicted in Figure 3.22.

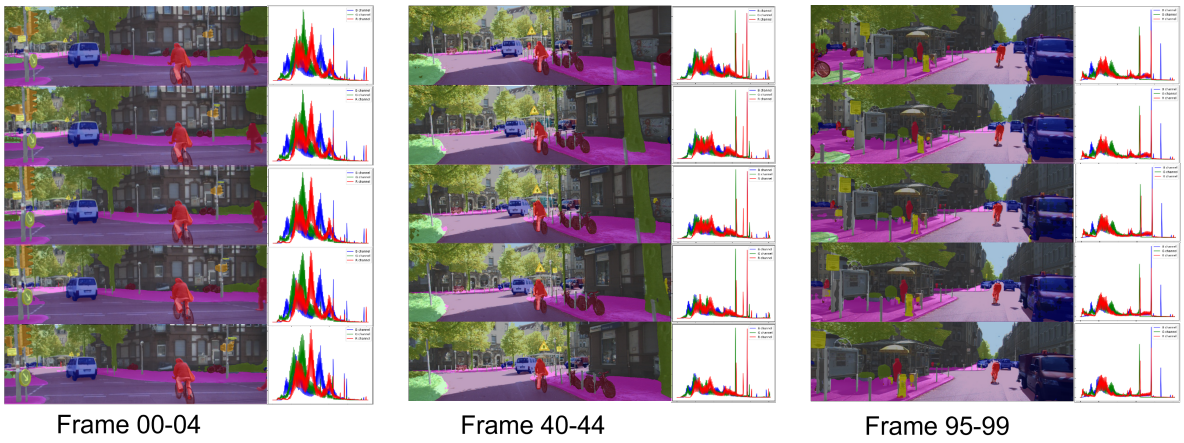


Figure 3.22: Histogram variation between frames.

We can see from the figure that the histograms of the images between consecutive frames are highly similar, while the histograms of the groups that do not pass through are more different. To measure this difference, define a metric to measure the histogram similarity ratio in Equation 3.15.

$$\rho = \sum_{k=b,g,r} \frac{\sum_i \min(H_k^1(i), H_k^2(i))}{\sum_i H_k^1(i) + \sum_i H_k^2(i) - \sum_i \min(H_k^1(i), H_k^2(i))} \tag{3.15}$$

where $H_k(\cdot)$ represent the histogram on different color channel (b, g, r). The numerator computes the intersection of two histograms. Denominator Computes the union of two histograms. In this formula, we perform similarity calculations on the histograms of each color channel separately, and then sum the results for all channels to get the overall similarity ratio.

Given the histogram, we can calculate the histogram similarity ratio following Algorithm 3.15. This similarity ratio ranges from 0 to 1, where 0 indicates no similarity (complete dissimilarity), while 1 indicates perfect similarity. Figure 3.23 draws the confusion matrix of calculated histogram similarity ratios between the chosen images set. In [32] and [7], both proposed the method of estimating the sliding average. However, this is a non-weighted average.

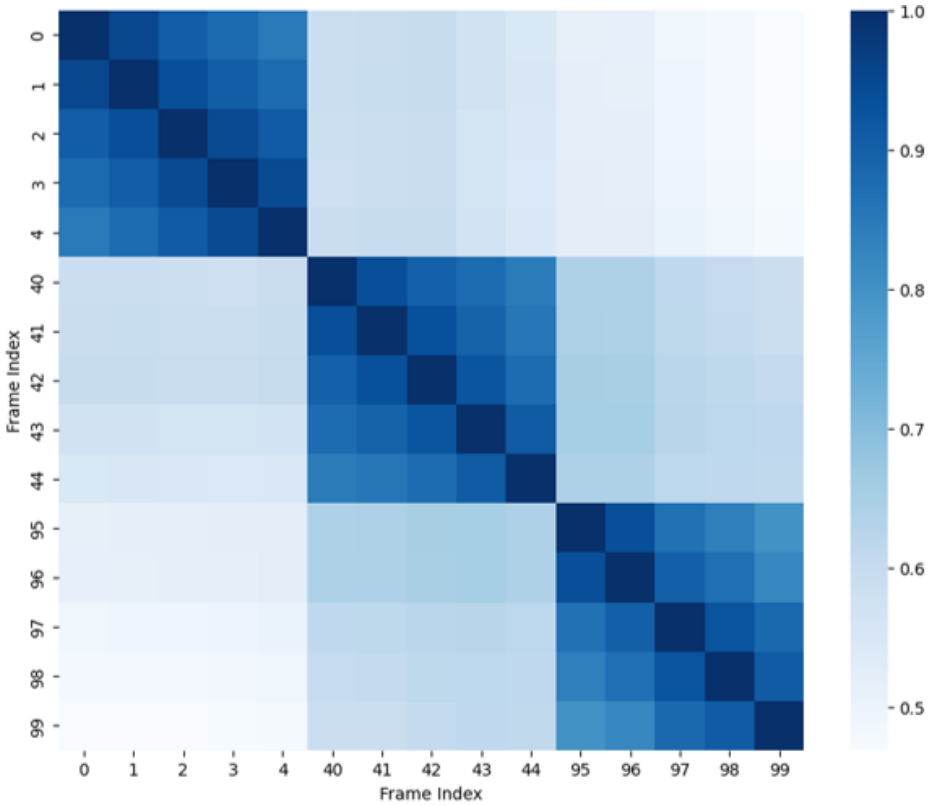


Figure 3.23: Histogram correlation between frames.

Using the feature of Histogram similarity, we believe that when the correlation is lower than 0.7, it means a change in the scene. Therefore, we propose weighted multi-frame averaging to improve the results of a single frame, as shown in Figure 3.24.

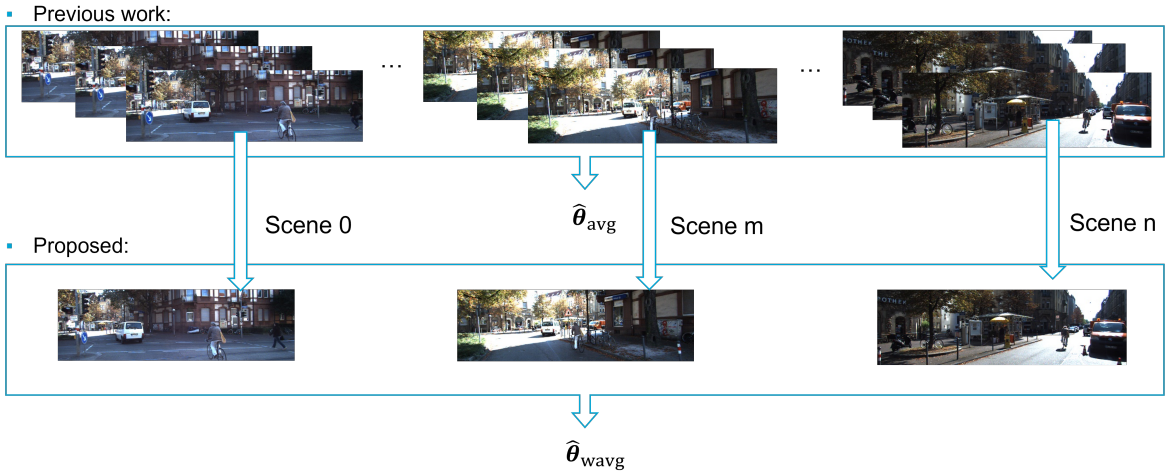


Figure 3.24: Multiple frames averaging flowchart

For consecutive image frames, we filter the representative scene frames by histogram

ratio with the following algorithm:

Algorithm 6 Scene selection with histogram similarity as threshold

Input: Image set $\{\mathcal{I}|\mathbf{I}_1, \mathbf{I}_2, \cdots\}$, threshold th

Histogram initial $\rho = 1.0$
Calculate the histogram h of the initial image $\mathbf{I}_1$
for each image $\mathbf{I}_{i+1}$ in $\mathcal{I}$ **do**
  Calculate the histogram h_{i+1} of the image $\mathbf{I}_{i+1}$
  Calculate the histogram similarity ρ between h and h_{i+1} according to (3.15)
  if $\rho < th$ **then**
    $\mathcal{S}$:=$\mathcal{S} \bigcup \mathbf{I}_{i+1}$
    $h \leftarrow h_{i+1}$
  end if
end for

Output: New image set $\{\mathcal{S}|\mathbf{I}_1, \mathbf{I}_k, \cdots\}$

Similar scenes have a similar number of features (number of edges), and filtering frames is to avoid the reuse of similar scenes in a set of consecutive frames. According to the selected scenes, we can assign the weight of each newly selected scene according to the ratio of the number of semantic pixels of a specific category. The more important categories, such as cars, pedestrians, traffic signs, and poles, the greater the proportion, we think This frame provides a more believable edge. Therefore, the weighted multi-frames average method can be expressed as

$$J'(\boldsymbol{\theta}) = \sum_{j \in S} w_j J_j(\boldsymbol{\theta}) \tag{3.16}$$

where S includes the index of selected scenes, the calculation of $J_j(\boldsymbol{\theta})$ follows the Equation 3.12 but normalized. w_j is the ratio of main semantic class pixels (vehicle, pedestrian, cyclist, pole, traffic) to the total effective semantic pixels.

