

An End-to-End Multi-Sensor Fusion System for Autonomous Driving Applications

MONRROY CANO Abraham Israel

博 士 論 文

An End-to-End Multi-Sensor Fusion System for
Autonomous Driving Applications

MONRROY CANO Abraham Israel

名古屋大学大学院情報科学研究科
情報システム学専攻

2022年3月

An End-to-End Multi-Sensor Fusion System for
Autonomous Driving Applications

MONRROY CANO Abraham Israel

Abstract

Autonomous Driving technologies promise the reduction of accidents, facilitate the transportation of people, the automation of goods distribution, and other applications which currently suffer from a shortage of personnel to satisfy transportation needs. Self-driving technologies additionally pledge to improve the quality of life of their users and the people in charge of providing the service, such as drivers and distributors. For the above reason, we can consider this technology brings social value, convenience, and benefits to all the persons involved, from its development and production to its end-users.

Navigation robots and self-driving technologies require the precise integration of multiple sensors to help them perceive their surroundings, detect obstacles and often base their decisions on the data provided by these sensors to reach their goals safely. The simultaneous integration of data from multiple sensors is known as fusion, and it is used to overcome weaknesses in each sensor. Case in point, LiDARs (Light Detection and Ranging) fit medium and long-range sensing applications, while cameras are suitable for high-resolution sensing that brings a loss in three-dimensional as a downside. Wide-angle lenses attached to cameras provide a wide-angle field view. However, due to its perspective nature, distant objects appear smaller. On the other hand, cameras equipped with telephoto lenses provide long-range sensing. Nevertheless, their field of view gets significantly narrowed.

State-of-the-art robots use multiple cameras and LiDARs, among other sensors, to satisfy their sensing requirements according to their defined application. As described before, LiDARs and cameras have a limited field of view. For this reason, the integration of multiple LiDARS and cameras is considered when designing these types of vehicles. However, to achieve multi-sensor fusion, it is required to accurately obtain the camera intrinsic parameters and the extrinsic calibration among all the sensors to enable its integration.

These multi-sensing modules can be used in ADAS (Advanced driver-assistance systems) to facilitate the development of reliable safety systems such as Collision

Warning Systems, Collision Intervention Systems, and Driving Control Assistance. These systems have significantly improved the safety of the driver and its companions over the years since its inception. Moreover, these systems are constantly improving and bringing social value in the shape of safety, reducing accidents, and, more importantly, protecting the users' lives. Cameras can be used in conjunction with computer vision algorithms to obtain imminent obstacles or feed lane detection modules to notify the driver about dangerous scenarios such as lane departure or pedestrian detection to enable emergency braking systems. LiDARs, on the other hand, thanks to their wide field of view and precise range detection, can be employed to facilitate the detection of occluded objects and the development of other systems such as adaptive cruise control.

Additionally, highly accurate multi-sensor systems are essential in self-driving systems as determined by the SAE (Society of Automotive Engineers) to generate the data required to feed the perception, localization, planning, and control modules. Perception modules use data from multiple sensors to obtain information such as the surrounding obstacles, ego lane information, traffic light state, and others. In a similar manner, localization modules require data from multiple sensors to reduce the inherent measurement and quantization error while using a single sensor. Sensor fusion techniques help mitigate the error accumulation caused while the vehicle moves and produce a more reliable localization estimation.

Significant efforts have been made towards the calibration of each sensor. However, multi-sensor calibration guidelines and fusion frameworks remain unexplored and scarce. In this dissertation, we develop an end-to-end multi-sensor calibration framework to accelerate the advancement of sensing and perception systems that require precise camera-LiDAR calibration.

The first module contained in our framework is the automatic single-shot intrinsic camera calibrator. With the help of a simulator, we generated thousands of synthetic image frames containing multiple calibration targets varying their positioning and rotation, evaluated them, and selected the optimal ones. We then constructed precise guidelines to obtain intrinsic camera parameters accurately using a single image containing multiple targets in a predefined setting; We validated these with real-world cameras and lenses commonly used in robotics and autonomous

driving applications. Our findings found that using seven checkerboard targets produces repeatable and accurate camera intrinsic parameters for its use in 3D applications, such as the projection of the point cloud generated by 3D LiDARs.

The multi-LiDARs calibration module in our frame framework can accurately calculate the relative position and angle among them, or in other words, find the extrinsic calibration parameters. Our multi-LiDAR calibration module uses a method that finds the geometric similarities using the normal distribution computed on the voxelized space of each point cloud obtained from each LiDAR. After that, our method uses an optimizer to find the minimum distance between the calculated geometric similarities until it converges, resulting in the translation and rotation that relates both point clouds, and therefore obtaining the relative transformation between LiDARs. This process is repeated sequentially to calibrate all the required LiDARs extrinsically and finally adjusted to minimize the relative error caused by the voxelization step.

The camera-LiDAR extrinsic calibration. It follows a guided approach to find the euclidean transformation between the camera and the LiDAR with the help of the projection matrix obtained in the intrinsic camera parameter calibrator. The user provides feature hints related to the image and the point cloud. These features are fed to an optimizer that follows the PnP method to relate the 3D and 2D features to obtain the desired camera-LiDAR transformation.

The last module in our framework is data preprocessing and fusion. It contains methods to classify point cloud as ground, fuse data obtained from a camera, and a LiDAR using the intrinsic and extrinsic calibration parameters at pixel-cloud level, also known as low-level fusion. Additionally, it contains back-projection methods that can enable higher-level perception using deep learning methods such as semantic or panoptic segmentation, among others.

Finally, we evaluated and validated our methods on multiple sensing devices and platforms, such as robots, data collection vehicles, and Vehicle to Infrastructure (V2I) systems. We found that our framework is accurate to the sub-centimeter level, and it helps accelerate the calibration process of multiple sensors, removing the need for specialized personnel to obtain the parameters. . . .

Acknowledgements

I am incredibly grateful to my supervisors, Dr. Kato and Dr. Edahiro, for their invaluable advice, continuous support, and patience. Their guidance helped me to overcome all kinds of obstacles during my Ph.D.

Besides my advisors, I would like to thank Dr. Takeda for his continuous encouragement, support, and insightful discussions that helped widen my research and everyday life.

My sincere thanks go to Dr. Nishida, Dr. Nagao, and Dr. Ide from the real-world data circulation leading program, which assisted me through the program.

I also thank my teammates, Dr. Wong, Dr. Lambert, Mr. Kitsukawa, Mr. Tanaka, Mr. Sekino, for their support through my Ph.D. studies.

Last but not least, I would like to thank my family: my wife, my parents, and my brother for supporting me spiritually throughout my stay in Japan, my studies, and my life in general.

...

Contents

Cover	iii
Spine	iv
Abstract	v
Acknowledgements	ix
Contents	xi
List of Figures	xvii
List of Tables	xxi
List of Abbreviations	xxiii
1 Introduction	1
1.1 Problem Statement	4
1.2 Applications	5
1.2.1 Autonomous Driving	5
1.2.2 Advanced Driver Assistance Systems	6
1.2.3 Visually Impaired Persons	6
1.3 Contributions	7
1.4 Outline	8
2 Related Work	9
2.1 Autonomous Driving	9
2.1.1 DARPA Challenges	10
2.1.2 Google Driverless Car	10
2.1.3 Autoware	11

2.1.4	Other Systems	11
2.2	Camera Intrinsic Calibration	12
2.3	Sensor Extrinsic Calibration	14
2.3.1	Multi-LiDAR Extrinsic Calibration	14
2.3.2	Camera-LiDAR extrinsic calibration	14
3	Background	17
3.1	Autonomous Driving	17
3.1.1	Operational Design Domain (ODD)	19
3.1.2	Dynamic Driving Task (DDT)	19
3.1.3	Object and Event Detection and Response (OEDR)	21
3.2	Advanced Driver-Assistance Systems (ADAS)	22
3.2.1	Collision Warning Systems	23
3.2.2	Collision Intervention Systems	24
3.2.3	Driving Control Assistance Systems	25
3.2.4	Other Assistance Systems	26
3.3	Sensing Systems	27
3.4	LiDAR	27
3.4.1	Point Clouds	30
3.5	Camera	31
3.5.1	The Camer Pinhole Model	31
3.5.2	Camera Plumb Bob Model	33
3.5.3	Images	34
3.6	Sensor Fusion	35
4	Automatic Single-Shot Camera Calibration	37
4.1	Problem	37
4.2	Previous work	39
4.3	Method	41
4.3.1	Baseline Calibration	41
4.4	Simulation	42
4.4.1	Checkerboard Coordinate System	42
4.4.2	Simulator Coordinate System	43

4.5	Checkerboard Corner Detector Evaluation	43
4.5.1	Corner Detector Metrics	43
4.5.2	Experiments	44
4.5.3	Results	45
4.6	Simulated Calibration Experiments	46
4.6.1	Checkerboard Pose Metrics	47
4.6.2	Control Points	48
4.6.3	Dual Checkerboard Calibration	49
	Dual Checkerboard Rotation Experiments	50
	Dual Checkerboard Horizontal Positioning Experiments	51
	Dual Checkerboard Vertical Positioning Experiments	53
	Dual Checkerboard Distance Experiments	55
4.6.4	Dual Checkerboard Calibration Results	55
4.6.5	Multiple Checkerboards Calibration	56
4.6.6	Multiple Checkerboards Calibration Results	57
4.7	Real-world Calibration Verification	59
4.7.1	Multiple checkerboard verification experiments	59
4.7.2	Real-world Calibration Results	62
4.8	Conclusion	65
5	Multi-Sensor Fusion Toolbox for Autonomous Driving	69
5.1	Introduction	69
5.2	Related Work	71
5.2.1	LiDAR-LiDAR extrinsic calibration	71
5.2.2	Camera-LiDAR extrinsic calibration	71
5.2.3	Camera-LiDAR fusion	73
5.2.4	Point Cloud Ground Classifier	73
5.3	Theory and Implementation	74
5.3.1	LiDAR-LiDAR extrinsic calibration	74
	Theory	75
	Implementation	76
5.3.2	Camera-LiDAR extrinsic calibration	77

	Theory	78
	Implementation	79
5.3.3	Image-Cloud fusion	80
	Theory	81
5.3.4	Ground classification	82
	Theory	82
	Implementation	85
5.4	Evaluation	85
5.4.1	LiDAR-LiDAR extrinsic calibration	86
5.4.2	Camera-LiDAR extrinsic calibration	88
5.4.3	Image-Cloud Fusion	89
5.4.4	Ray Ground Classifier	91
5.5	Discussion	92
5.5.1	LiDAR-LiDAR Extrinsic Calibration	92
5.5.2	Camera-LiDAR Extrinsic Calibration	94
5.5.3	Image-Cloud Fusion	96
5.5.4	Ray Ground Classifier	96
5.6	Conclusion	97
6	Real World Applications	99
6.1	Introduction	99
6.2	Related Work	100
6.3	Design and Implementation of Hexacam	101
6.3.1	FPGA board	102
6.3.2	Camera sensors	103
6.3.3	Image processing	104
6.3.4	Network transmission to host	106
6.3.5	Object detector	108
6.4	Performance Evaluation	110
6.4.1	Camera sensors	110
6.4.2	Network speed	110
6.4.3	Perception System performance	110