3.7 Summary

In this chapter, we first present a detailed description of the edge extraction operator and its underlying operating principle, setting the stage for the subsequent processes. The process of edge extraction is bifurcated into two distinct segments: image edge extraction and point cloud edge extraction. Each segment employs specific techniques and methodologies tailored to the unique characteristics of the data format - one for the two-dimensional image data and the other for three-dimensional point cloud data. This step is critical in preparing the data for the subsequent alignment phase. To facilitate and refine the alignment process, we introduce the concept of edge similarity scores. These scores serve as a quantitative measure of how well the edges from the two datasets align with each other. By defining and computing these scores, we are able to iteratively optimize the parameters of the calibration algorithm, ensuring a high degree

of accuracy and efficiency in the alignment. In summary, this section elaborates on our innovative approach to camera-LiDAR extrinsic calibration, focusing on the techniques of edge extraction, alignment, and the utilization of edge similarity scores for parameter optimization.

4 Experiments and Results

This chapter focuses on experiments and evaluations of extrinsic parameter estimation using the KITTI dataset, a leading computer vision algorithm evaluation platform in autonomous driving. We will detail the data platform, its format, and file structure within the KITTI dataset. Experiments are conducted in both single-frame and multi-frame formats across three scenes from the KITTI dataset, with subsequent results and analysis.

4.1 KITTI Dataset

The KITTI collects 3D LiDAR point cloud information and 2D camera-captured image data, which provides multiple reliable benchmarks and evaluations for autonomous driving tasks, and many state-of-art multi-sensor fusion algorithms are evaluated based on KITTI databases [49], [50], [51]. In addition, the KITTI dataset provides data collected in multiple scenarios, therefore we select the KITTI dataset for our experiments and evaluations, and this section describes the sensor configurations and the data structure and composition of the KITTI dataset.

4.1.1 Sensor Configuration

The data is recorded using an eight-core i7 computer equipped with a RAID system, running Ubuntu Linux and a real-time database.

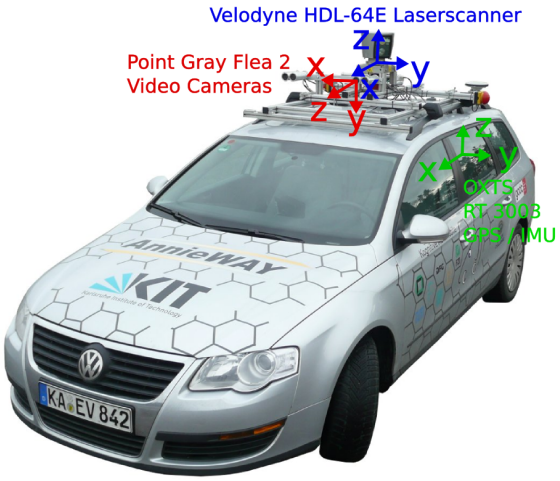


Figure 4.1: Configuration of the sensors mounted on the data recording vehicle [6]

The data collection vehicle is equipped with two color and two grayscale PointGrey Flea2 video cameras (10 Hz, resolution: 1392×512 pixels, opening: 90°×35°), a Velodyne HDL-64E 3D laser scanner (10 Hz, 64 laser beams, range: 100 m) and a GPS/IMU localization unit with RTK correction signals (open sky localization errors is smaller than 5 cm) [52]. The configuration of the camera, LiDAR, and GPS sensors equipped on the recording vehicle are shown in Figure 4.1. The model of the sensors is listed as follows:

- Inertial Navigation System (GPS/IMU): OXTS RT 3003
- Laserscanner: Velodyne HDL-64E
- Grayscale cameras, 1.4 Megapixels: Point Grey Flea 2 (FL2-14S3M-C)
- Color cameras, 1.4 Megapixels: Point Grey Flea 2 (FL2-14S3C-C)
- Varifocal lenses, 4-8 mm: Edmund Optics NT59-917

In order to facilitate the calibration of sensor data, the direction of the camera and LiDAR coordinate system are defined as follows:

- Camera: x = right, y = down, z = forward
- LiDAR: x = forward, y = left, z = up
- GPS/IMU: x = forward, y = left, z = up

Figure 4.2 shows the relative positions of the four stereo cameras, LiDAR, and GPS/IMU under the plane. The four cameras were numbered 0, 1, 2, and 3, where the first two were left and right grayscale cameras, and the last two were left and right colored cameras. Among these, camera 0 was used as a reference indicating the origin of the camera reference coordinate system. To produce binocular stereo images, cameras of the same type were mounted 54 cm apart.

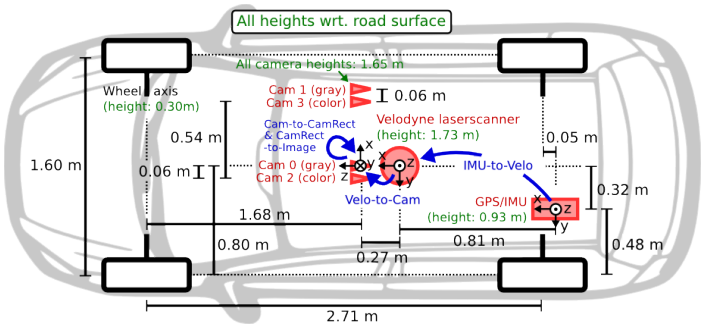


Figure 4.2: Sensor setup [6]

In our experiments, we used the 3D object detection benchmark, which consists of a total of 7481 training images and 7518 test images as well as the corresponding point clouds, comprising a total of 80,256 labeled objects.

4.1.2 Data Structure and Organization

In our experiments, the files involved and the data structure are explained in Figure 4.3. File `image_0x` contains the images captured by camera x. File `velodyne_points` stores the LiDAR point cloud data collected at the corresponding moment. File `calib_cam_to_cam.txt` and `calib_velo_to_cam.txt` record the camera intrinsic and extrinsic parameters respectively.

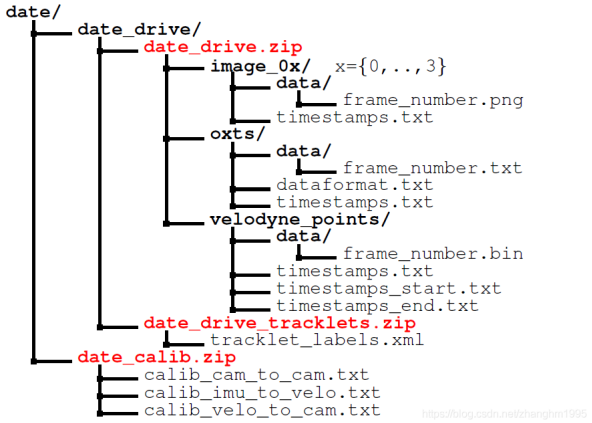


Figure 4.3: Structure of the KITTI raw data dataset

The color images captured by camera 2 are stored in 8-bit PNG format with a resolution of 1242×375, and the atlas is as follows

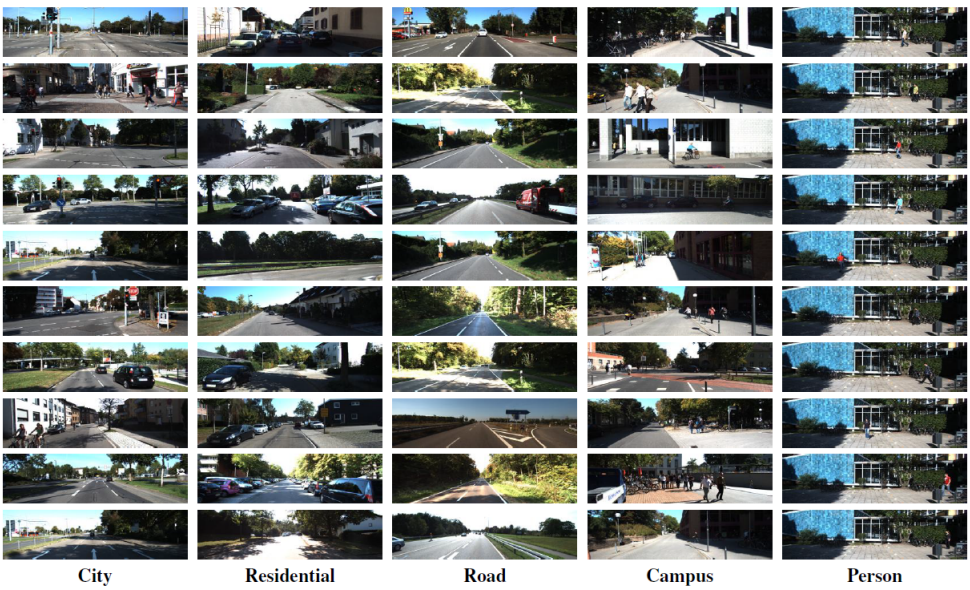


Figure 4.4: Example of KITTI image data [6]

There are 5 scenarios in the KITTI image dataset, which are 'City', 'Residential', 'Road' and 'Campus', and 'Person'.