6.4.4	Power consumption performance	112
6.5	System Calibration and Fusion	112
6.6	Conclusion	114
7	Real World Data Circulation in Multi-Sensor Systems	117
7.1	Introduction	117
7.2	Automatic Single-Shot Camera Calibration	118
7.3	Multi-Sensor Fusion Toolbox for Autonomous Driving	121
7.4	Leading a Start-Up Company	123
7.5	Summary	125
8	Conclusions	127
	Bibliography	129

List of Figures

1.1	LiDAR Market Expansion Forecast.	3
1.2	Road Traffic Accidents in Japan with Fatalities	6
1.3	Road Traffic Accidents in Japan with Serious Injuries	7
2.1	Autonomous Driving History	12
3.1	Levels of Automation SAE J3016.	18
3.2	Autonomous Mode Functional Architecture Flow Diagram.	19
3.3	ADAS Collision Warning Systems	24
3.4	LiDAR classifications according to its scanning method.	28
3.5	Camera Pinhole Model.	32
3.6	Types of Radial Distortion.	33
4.1	Our modeled checkerboard simulated in the LGSVL.	42
4.2	Effects of roll (α), pitch (β) and yaw (γ) rotations on the checkerboard corner detector.	46
4.3	Effects of distance between the checkerboard and the camera on the checkerboard corner detector.	47
4.4	Control points in camera frustum	48
4.5	Control points as an auxiliary metric	49
4.6	Quantitative result for multiple checkerboards	57
4.7	Multiple checkerboard verification experiments in a garage	62
4.8	Point cloud projection on the Lucid camera using the intrinsic param- eters by the one-shot experiments.	63
4.9	Point cloud projection on the FLIR camera	64
4.10	Real-world results for the multi-checkerboard experiments.	67

5.1	Successful calibration result of six wide-angle cameras	70
5.2	High-level diagram of the LiDAR-to-LiDAR calibration tool.	77
5.3	High-level diagram of the camera-to-LiDAR calibration tool.	79
5.4	UI presented to the user to ease the selection of corresponding points.	80
5.5	Result of the real time Image-Cloud fusion	81
5.6	High-level diagram of the image-cloud fusion.	81
5.7	Ray-Ground Filter radial dividers	83
5.8	Ray Ground Filter classification diagram	84
5.9	High-level diagram of the Ray Ground Classifier	85
5.10	Ray ground filter on KITTI dataset	86
5.11	Prius Vehicle Setup	87
5.12	Alphard Vehicle Setup	88
5.13	Prius Setup Sensor Calibration Qualitative	88
5.14	Prius Setup Sensor Calibration Quantitative results	90
5.15	Alphard Setup Sensor Calibration Quantitative results	91
5.16	Translation absolute error compared with number of selected points on the Alphard Setup.	92
5.17	Rotation absolute error compared with number of selected points on the Alphard Setup.	93
5.18	Fusion results on different setups	94
5.19	Ground Classification qualitative results	95
5.20	Successful calibration of a thermal vision camera	95
6.1	An overview of our prototype system.	102
6.2	Picture of the FPGA board prototype.	103
6.3	Board architecture.	103
6.4	Custom tailored FPGA board diagram.	104
6.5	RGB color reconstruction from RAW image.	105
6.6	Left side, original image. Right side, artifacts due to pixel interpolation	106
6.7	HOG descriptor calculation process	109
6.8	Network speed measured while capturing	111
6.9	Framerate for each camera while capturing	112

6.10	Processing time on GPU	113
6.11	Processing time on CPU	113
6.12	Qualitative results of the Image-cloud on the HexaCam device and a Velodyne VLP16.	114
6.13	Qualitative results of the Image-cloud on the Ladybug camera and a Velodyne HDL64.	114
7.1	Continuous improvement RWDC loop applied to sensing systems. . .	119
7.2	Percent change in claim frequency associated with collision avoidance technologies	119
7.3	Example Vehicles using real-world multiple sensor calibration tech- nologies developed by Perception Engine.	124

List of Tables

1.1	Sensor comparison in ADAS systems	3
2.1	Feature comparison among the analysed Camera-LiDAR algorithms.	15
4.1	Summary of the rotation experiments with dual checkerboards and their results	51
4.2	Summary of the horizontal and vertical positioning experiments with dual checkerboards and their results.	54
4.3	Summary of the distance between camera and checkerboard experiments with dual checkerboards and their results.	56
4.4	Summary of the simulation experiments with multiple checkerboards	58
4.5	Summary of the results for the simulation experiments with multiple checkerboards.	59
4.6	Summary of the extrinsic parameters between the cameras and the 3D LiDAR for the outdoors dataset	60
4.7	Summary of the real-world experiments with multiple checkerboards	61
5.1	Feature comparison among tested Camera-LiDAR algorithms.	73
5.2	Prius Setup Sensor Calibration Quantitative	87
5.3	Alphard Setup Sensor Calibration Quantitative	89
5.4	Camera-LiDAR extrinsic calibration results comparison	89
5.5	Fusion measurements for different sensors	90
5.6	Classification performance of the Ray Ground Classifier	91
6.1	Camera sensor supported resolutions and transfer rates	104
6.2	FRAME_START packet structure	107
6.3	Image reception packet queue	107

6.4	FRAME_END packet structure	108
6.5	Packet Queue	109
6.6	Comparison between HexaCam and Ladybug	111
7.1	Single-shot calibration related to the Data Circulation	120
7.2	Multi-Sensor Fusion Toolbox related to the Data Circulation	122