The LiDAR point cloud data captured by Velodyne HDL-64E are stored in binary format, and each line in the point cloud file stores eight values, representing two points. The sensor intrinsic parameters are given in Table 4.1. The Velodyne HDL-64E achieves

Table 4.1: Velodyne HDL-64E important parameters.

Device parameters	**Value**
Number of Channels	64 lasers
Vertical Resolution	0.08 degree
Horizontal Resolution	0.35 degree
Horizontal Field of View (FOV)	360 degree (azimuth)
Vertical Field of View (FOV)	26.8 degree (elevation) (+2 up to −24.8 down)
Range	50 meters

360-degree environmental scanning by rotating its internal lens array. The entire device rotates around a vertical axis and 64 laser beams scan multiple horizontal planes simultaneously. Due to the characteristics of multiple laser beams and high-speed rotation, HDL-64E can generate higher-density point cloud data.

4.2 Results and Analysis

This section shows the experiments of the proposed method on three scenarios, city, residential area and road (highway) scenarios. We compare with the methods in Table 4.2. The evaluation of multi-frame averaging is given in the last section.

Table 4.2: Summary of 5 camera-LiDAR extrinsic calibration methods.

Method	**Type**	**LiDAR Attribute — Camera Attribute**	**Method Complexity**
Pandey [24]	MI	Reflectivity — Grayscale intensity	+
Levinson and Thrun [32]	Edge	Depth discontinuity — Grayscale intensity difference	++
Castorena [33]	Edge	Depth discontinuity — Grayscale intensity difference	+++
Zhang (MulFEAT) [7]	Edge	Depth & reflectivity discontinuity + Density distribution difference — Grayscale intensity difference	++++
Proposed	Edge	Depth & reflectivity discontinuity + Density distribution difference — Grayscale intensity difference + 2D Semantic information	++++

4.2.1 Comparison of Related Methods

In this section, we focus on comparing the following four classical and cutting-edge approaches, one mutual information-based approach [24], and three feature-based approaches [32], [33] and [7]. In Table 4.2 and 4.3, we summarize these five methods for camera-LiDAR extrinsic calibration.

Pandey's method maximizes the (mutual information) MI of the two attributes, reflectivity value $\{r_i; i = 1, 2, \cdots N\}$ of the points and grayscale intensity value $\{X_i; i = 1, 2, \cdots N\}$ of the corresponding image pixel upon which the 3D LiDAR point projects, as shown below:

$$J(\boldsymbol{\theta}) = MI(r, X; \boldsymbol{\theta}) \tag{4.1}$$

where $MI(\cdot)$ is the mutual information metric.

Levinson and Thrun maximize the summation of the depth discontinuities ξ times the "edginess" D of the image. Each pixel in the grayscale image is set to the largest absolute value of the difference between it and any of its 8 neighbors to obtain the edge image E. The "edginess" of each pixel D_{ij} is calculated by:

$$D_{i,j} = \alpha \cdot E_{i,j} + (1 - \alpha) \cdot \max_{x,y} E_{x,y} \cdot \gamma^{\max(|x-i|,|y-j|)} \tag{4.2}$$

Then iterate over all 3D points, the overall measure of this method is

$$J(\boldsymbol{\theta}) = \textstyle\sum_p \xi_p \cdot D_{i,j} \tag{4.3}$$

where (i, j) refers to the coordinates in image space onto which the 3D LiDAR point p projects.

Castorena uses the depth discontinuity ξ to construct a high-resolution depth map $\boldsymbol{\phi_\theta} \in \mathbb{R}^{H \times W}$, with the image $\mathbf{I} \in \mathbb{R}^{H \times W}$ providing two gradient variations in both the vertical and horizontal direction. Matching the edges of the depth map and intensity image, the objective function is shown below:

$$\mathcal{F}(\boldsymbol{\theta}) = \textstyle\sum_{k \in \{x,y\}} \dfrac{\mathcal{A}_k(\boldsymbol{\phi_\theta})}{\mathcal{N}_k(\boldsymbol{\phi_\theta})} \tag{4.4}$$

where the numerator and denominator are given by

$$\begin{aligned} \mathcal{A}_k(\boldsymbol{\phi_\theta}) &= \sum_{n \in \Omega_{\boldsymbol{\theta}}} w_{k,n} \left| \{\nabla_k \boldsymbol{\phi_\theta}\}_n \right| \\ \mathcal{N}_k(\boldsymbol{\phi_\theta}) &= \Big(\sum_{n \in \Omega_{\boldsymbol{\theta}}} w_{k,n}\Big)\Big(\sum_{n \in \Omega_{\boldsymbol{\theta}}} \left| \{\nabla_k \boldsymbol{\phi_\theta}\}_n \right|\Big) \end{aligned} \tag{4.5}$$

where ∇_k is the gradient operation on depth image $\boldsymbol{\phi_\theta}$ along the direction k (horizontal or vertical). The weight is imposed by the gradient change of image $w_{k,n} = \exp \gamma \left| \{\nabla_k \mathbf{I}\}_n \right|$. This equation is evaluated only at the point of the actual LiDAR measurements $\Omega_{\boldsymbol{\theta}}$. The denominator works as a normalization factor that eliminates the impact of differences in point numbers.

MulFEAT constructs a multi-feature density map and matches the edge intensity of the image, expressed as the sum of the edge strength multiplied by the probability of being identified as an edge in the point cloud.

$$J(\boldsymbol{\theta}) = \frac{N_m}{N_e} \sum_{n=1}^{N} P(\mathbf{E}_n^L) \cdot E_n^C \tag{4.6}$$

where $P(\mathbf{E}_n^L) = \frac{1}{3} \sum_{k \in \{d,r,o\}} \mathbf{E}_k(i_n, j_n)$ is the edge probability of a LiDAR point obtained from the multi-feature density map, E_n^C is the edge intensity of the corresponding pixel in the image where the 3D point projects.

Table 4.3: Objective forms of 5 methods.

Method	Objective	Optimization
Pandey [24]	$J(\boldsymbol{\theta}) = MI(r, X; \boldsymbol{\theta})$	Barzilai-Borwein steepest descent method
Levinson and Thrun [32]	$J(\boldsymbol{\theta}) = \sum_p^{all\ points} \xi_p \cdot D_{i,j}$	Grid search
Castorena [33]	$\mathcal{F}(\boldsymbol{\theta}) = \sum_{k \in \{x,y\}} \frac{\mathcal{A}_k(\phi_{\boldsymbol{\theta}})}{\mathcal{N}_k(\phi_{\boldsymbol{\theta}})}$	FISTA
Zhang (MulFEAT) [7]	$J(\boldsymbol{\theta}) = \frac{N_m}{N_e} \sum_{n=1}^{N} P(\mathbf{E}_n^L) \cdot E_n^C$	Barzilai-Borwein steepest descent method
Proposed	Equation (3.12)	Nelder-Mead downhill simplex method

The principle of Pandey's method is relatively easy to understand, but all LiDAR point data are used when calculating mutual information, which will bring a certain amount of calculation. The complexity of Castorena and MulFEAT methods mainly comes from the operations when completing sparse feature maps. Therefore, complex sparse feature map completion parameterized by extrinsic parameters is not suitable to appear in the cost function. In particular, MulFEAT does not project the point cloud to the image plane, but uses a cylinder model to perform non-parametric projection. The proposed method is far superior to these two methods in calculating sparse feature maps, and can achieve real-time calculations, so it can still be projected to the image plane for matching without the need for a virtual geometric model. Though Our method has high requirements on computing power in the semantic model, the introduction of semantics can solve the initial value problem and avoid random initial value settings.