List of Abbreviations

LiDAR	L ight D etection A nd R anging
SLAM	S imultaneous L ocalization A nd M apping
ROS	R obot O perating S ystem
NDT	N ormal D istributions T ransform
ADS	A utonomous D riving S ystem
ADAS	A dvanced D river- A ssistance S ystem
ODD	O perational D esign D omain
DDT	D ynamic D riving T ask
OEDR	O bject and E vent D etection and R esponse
FCW	F orward C ollision W arning
LDW	L ane D eparture W arning
CTW	C ross T raffic W arning
BSW	B lind S pot W arning
FCW	F orward C ollision W arning
AEB	A utomatic E mergency B raking
DBS	D ynamic B reak S upport
CIB	C rash I mminent B raking
PAEB	P edestrian A utomatic E mergency B raking
RAB	R ear A utomatic B raking
BSI	B lind S pot I ntervention
ACC	A daptive C ruise C ontrol
LCA	L ane C entering A ssistance
FCW	F orward C ollision W arning
LKA	L ane K eeping A ssistance
AHB	A utomatic H igh B eams
ACN	A utomatic C rash N otification
PDL	P roperty D amage L iability
BI	B odily I njury
PIP	P ersonal I njury P rotection
MedPay	M edical P ayment

Chapter 1

Introduction

Recent advancements in computing, sensing, and robotics have been the driving force to the swift progress in the development of autonomous driving vehicles. These self-driving cars promise the reduction of accidents, facilitate the transportation of the elderly, the automation of goods distribution, and other applications which are or might suffer shortly from a shortage of personnel to satisfy transportation needs. Additionally, self-driving technologies also promise to improve the quality of life of its users and the people in charge of providing the service; This brings social value, convenience, and joy to all the persons involved in the research, development, production up to the end-user.

Autonomous vehicles require a highly accurate representation of their surroundings to navigate and reach their target safely. Sensors such as cameras, radars, and LiDARs are commonly used to provide rich perception information. Each of these sensors can complement each other to supply reliable and accurate data. For instance, cameras produce a dense representation of the world, including color, texture, and shape. However, cameras cannot provide reliable depth information at longer distances. On the other hand, LiDARs capture sparse but highly accurate range information at short, middle, and in some cases at long range regardless of the lighting conditions. For this reason, integrating data from multiple sensor sources is desired.

Cameras have become ubiquitous thanks to their low cost, high quality, and ability to represent the world with dense and feature-rich images. The images created by these devices resemble our own vision, depicting objects located at different distances with different apparent dimensions. The mathematical model commonly

used to project the three-dimensional world is the pin-hole camera model. In addition, the plumb-bob model, also known as the Brown-Conrady model, represents the distortion caused by the lens attached to the camera [1].

LiDAR, on the other hand, measures its surroundings using lasers transceivers, generating highly accurate 3D sparse points, commonly known as point clouds [2]. Unlike the images generated by cameras, these point clouds use euclidean coordinates that allow the precise representation of depth.

The simultaneous integration of data from multiple sensors is known as fusion, and it is used to overcome weaknesses in each sensor. For instance, to project the point clouds obtained from 3D LiDAR sensors on the images generated by cameras and use the highly accurate range data contained in the point cloud. It is essential to get the sensor's relative physical position and define a transformation method between the representational sensor data. This process is known as extrinsic calibration. Despite the calibration of each sensor has been widely studied [3–8], the definition of clear guidelines for multi-sensor calibration, its fusion, and a consistent preprocessing frameworks still remain unexplored and scarce.

ADAS systems often use other sensors such as radars, ultrasonic, and stereo camera setups. However, this work focuses on developing multi-sensor fusion using cameras and LiDARs. Table 1.1 summarizes of each sensor used in ADAS systems, which helps understand our sensor selection. Cameras provide high-resolution and rich information based on the focal length of the lens installed. Case in point, wide-angle lenses capture an image with a wide field of view. Nevertheless, long-range objects are projected in a small sensor area due to their perspective nature, limiting the use of the data or in some cases drawing it unusable. On the other hand, Telephoto lenses capture objects at long ranges in a narrow field of view. LiDARs as a complementary sensor delivers centimeter-level accurate 3D measurements from short to long ranges on a limited resolution. Multiple cameras and LiDARs balance each other and help to solve many of the challenges introduced in Sections 3.1 and 3.2.

Some might argument LiDARs sensors are expensive for its deployment in production vehicles. This holds true for current generation sensors, which are produced in limited volumes. Nevertheless, in the last decade, we have witnessed a swift price reduction thanks to the constant development and application expansion of LiDAR

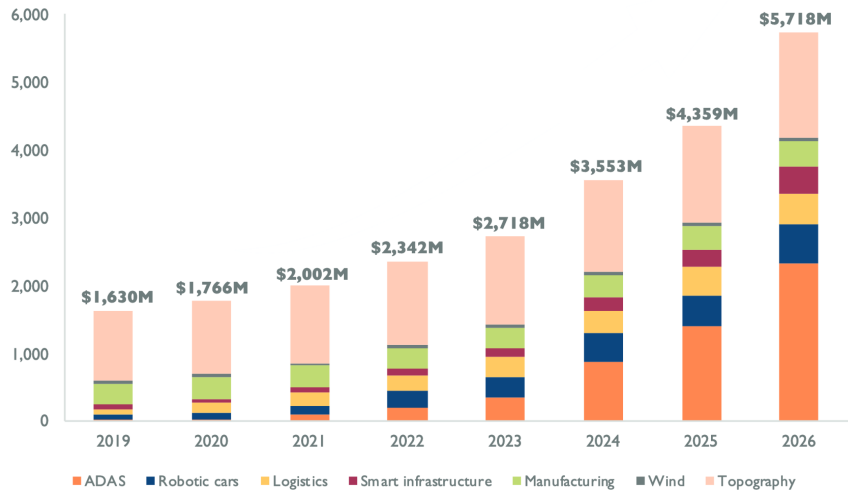


FIGURE 1.1: LiDAR Market Expansion Forecast.

sensors. Case in point, according to the Yole development group [9], the expansion of LiDAR is expected to increase by more than 300% in the next 5 years as shown in Figure 1.1.