4.2.2 Scenario 1: Urban Road

The first example is a classic urban road scene with a pedestrian, a vehicle, a cyclist, some traffic lights and signs, road poles, and urban buildings in the background. The statistical diagram of various semantic categories in the image is shown in Figure 4.5.

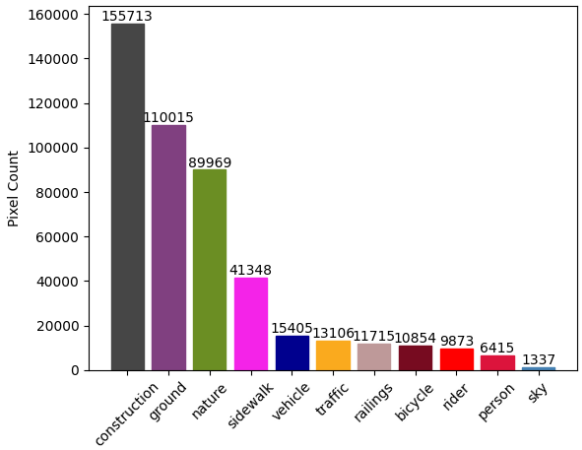


Figure 4.5: Histogram of pixel counts for semantic classes - Scenario 1: Urban road.

Figure 4.6 shows the calibrated image result. There is less greenery and therefore less noise in the image for edge extraction. Clustering performs better in this scene, there are multiple major objects that are completely separated from each other, which means we have more edge information available from the depth discontinuity between the objects and the background. Traffic lights and signs make the involvement of reflection intensity information provide more edge information.

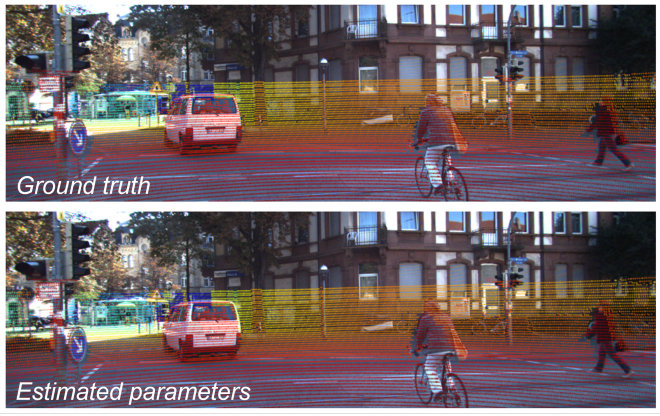


Figure 4.6: Qualitative error for one frame - Scenario 1: Urban road.

Figure 4.7 shows the loss curve on different methods. According to Bichi's paper, the gradient optimizer can get stuck because the method is fixing small displacements online rather than searching globally for optimal values. In our improved method, the initial value is estimated using semantic information, which allows a global search around the initial value. The loss profile of the proposed method looks a little sharper and is slightly better than Bichi's method overall.

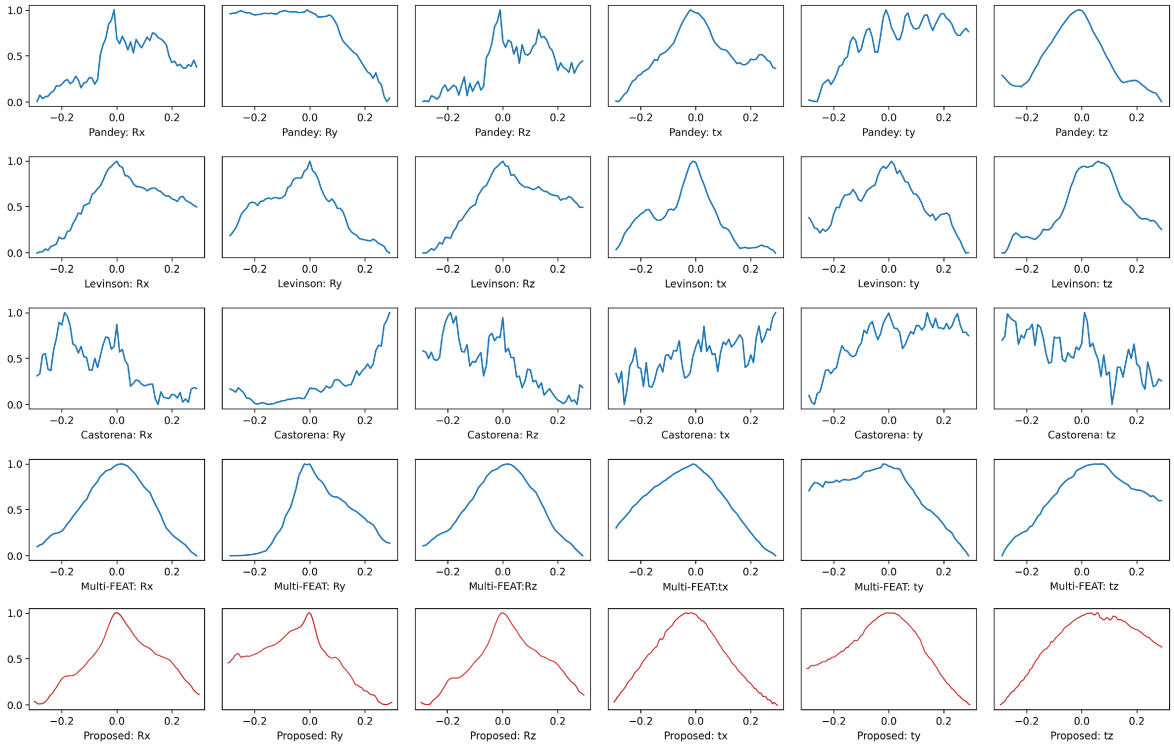


Figure 4.7: Loss curve in 6-axis - Scenario 1: Urban road.

4.2.3 Scenario 2: Neighborhood Alley

The scene of the second example is a narrow road in an urban residential area with parked vehicles on both sides of the road, no pedestrians or cyclists, and no greenery. The statistics of the various semantic categories in the image are shown in Figure 4.8, where construction and vehicles occupy the major pixels.

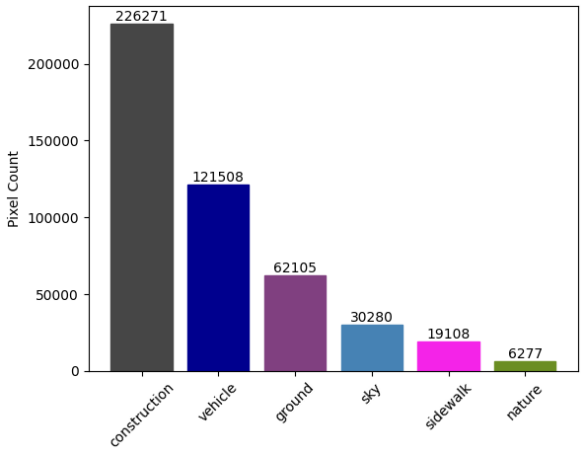


Figure 4.8: Histogram of pixel counts for semantic classes - Scenario 2: Neighborhood alley.

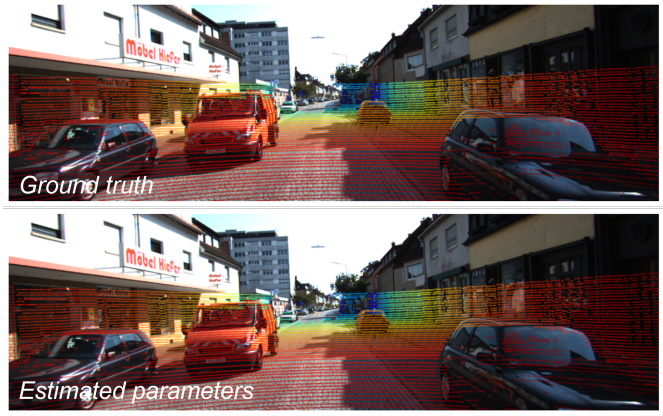


Figure 4.9: Qualitative error for one frame - Scenario 2: Neighborhood alley.

Figure 4.9 shows the calibrated image result. Due to the special surface material of the vehicles, specular reflection occurs on the surface of the vehicles in the near field, causing some points to be lost. Conventional image processing recognizes the shadows of buildings on the ground as edges, which can be avoided by using a semantic model. In addition, the reflectivity can provide limited feature information due to the absence of any traffic signs.