TABLE 1.1: Sensor comparison in ADAS systems

Sensor	Range	Accuracy	Resolution
Camera	Variable	Good	Excellent
LiDAR	High	Excellent	Good
RADAR	Very High	Medium	Medium
Ultrasonic	Low	Medium	Good
Stereo Cameras	Medium	Good	Excellent

Given the accuracy of the LiDARs, the ubiquity of the cameras, and the promising foreseeable future of both sensors, in this work we present an end-to-end sensor fusion and data preprocessing framework that simplifies the calibration process for multiple cameras and LiDAR sensors for robotics and autonomous driving applications, and methods for data preprocessing widely tested in the real world on autonomous vehicles and navigation robots. Nevertheless, in future work we aim to integrate other sensors to our framework. Each sensor has its unique applications and depending on the specification of the ADAS system, or the ODD definition (Section 3.1.1) the final cost of the system can be reduced to match the requirements using different sensors.

1.1 Problem Statement

Given a number of cameras and LiDARs sensors, we are interested in obtaining the cameras projection parameters, the relative position among the sensors, and the processing methods to merge and employ the images and point clouds in a fusion system. In particular, we aim to answer the following questions:

- How to intrinsically calibrate the cameras using a single frame with multiple targets?

Existing methods commonly use multiple frames and a single target with random positionings. In this dissertation, we aim to use multiple targets in a single frame instead. This kind of setup helps accelerate and obtain reproducible intrinsic parameters since multiple strategically positioned targets can be fixed and used multiple times. In Chapter 4 we present guidelines developed with the help of thousands of images generated by a simulator to define the optimal multiple target poses in a single frame to obtain accurate camera intrinsic parameters.

- How to obtain the relative position between cameras and LiDARs?

In order to fuse lidars and cameras, having an accurate camera-LiDAR extrinsic calibration system is required. We present a method on how to achieve this in Chapter 5.

- How to get the relative position between multiple LiDARs?

Surround sensing is essential for safety. For this reason, installing multiple LiDARs is commonly used to cover the required field of view, which brings the issue of requiring the extrinsic calibration to enable its fusion. We introduce a method to achieve this in Chapter 5.

- How to use the obtained relative position among the sensors to merge the data in a fusion system?

Setups with multiple cameras and LiDARs require intrinsic camera parameters and the camera-LiDAR extrinsic parameters. We present methods on how

to use the fused data from multiple LiDARs and sensors in Chapter 5. Additionally, we present real-world applications in Chapters 6 and 7.

- Is the calibration accurate enough for 3D applications?

Accurate intrinsic and extrinsic parameters is essential to fuse the data from each sensor, in the case of self-driving system an accuracy of at least 0.1 m is required, as we further analyse in Chapter 3.

We try to answer the above questions through a multi-sensor fusion framework developed with the help of synthetic data and validated in the real world with autonomous vehicles and navigation robots featuring different number of sensors.

1.2 Applications

We highlight three critical applications of multi-sensor fusion.

1.2.1 Autonomous Driving

According to the World Health Organization (WHO), more than one million persons die in road traffic accidents every year, causing more deaths than HIV/AIDS and other common diseases combined [10].

In Japan, even if the number of fatal accidents has been decreasing over time, in 2019, more than 3,000 fatal cases were registered, while the number of accidents with severe injuries surpassed 30,000 incidents as we see in Figure 1.2 and Figure 1.3.

The ultimate goal of autonomous driving is to substitute the human driver with an intelligent system that can process the incoming sensor information and react appropriately to maneuver the vehicle between two positions. This technology can reduce traffic accidents and vehicle emissions significantly [11, 12], for example, by increasing road capacity and reducing traffic congestions [13]. So far, autonomous driving has been successfully demonstrated on highways with little or no traffic. Busy intersections and urban navigation, however, are well-known challenges.

Sensing systems are one of the integral pieces forming an autonomous driving platform, these provide raw data to other modules such as perception, localization, control and planning. We explain further these integrations in Section 3.1.

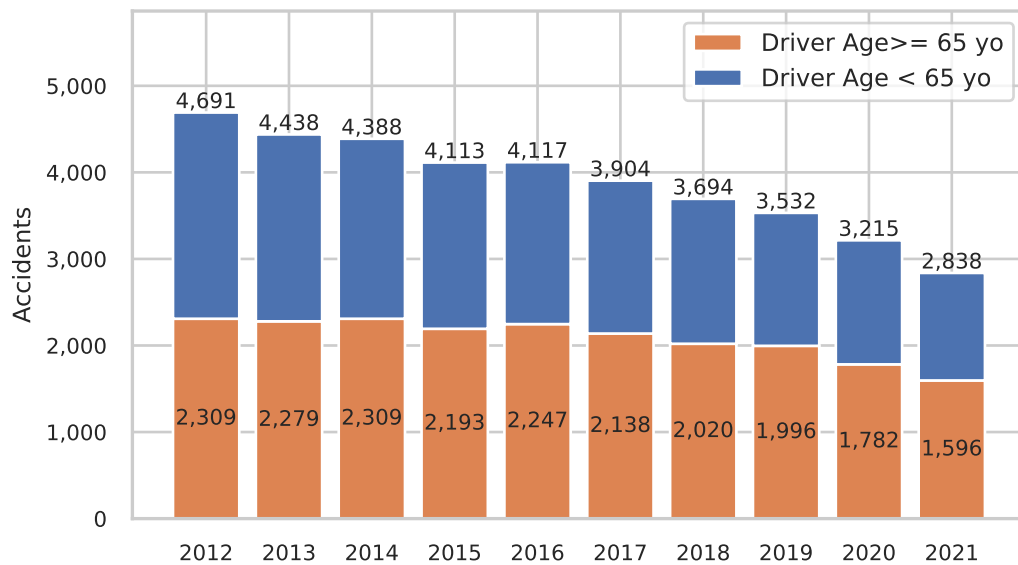


FIGURE 1.2: Number of fatal road traffic accidents in Japan during the last 10 years.

1.2.2 Advanced Driver Assistance Systems

While the full deployment of self-driving vehicles at a large scale might still be decades away, numerous research outcomes are doing their way into commercial ADAS, such as lane departure warning [14], automatic parking or collision avoidance, pedestrian detection, among others [15]. A multi-sensor fusion system can enhance the perception and sensing capabilities of other ADAS systems to improve the general safety of vehicle riders. Case in point, accurate intrinsic calibration of cameras is essential for the correct detection of lanes for collision warning and avoidance systems (Section 3.2.1). LiDAR and camera fusion systems can additionally enable a wide variety of ADAS features such as cross traffic warning, blind spot warning systems, or automatic braking systems (Section 3.2.3)

1.2.3 Visually Impaired Persons

Accurate multi-sensor fusion systems can be used in perception systems or navigation robots to provide mobility to those with limited vision or other physical disabilities. LiDAR sensors are constantly reducing in price, and improving its

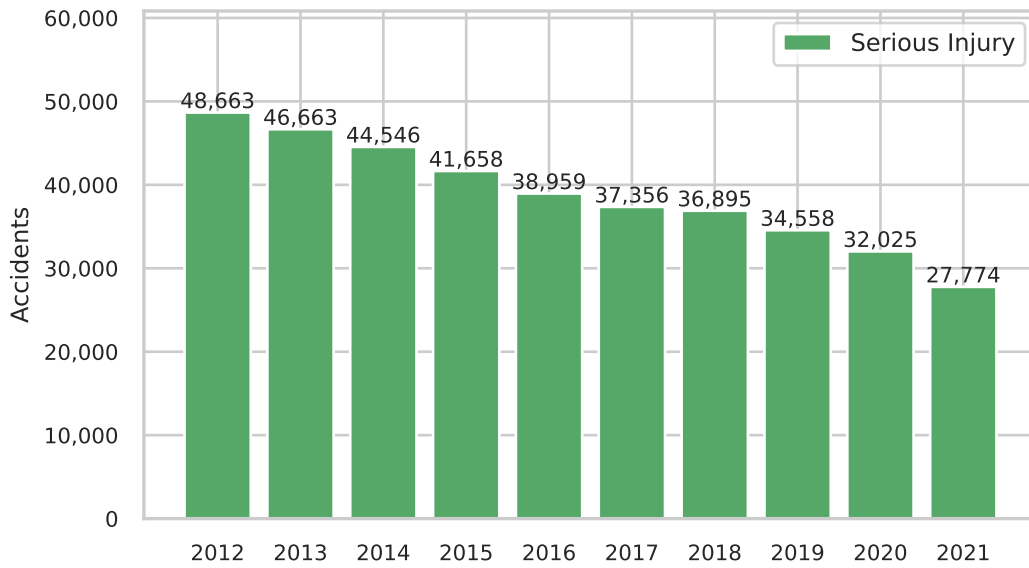


FIGURE 1.3: Number of road traffic accidents that caused serious injuries in Japan during the last 10 years.

performance and measurement accuracy. Additionally low-cost cameras and well-established computer vision techniques [16–18] can provide robust perception, altogether contributing to a higher quality of life and an increased user safety.

1.3 Contributions

The contributions of this dissertation are as follows:

- A novel method to obtain the intrinsic camera parameters using a single shot for three-dimensional applications. In contrast with existing techniques, we validate its accuracy with the help of thousands of synthetic images and later verified with real machine vision cameras. We additionally project the point cloud obtained from a high-resolution LiDAR to validate the parameter accuracy.
- A multi-sensor semi-supervised calibration framework that accelerates and facilitates the fusion process of cameras and LiDARs for self-driving applications. This framework also contains methods to perform data preprocessing to

classify the fused point cloud from multiple LiDAR sensors and the real-time low-level integration from RGB and range data.

- Extensive evaluations on multiple devices setups and vehicles show the method's applicability and confirm that the framework facilitates the fast integration of multi-sensor systems that provide reliable and accurate sensing information for higher levels of perception. Additionally, we present a camera-LiDAR sensing system as a proof-of-concept for surround perception that utilizes directly the two above items.

1.4 Outline

This dissertation is structured as follows; Chapter 2 reviews the current state-of-the-art and differences the proposed technique to previous work. Chapter 3 introduces the background and concepts used over the rest of this thesis. Chapter 4 presents an automatic single-shot camera calibration technique to obtain accurate camera projection parameters required to fuse the image with other sensor data. Chapter 5 presents a multi-sensor fusion toolbox that accelerates the calibration of multiple LiDAR sensors and cameras. It additionally introduces fusion applications derived from the fused data, such as ground segmentation and RGB-range fusion. Chapter 6 presents a further evaluation of the fusion methods and their application on different devices, self-driving vehicles platforms, and navigation robots. Chapter 7 exhibits the application of these techniques in the real world through data circulation. Case in point, technology applied to the industry with loop feedback aiming for constant improvement. Finally, Chapter 8 concludes this manuscript and presents the remaining challenges in this line of work.