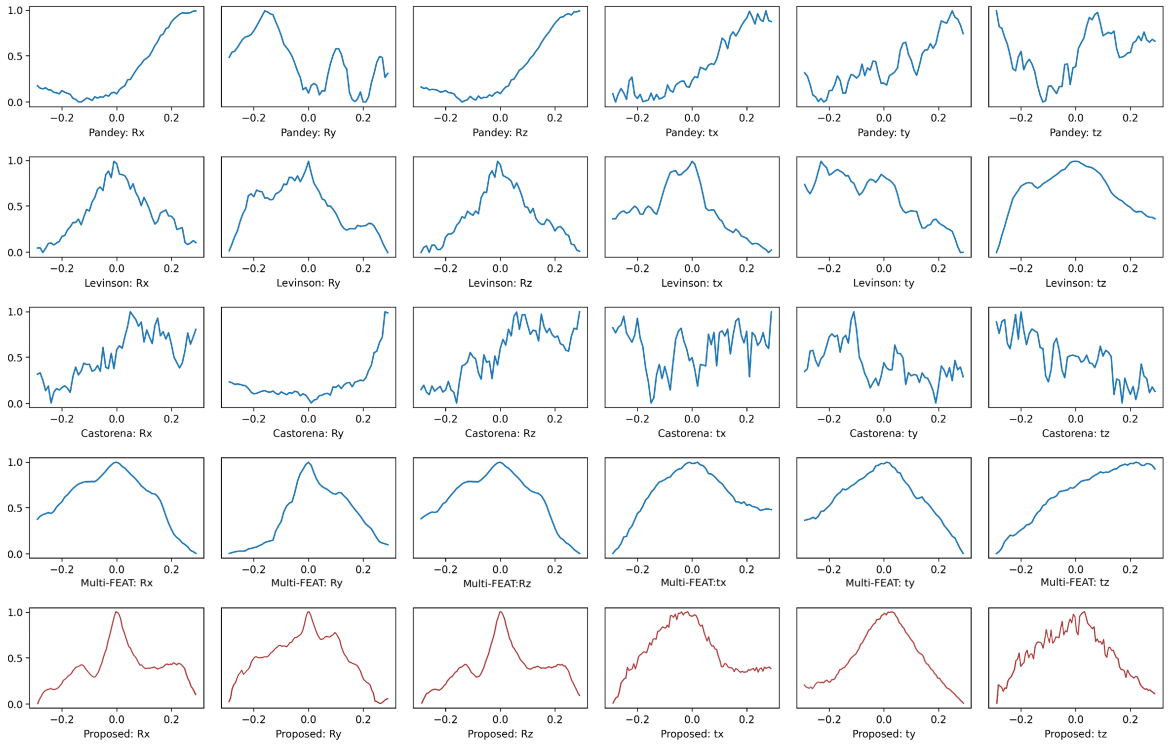


Figure 4.10: Loss curve in 6-axis - Scenario 2: Neighborhood alley.

Our approach performs best in all six dimensions, as shown in Figure 4.10. Due to

the interference of building occlusion, the matching of mutual information intensity and reflectance is not consistent, which may lead to poor performance of the mutual information-based method in this case. Despite some fluctuations in the sixth dimension, which may be due to inaccurate extraction of horizontal edges at the vehicle, the edges cannot be accurately restored when performing depth complementation due to point cloud distortions and missing point clouds on both sides of the lens, and it is not difficult to find that the point clouds of the vehicles on both sides are misaligned with the visual vehicle, even in the true-value projected image.

4.2.4 Scenario 3: Highway

The third case is the highway, where the main objects of the scene are only the vehicles. The statistical diagram of various semantic categories in the image is shown in Figure 4.11. In this case, although there is no interference from the shadows of the buildings, the shadows of the vehicles give a false contour on the ground. Figure 4.12 shows the calibrated image result.

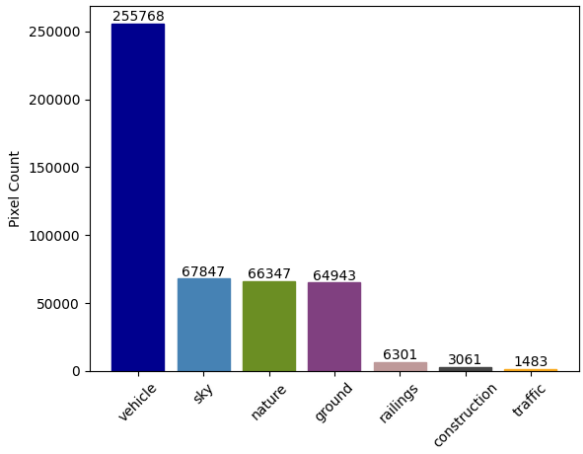


Figure 4.11: Histogram of pixel counts for semantic classes - Scenario 3: High way (Road).

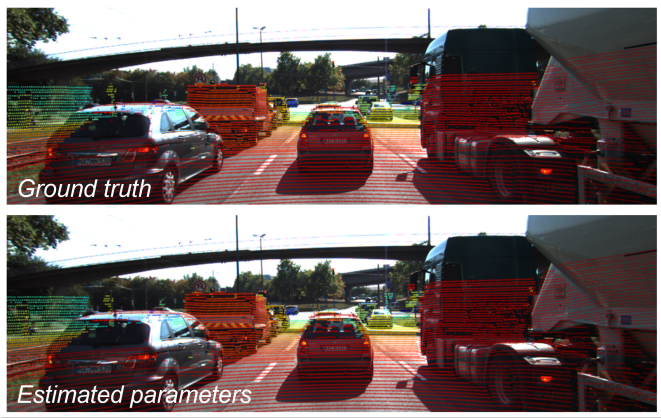


Figure 4.12: Qualitative error for one frame - Scenario 3: High way (Road).

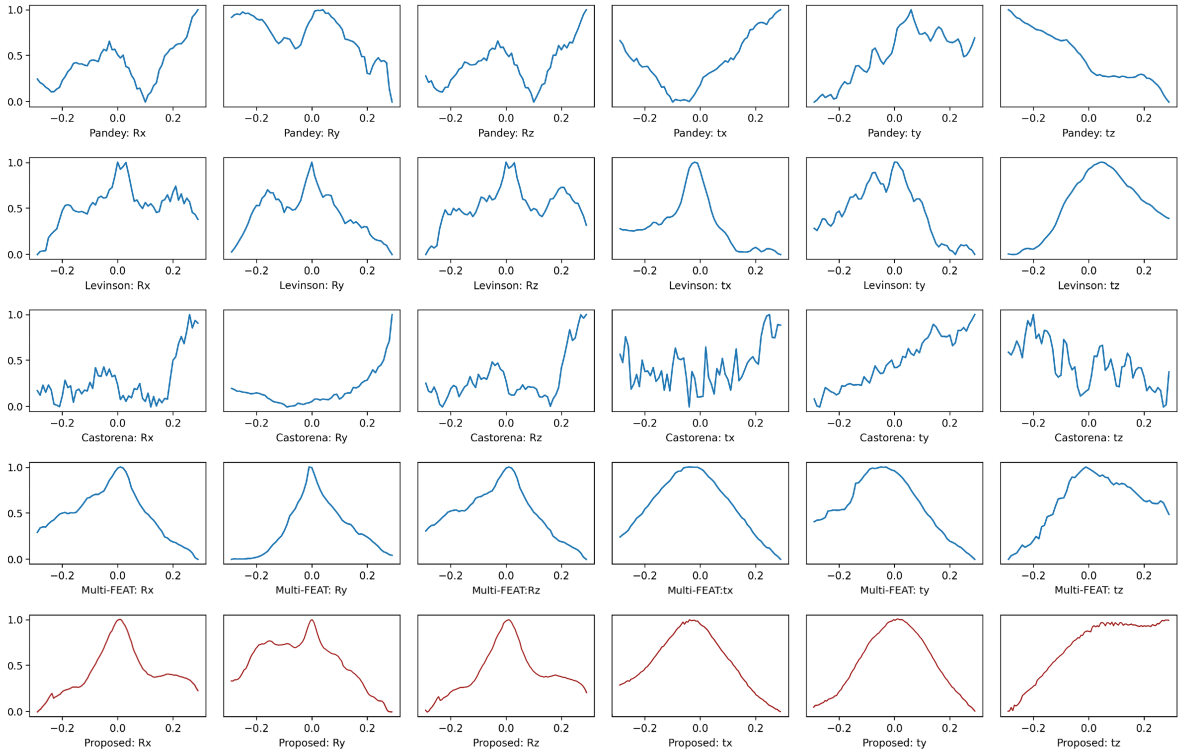


Figure 4.13: Loss curve in 6-axis - Scenario 3: High way (Road).