Chapter 2

Related Work

This chapter discusses the state-of-the-art multi-sensing systems for self-driving and positions the contributions of this manuscript concerning other work. We begin with a summary of the development of autonomous driving systems, survey their capabilities, and present the open challenges.

2.1 Autonomous Driving

Since early as the 1920s across the United States of America, experiments with driverless vehicles started with the "Phantom Autos." Newspapers of the time called them "one of the most amazing products of modern science." These vehicles could brake, maneuver the steering wheel, sound the horn, and advance thus as though there was an invisible driver at the wheel. People from all over the continent traveled to witness the technology, [19]. Many of the attendants were looking to expect these splendid cars, and many exclaimed, "The blind were safe for the first time. Parents found they could more safely send their children to school in the new car than in the old cars with a chauffeur." A piece of the newspaper from that time can be appreciated in Figure 2.1. However, these phantom cars were not computer-driven but remote-controlled with the help of morse-code. The thing people focused on most was the promise of improved safety at a time when vehicles were deadly. Hundred years later, these words still hold. It was not until the 1980s that the first and self-driving vehicles appeared. Carnegie Mellon University (CMU) developed the Navlab and the Autonomous Land Vehicle Project (ALV), which had equipped cameras, road detection, obstacle avoidance, and navigation algorithms [20, 21]. A picture of that vehicle can be found in Figure 2.1

In the following subsection we introduce notable projects such as the Defense Advanced Research Projects Agency(DARPA), the Google Driverless car, and the Autoware project which have significantly shaped the autonomous driving technology field.

2.1.1 DARPA Challenges

The DARPA Grand Challenge was a prize competition for American autonomous vehicles, funded by the Defense Advanced Research Project Agency(DARPA), the most significant research organization of the United States Defense Department. This Challenge was created to encourage the development of self-driving technologies to create a fully autonomous ground vehicle capable of completing a substantial off-road course within a limited time [22]. The first challenge was held on March 13, 2004 in the Mojave Desert (USA), along a 240 km route. No winner was declared, and the cash prize was not given. Therefore, a second competition was carried out on October 8, 2005. Five vehicles successfully completed the 212 km course. It was won by the Stanford Team [23]. In 2007, DARPA initiated the Urban Challenge, its purpose was to benchmark the state-of-the-art in autonomous inner-city driving on a 96 km test course at an abandoned Air Force Base. While this Urban Challenge endeavor came closer to urban traffic scenarios, the streets were wider, and the field of view was occluded only by a few other vehicle participants [24]. The vehicles that participated in the Urban Challenge are shown on Figure 2.1.

These projects used multiple laser sensors, cameras and other ranging sensors, each used individually for a different purpose: perception, localization, short-range sensing. Nevertheless, no signs of low-level or extrinsic calibration were mentioned in this project to achieve multi-sensor fusion.

2.1.2 Google Driverless Car

In 2010 Google officially initiated the driverless car project. The vehicle was equipped with a 3D laser scanner, a Velodyne spinning LiDAR, similar to vehicles participants of the DARPA Urban Challenge. Figure 2.1 shows the vehicle setup at the time. In

August 2012, Google announced that their car completed over 300,000 miles without accidents. In 2017, the project was renamed Waymo and spun off as a new start-up company part of the Alphabet group [25]. Waymo's latest vehicle uses multiple cameras, LiDARs, and radars to perceive the areas around it, as shown in Figure 2.1. This setup allows the car to see 360 degrees in every direction, day or night, and at long ranges [26]. Similarly, this work aims to develop automatic and semi-supervised calibration techniques for multiple sensors. The Waymo project, a private company, does not publicly state which methods or algorithms to achieve the sensor calibration. In contrast, this work targets to develop open methods for multi-sensor fusion.

2.1.3 Autoware

The Autoware project started in 2015 at Nagoya University as a research project. It was the first open-source Autonomous driving framework based on the widely used Robot Operating System (ROS) [27]. For this reason, it quickly became famous all over the world. In 2018, the first embedded version of Autoware aimed at low power consumption systems was released [28]. During the same year, The Autoware Foundation was formed to promote the global development of Autoware by creating synergies among the worlds' leading tech companies, academic/non-profit organizations, and individual contributors [29]. The Autoware foundation fits with the objectives of this work, the global and open development of technologies for the development of self-driving technologies.

2.1.4 Other Systems

In addition to the previously mentioned self-driving systems. Many other private endeavors are working on the development of autonomous driving technologies. Some worth mentioning efforts are Aptiv, Baidu, Lyft, Nuro, and Zoox. However, except for Baidu and partially Lyft, none of them publicly share their sensor calibration techniques in the form of open-source software.

FIGURE 2.1: Autonomous Driving History



(A) Phantom vehicle advertisement on the newspaper in 1935[30]



(B) Carnegie Mellon University's NavLab vehicle in 1983



(C) Participant Vehicles to the DARPA Urban Challenge in 2007



(D) Google's Vehicle in 2010



(E) Google's Vehicle in 2010

2.2 Camera Intrinsic Calibration

There exists a considerable amount of work dedicated to developing techniques for estimation of camera intrinsic parameters. Notable mentions include the work by Zhang [31], Kannala and Brandt [32], and Heikkila and Sliven [33]. Despite being published more than twenty years ago, these methods provide consistent and reliable results. Moreover, the widely-used open-source computer vision library OpenCV [34] and the proprietary Matlab [35] platform use these methods in

their camera calibration toolboxes due to their proven accuracy. More recent approaches use deep learning methods to estimate the camera intrinsic parameters using neural networks trained on large datasets of images with known intrinsic parameters [36][37]. These methods are convenient since they do not require any targets or calibration datasets. Nevertheless, these approaches are still far from matching the accuracy achieved by target-based techniques.