In Figure 4.13, Bichi's method performs better in the sixth dimension, where the shadows of the objects produced by the sunlight cause a mismatch in the maximum mutual information, which makes the mutual information-based method very non-robust to light, noise. In the loss curve for the sixth dimension, we see that the cost decreases and then increases as the sixth dimension is shifted upwards, which should be due to the fact that the point cloud has no information above the camera image field of view, while the vehicle on the right side of the image frame and the bridge located at the top both produce horizontal edges. Overall performance in the other dimensions is as expected.

4.2.5 Multi-frame Evaluation

To verify our overall performance on multiple frames, we perform a multi-frame test on 100 frames of point cloud and images, and compare with MulFEAT methods, as shown in Figure 4.14 and 4.15, and the quantitative results are shown in Table 4.4.

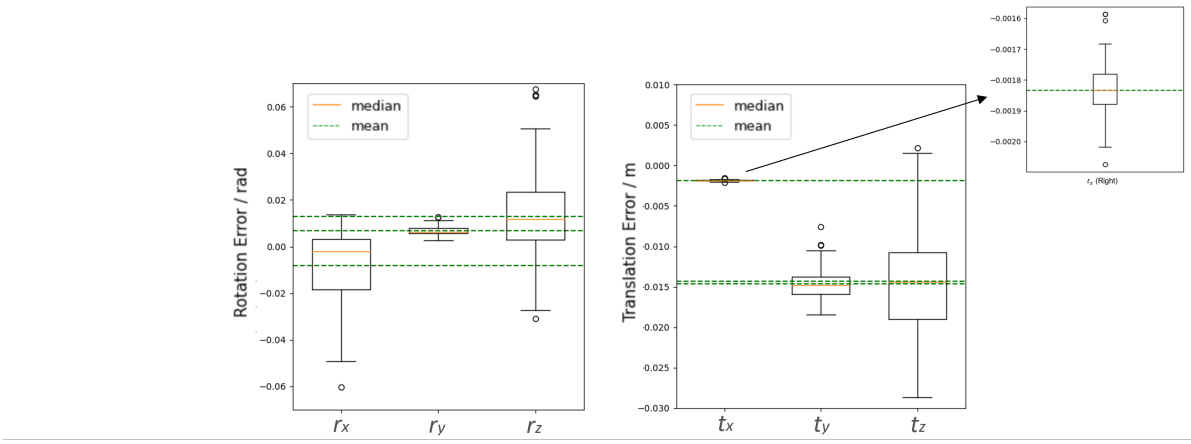


Figure 4.14: Box plot of the multi-frames results of the proposed method.

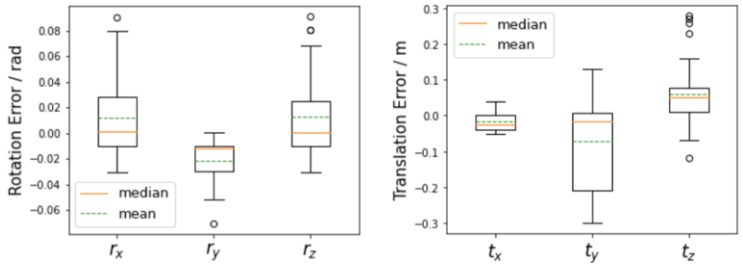


Figure 4.15: Box plot of the multi-frames results of MulFEAT. [7]

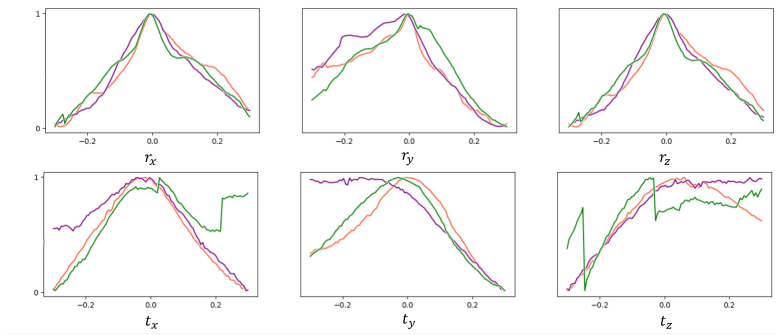
The performance of the algorithm in different dimensions is slightly different. The Yaw angle has the smallest error fluctuation. This may be because the true value of the Yaw angle is close to 90 degrees and is more sensitive to angle changes. The error fluctuations at the other two rotation angles are also within about ± 1 degree. Residual fluctuations in the vertical axis are more pronounced, probably because the point cloud does not have matching data above the image frame, and changes in the vertical direction do not cause a significant decrease in cost.

Table 4.4: Error analysis on multi-frame results.

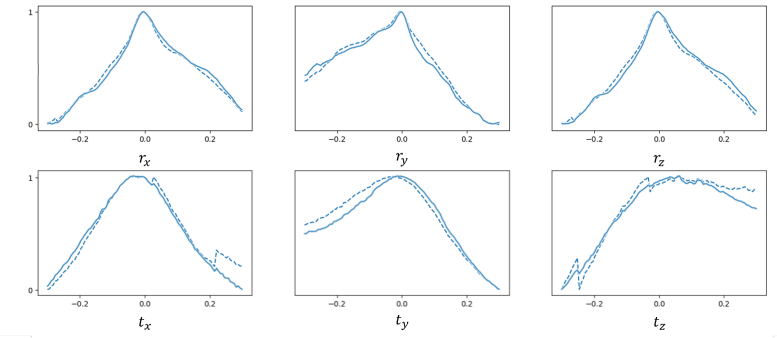
	$r_x(/rad)$	$r_y(/rad)$	$r_z(/rad)$	$t_x(/m)$	$t_y(/m)$	$t_z(/m)$
Mean error	-0.008057	0.006812	0.012980	-0.001834	-0.014615	-0.014337
Median error	-0.002031	0.006373	0.011941	-0.001834	-0.14786	-0.014435

Our algorithm has proved robust and efficient, and there is no significant deviation from the true value in the test.

According to the weighted average method we proposed in Section 3.6, we compare this method with the unweighted average method. The result is shown in Figure 4.16.



(a) Un-weighted result on 3 scenes.



(b) Weighted average result on 3 scenes.

Figure 4.16: Loss curves in 6-axis for multi-frame estimation. The orange, green, and purple lines represent three different scene frames, the solid blue line is the result of the weighted average method and the dashed blue line is the result of the unweighted average method

In the r_y dimension, the results of the weighted average method are a little sharper at the peaks, and in both the t_x and t_z dimensions, the loss curves of the weighted average method become somewhat smoother, and the phenomenon of no-peak or double-peak can be effectively avoided. Using a weighted average frame can reduce the proportion of bad frames (the main object has a small number of pixels and limited features that can be extracted) and effectively smoothes the loss curve.

4.3 Summary

In this chapter, we mainly compare five camera-LiDAR extrinsic calibration methods from principle and experiment. Among the five methods presented, the feature-based methods are single-feature to single-feature alignment, except for MulFEAT and the proposed method. The mutual information-based methods also consider only single attributes of the two modes. The complexity of such methods will be smaller, but the robustness applied to different scenarios is not strong enough. We experimented with the five methods in three scenarios, the overall performance of multi-feature-based methods is better than single-feature-based methods, and our method has a steeper loss

curve near the true value in most cases compared to MulFEAT, which is more conducive to convergence.

Mutual information-based calibration methods are particularly sensitive to light and shadows, which can lead to a mismatch in the entropy of the two attributes in the same material region, and feature-alignment-based calibration methods also have certain requirements on the scenes, but are better at resisting noise than mutual information-based methods. The inclusion of multiple features will bring a trade-off between robustness, generalizability, and computational cost.

5 Conclusion and Future Work

In this paper, we address the importance of the camera-LiDAR extrinsic parameter calibration problem for various perception tasks in the field of autonomous driving, establish a camera radar external parameter calibration model, and among the existing camera radar extrinsic parameter calibration methods, we focus on the automatic targetless approach, synthesize the traditional geometric features and the semantic features, and propose an improved semantic edge-based camera radar calibration method. This section, as a summary of the whole paper, focuses on the innovations and limitations of our approach compared to the existing methods, and suggests future work that can be improved.

5.1 Challenges of Camera-LiDAR Extrinsic Calibration

Camera-LiDAR extrinsic calibration is critical to integrating data from both sensors for accurate sensing in systems such as self-driving cars. There are several challenges associated with this task:

Sensor differences: There are very significant differences in the way camera and LiDAR sensors capture data.

- Dimensionality: Cameras record 2D images based on light intensity, while LiDAR provides 3D spatial data based on time-of-flight measurements.
- Resolution: The spatial resolution of cameras is usually higher than that of LiDAR, and when we convert the 3D-2D calibration problem to 2D-2D planar calibration, this poses a challenge for correlation between data points due to the sparsity and inhomogeneity of LiDAR projection.
- Field of view (FOV): The prerequisite of the feature-based approach is that camera and LiDAR have enough overlapping FOV, though it is true under most conditions, otherwise data fusion will be meaningless.
- Occlusion: Objects visible to one sensor may be occluded by data from another sensor, resulting in incomplete data fusion.

Data synchronization: Even with data that has been synchronized by timestamps, given the non-instantaneous nature of LiDAR scanning, points at different scanning positions will be displaced due to the accumulation of velocity at a small time lag as the vehicle undergoes relative displacement. As shown in Figure 5.1, the larger

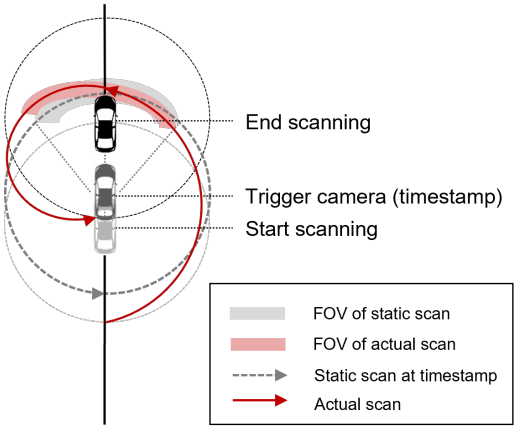


Figure 5.1: The non-instantaneous nature of LiDAR scanning for moving vehicles.

the relative velocity, the longer the LiDAR scanning interval and the larger the error, causing some alignment difficulties.

Environmental factors: Different lighting conditions can significantly affect camera data, but not LiDAR data, making consistent calibration difficult. Camera imaging depends on the color, reflectivity, and texture of the object, which are independent of LiDAR and can complicate data integration. As we discussed in our experiments, shadows caused by buildings, light, or trees may introduce redundant edges, and we will never be able to extract any edge information from LiDAR for such textures.

5.2 Innovations and Limitations

In this paper, we propose a semantically-assisted edge-based outer parameter estimation method, where the random initial values can be solved by the PnP problem for semantic point pairs. The laser point cloud is projected onto the 2D plane, and the 3D-2D correlation is converted into 2D-2D correlation. gradient edges and semantic edges are extracted and blended for the image, and for the point cloud, we blend the depth discontinuity extracted directly from the 3D space and the edge extraction from the blended feature density maps in 2D. The main innovations of this paper are as follows:

- Applying advanced SMA semantic model to image edge extraction, and utilize semantic edges and gradient edges simultaneously.
- Using SDC Net and SqueezeSegV3 pre-trained network to perform semantic recognition on images and point cloud, define semantic centroids, and solve the PnP problem associated with three sets of semantic centroids of the same category to find initial values close to the true value.
- A fast density map completion is applied and improved, which is faster than Bichi's density map completion algorithm.

- Mining the semantic information of image scenes deeply and exploiting the semantic correlation between scenes, the multi-frame averaging method is improved.

The results tested on the data set prove that our algorithm is robust and effective. Our algorithm still has limitations due to the multiple challenges of LiDAR and camera extrinsic calibration. The proposed algorithm has many hyperparameters, which need to be set according to the sensor configuration and the volume of data. Although we have not discussed the impact of these hyperparameters, unlike network model parameters based on deep learning, they can still be reasonably explained. Algorithms based on edge extraction are very much influenced by the scene, as we have shown in the example in our experiments, when there are fewer main objects (people, cars, traffic signs) in the frame, we do not have a clear object to extract the intensity of the edge discontinuity created by the object and the background, and we are not able to find potentially valid edges based on the difference in reflectivity intensity between highly reflective objects and lowly reflective objects, so the edge characterization will become very challenging, which in turn affects edge matching and optimization. Similarly, when the main object appears on both sides of the camera's field of view, the viewpoints of the point cloud and the camera will be slightly different. At the same time, missing point clouds may occur due to the relative motion of the point cloud or near-field reflections. Having the main object too far or too close in the field of view is not good for edge extraction and alignment. Although we can solve the disturbance caused by shadows and light with a semantic model, there is no solution that can deal well with the greening scene, which appears as having noisy and unclear edges in the image, and sparse and discontinuous in the point cloud, and the contours of the greening are difficult to define in both the image and the point cloud.

5.3 Future Work

Motion compensation for point clouds can be added to the pipeline of point cloud preprocessing to mitigate motion distortion of objects in the image edges. Although our approach takes into account semantic features, this is limited by the accuracy of the semantic segmentation models, and future work could consider the use of more advanced point cloud semantic segmentation models added to our existing pipeline. A possible pipeline for estimating extrinsic parameters is shown in Figure 5.2.

If two semantic models with the same labels are used, we can use class masks to match features for each class and can give different weights depending on the importance of the class, e.g. we can give less weight to edges or masks of ground, greenery and more weight to edges or submergence of people and vehicles.

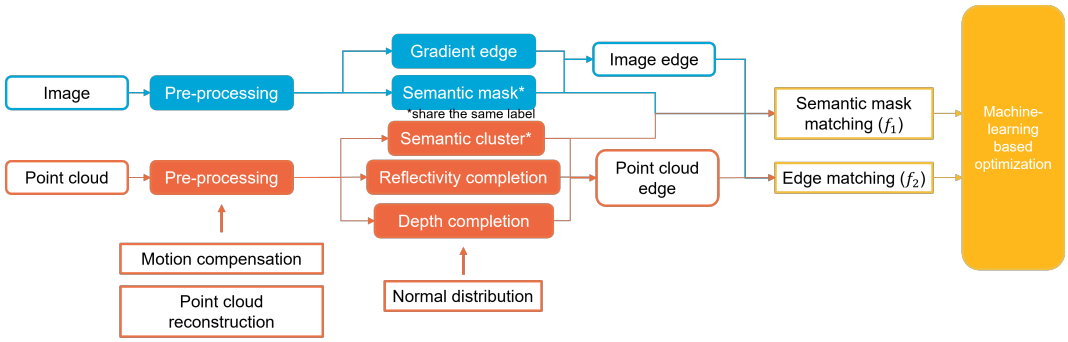


Figure 5.2: New proposed camera-LiDAR extrinsic calibration pipeline.

Considering the sparsity of the point cloud, the matrix or tensor that stores the electric cloud is also sparse, for this reason, we can consider more complex machine learning optimization models suitable for sparse data processing to improve the computational speed of the algorithm.

Engineering problems arise when the system vibrates due to drastic changes in the speed of the self-driving vehicle or exposure to strong lighting. This can lead to errors in various factors between successive frames, requiring re-calibration of the LiDAR-camera system. According to [53], the camera should be self-calibrating between successive frames to ensure that the LIDAR camera's external parameters are updated in a timely manner.

Bibliography

[1] C. Rablau, "Lidar–a new (self-driving) vehicle for introducing optics to broader engineering and non-engineering audiences," in *Education and Training in Optics and Photonics*, p. 11143_138, Optica Publishing Group, 2019.

[2] C. Debeunne and D. Vivet, "A review of visual-lidar fusion based simultaneous localization and mapping," *Sensors*, vol. 20, no. 7, p. 2068, 2020.

[3] X. Li, Y. Xiao, B. Wang, H. Ren, Y. Zhang, and J. Ji, "Automatic targetless lidar–camera calibration: a survey," *Artificial Intelligence Review*, vol. 56, no. 9, pp. 9949–9987, 2023.

[4] J. Ku, A. Harakeh, and S. L. Waslander, "In defense of classical image processing: Fast depth completion on the cpu," in *2018 15th Conference on Computer and Robot Vision (CRV)*, (Los Alamitos, CA, USA), pp. 16–22, IEEE Computer Society, may 2018.

[5] R. Kshirsagar, S. Jones, J. Lawrence, and J. Tabor, "Optimization of tig welding parameters using a hybrid nelder mead-evolutionary algorithms method," *Journal of Manufacturing and Materials Processing*, vol. 4, no. 1, 2020.

[6] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[7] B. Zhang and R. T. Rajan, "Multi-feat: Multi-feature edge alignment for targetless camera-lidar calibration," 2022.

[8] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt, M. Sokolsky, G. Stanek, D. Stavens, A. Teichman, M. Werling, and S. Thrun, "Towards fully autonomous driving: Systems and algorithms," in *2011 IEEE Intelligent Vehicles Symposium (IV)*, pp. 163–168, 2011.

[9] J. Van Brummelen, M. O'Brien, D. Gruyer, and H. Najjaran, "Autonomous vehicle perception: The technology of today and tomorrow," *Transportation research part C: emerging technologies*, vol. 89, pp. 384–406, 2018.

[10] H. Zhu, K.-V. Yuen, L. Mihaylova, and H. Leung, "Overview of environment perception for intelligent vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 10, pp. 2584–2601, 2017.

[11] K. Huang, B. Shi, X. Li, X. Li, S. Huang, and Y. Li, "Multi-modal sensor fusion for auto driving perception: A survey," 2022.

[12] M. Kam, X. Zhu, and P. Kalata, "Sensor fusion for mobile robot navigation," *Proceedings of the IEEE*, vol. 85, no. 1, pp. 108–119, 1997.

[13] Q. Li, J. P. Queralta, T. N. Gia, Z. Zou, and T. Westerlund, "Multi-sensor fusion for navigation and mapping in autonomous vehicles: Accurate localization in urban environments," *Unmanned Systems*, vol. 8, no. 03, pp. 229–237, 2020.

[14] Z. Qiu, J. Martínez-Sánchez, P. Arias-Sánchez, and R. Rashdi, "External multi-modal imaging sensor calibration for sensor fusion: A review," *Information Fusion*, p. 101806, 2023.

[15] W. Zhou, X. Yin, Z. Cao, J. Shao, *et al.*, "Two measures for enhancing data association performance in slam," *Journal of Sensors*, vol. 2014, 2014.

[16] A. Martins, J. T. Farinha, and A. M. Cardoso, "Calibration and certification of industrial sensors—a global review," *WSEAS Trans. Syst. Control*, pp. 394–416, 2020.

[17] F. M. Mirzaei, *Extrinsic and intrinsic sensor calibration*. PhD thesis, University of Minnesota, 2013.

[18] Q. Zhang and R. Pless, "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, pp. 2301–2306, IEEE, 2004.

[19] R. Unnikrishnan and M. Hebert, "Fast extrinsic calibration of a laser rangefinder to a camera," *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-05-09*, 2005.

[20] A. Geiger, F. Moosmann, Ö. Car, and B. Schuster, "Automatic camera and range sensor calibration using a single shot," in *2012 IEEE international conference on robotics and automation*, pp. 3936–3943, IEEE, 2012.

[21] P. Núnez, P. Drews Jr, R. P. Rocha, and J. Dias, "Data fusion calibration for a 3d laser range finder and a camera using inertial data.," in *ECMR*, pp. 31–36, 2009.

[22] T. Tóth, Z. Pusztai, and L. Hajder, "Automatic lidar-camera calibration of extrinsic parameters using a spherical target," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 8580–8586, 2020.

[23] D. Scaramuzza, A. Harati, and R. Siegwart, "Extrinsic self calibration of a camera and a 3d laser range finder from natural scenes," in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4164–4169, IEEE, 2007.

[24] G. Pandey, J. McBride, S. Savarese, and R. Eustice, "Automatic targetless extrinsic calibration of a 3d lidar and camera by maximizing mutual information," in *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 2053–2059, 2012.

[25] Z. Taylor and J. Nieto, "A mutual information approach to automatic calibration of camera and lidar in natural environments," in *Australian Conference on Robotics and Automation*, pp. 3–5, 2012.

[26] Z. Taylor and J. Nieto, "Automatic calibration of lidar and camera images using normalized mutual information," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, Citeseer, 2013.

[27] K. Irie, M. Sugiyama, and M. Tomono, "Target-less camera-lidar extrinsic calibration using a bagged dependence estimator," in *2016 IEEE International Conference on Automation Science and Engineering (CASE)*, pp. 1340–1347, 2016.

[28] Z. Taylor and J. Nieto, "Motion-based calibration of multimodal sensor arrays," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4843–4850, 2015.

[29] Z. Taylor and J. Nieto, "Motion-based calibration of multimodal sensor extrinsics and timing offset estimation," *IEEE Transactions on Robotics*, vol. 32, no. 5, pp. 1215–1229, 2016.

[30] D. González-Aguilera, P. Rodríguez-Gonzálvez, and J. Gómez-Lahoz, "An automatic procedure for co-registration of terrestrial laser scanners and digital cameras," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 64, no. 3, pp. 308–316, 2009.

[31] J. Böhm and S. Becker, "Automatic marker-free registration of terrestrial laser scans using reflectance," in *Proceedings of the 8th conference on optical 3D measurement techniques, Zurich, Switzerland*, pp. 9–12, 2007.

[32] J. Levinson and S. Thrun, "Automatic online calibration of cameras and lasers.," in *Robotics: science and systems*, Citeseer, 2013.

[33] J. Castorena, U. S. Kamilov, and P. T. Boufounos, "Autocalibration of lidar and optical cameras via edge alignment," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2862–2866, 2016.

[34] W. Wang, S. Nobuhara, R. Nakamura, and K. Sakurada, "Soic: Semantic online initialization and calibration for lidar and camera," *arXiv preprint arXiv:2003.04260*, 2020.

[35] P. Jiang, P. Osteen, and S. Saripalli, "Semcal: Semantic lidar-camera calibration using neural mutual information estimator," in *2021 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, pp. 1–7, IEEE, 2021.

[36] Z. Luo, G. Yan, and Y. Li, "Calib-anything: Zero-training lidar-camera extrinsic calibration method using segment anything," 2023.

[37] Z. Liu, H. Tang, S. Zhu, and S. Han, "Semalign: Annotation-free camera-lidar calibration with semantic alignment loss," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 8845–8851, IEEE, 2021.

[38] N. Schneider, F. Piewak, C. Stiller, and U. Franke, "Regnet: Multimodal sensor registration using deep neural networks," in *2017 IEEE intelligent vehicles symposium (IV)*, pp. 1803–1810, IEEE, 2017.

[39] G. Iyer, R. K. Ram, J. K. Murthy, and K. M. Krishna, "Calibnet: Geometrically supervised extrinsic calibration using 3d spatial transformer networks," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1110–1117, IEEE, 2018.

[40] X. Lv, B. Wang, Z. Dou, D. Ye, and S. Wang, "Lccnet: Lidar and camera self-calibration using cost volume network," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2894–2901, 2021.

[41] J. Li and G. H. Lee, "Deepi2p: Image-to-point cloud registration via deep classification," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 15960–15969, 2021.

[42] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[43] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *kdd*, vol. 96, pp. 226–231, 1996.

[44] J. Barzilai and J. M. Borwein, "Two-point step size gradient methods," *IMA journal of numerical analysis*, vol. 8, no. 1, pp. 141–148, 1988.

[45] J. A. Nelder and R. Mead, "A simplex method for function minimization," *The computer journal*, vol. 7, no. 4, pp. 308–313, 1965.

[46] Y. Zhu, K. Sapra, F. A. Reda, K. J. Shih, S. Newsam, A. Tao, and B. Catanzaro, "Improving semantic segmentation via video propagation and label relaxation," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 8856–8865, 2019.

[47] F. A. Reda, G. Liu, K. J. Shih, R. Kirby, J. Barker, D. Tarjan, A. Tao, and B. Catanzaro, "Sdc-net: Video prediction using spatially-displaced convolution," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 718–733, 2018.

[48] C. Xu, B. Wu, Z. Wang, W. Zhan, P. Vajda, K. Keutzer, and M. Tomizuka, "Squeezesegv3: Spatially-adaptive convolution for efficient point-cloud segmentation," in *European Conference on Computer Vision*, pp. 1–19, Springer, 2020.

[49] M. Liang, B. Yang, Y. Chen, R. Hu, and R. Urtasun, "Multi-task multi-sensor fusion for 3d object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7345–7353, 2019.

[50] Z. Liu, T. Huang, B. Li, X. Chen, X. Wang, and X. Bai, "Epnet++: Cascade bi-directional fusion for multi-modal 3d object detection."

[51] X. Wu, L. Peng, H. Yang, L. Xie, C. Huang, C. Deng, H. Liu, and D. Cai, "Sparse fuse dense: Towards high quality 3d detection with depth completion," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5418–5427, June 2022.

[52] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[53] A. D. Nguyen and M. Yoo, "Calibbd: Extrinsic calibration of the lidar and camera using a bidirectional neural network," *IEEE Access*, vol. 10, pp. 121261–121271, 2022.