Zhang [31] used synthetic data while testing his calibration method to evaluate resilience against noise. He obtained good results with as few as three checkerboards, without aiming to use the parameters in 3D applications. However, he only used the checkerboard corners to measure the error, which might result in over-fitted parameters. Moreover, he did not consider the error introduced by the corner detection phase.

The work dedicated to the extrinsic calibration of LiDARs, radars, and cameras explicitly states that accurate intrinsic camera parameters are required [3, 6–8, 38–42]. These methods find shared features between the 2D perspective space on the images generated by the camera and the 3D Euclidean space employed by radars and LiDARs. The shared features are then input to an optimizer to estimate the extrinsic parameters (relative position \mathbf{t} , and rotation \mathbf{R}), which attempts to reduce the projection error of the 3D features ($\mathbf{p}_{\text{lidar}}$) while using the given camera intrinsic parameters ($\mathbf{P}_{\text{lidar,cam}}$) in the form $\mathbf{p}_{\text{cam}} = \mathbf{P}_{\text{lidar,cam}} \cdot \mathbf{R} \cdot \mathbf{t} \cdot \mathbf{p}_{\text{lidar}}$. This equation illustrates the importance of having high-quality camera intrinsic parameters in order to obtain accurate sensor extrinsic parameters.

There are a limited number of published studies about verifying estimated camera intrinsic parameters for use in 3D applications. Basso et al. [4] stressed the requirement of accurate intrinsic parameters for 3D applications such as SLAM, and introduced a method for the intrinsic and extrinsic calibration optimizer for short-range time-of-flight (ToF) sensors such as the Microsoft Kinect. Geiger et al. [5] presented a single-shot calibration method for short and long-range LiDARs and cameras. Their approach uses multiple checkerboards in a single frame to accelerate and simplify the data acquisition. However, they did not analyze or demonstrate why these positions are optimal. We extend this research direction to create clear guidelines on how to achieve consistent and accurate intrinsic parameters and introduce

validation metrics to verify them.

2.3 Sensor Extrinsic Calibration

Extrinsic calibration can be defined as the process of calculating the relative position in space between sensor coordinate frames. This can be achieved by looking for co-observable features in the data from both sensors. Calibrating different sensors may use similar optimization methods. However, the features to search for are dependent on the sensor. In the following subsections we will summarize some of the most recent developments in each area.

2.3.1 Multi-LiDAR Extrinsic Calibration

To the best of our knowledge, at the time of writing, very few publications can be found that address the calibration of multiple multi-layer LiDAR sensors. In [43] a method to calibrate a single-layer LiDAR is presented. This work maximizes the mutual information entropy, through the estimation of the projection coefficients between the spaces.

The work presented in [44], shows a semi-automatic extrinsic calibration method for multiple LiDARs and cameras. In this work, features are inserted in the field of view of the sensors with the help of spheres. These are detected with the help of the PointCloud Library (PCL)[2] segmentation and sphere fitting toolbox. The rigid body transformation is calculated using the Iterative Closest Point (ICP) algorithm[45].

2.3.2 Camera-LiDAR extrinsic calibration

While there is extensive work in this field, most of it focuses on the calibration of a single LiDAR and a camera. In this subsection we can classify the available methods into two:

1. The target category requires predefined and specific setups to ease the identification of shared features between the sensors. Under this class we can find notable mentions such as [3, 5, 44, 46–48], which take advantage of the purposely inserted

features, fixing the number of targets and optimizing the calibration algorithm under these constraints. The work presented by [3, 5, 44] require the construction of a setup consisting of several specific targets, and scatter them across the shared field of view of the camera and the LiDAR. This approach works well in laboratories and closed environments. However, its application is difficult in the field. Moreover, all the mentioned work, except in [3], are not open-sourced, reducing their impact and reachability.

2. The target-less methods focus on finding inherent features to the scene in both sensors. In this category [49] is a notable mention. It requires an initial rough position estimate between the sensors, provided by the user. In this position, it generates an image projection of the LiDAR reflectance values using the given camera projection matrix. It then slowly modifies the transformation to try to match the generated reflectivity image with the camera gray-scale image. To measure the error, it compares the camera brightness histogram, and the LiDAR’s reflectivity histogram. Since this method relies completely on the reflectivity values, it requires manual pre-calibration of the LiDAR unit[50], instead of using the parameters given by the manufacturer. This extra step involves the use of specific equipment, making this method difficult to deploy and test in practical situations.

Table 2.1 presents a feature summary comparison of the Camera-LiDAR calibration methods mentioned. From this we can compare characteristics such as integration, required target type, and sensing devices. After a quick analysis, we can identify the need of a method that works with different types of cameras and LiDARs, while maintaining the target-less property. Target-less methods offer the possibility to calibrate without the need of a special setup, or target. This feature allows users to reduce the time required to obtain the calibration parameters.

TABLE 2.1: Feature comparison among the analysed Camera-LiDAR algorithms.

Method	Open-source	Data Available	ROS Compatible	Target-Type	LiDAR Type	Camera Type
Geiger, et.al [51]	X	O	X	Chessboard	Rotating	Single
Naroditsky, et.al.[46]	X	X	X	Blackboard	Rotating	Single
Velas, et.al. [47]	O	X	O	Custom board	Rotating	Single
Weimin, et.al.[3]	O	O	X	Chessboard	Rotating	Single
Pandey, et.al.[49]	O	O	X	Target-less	Rotating	Omni

Having introduced the work related to our study, we will present the formal theory and mathematical background required to understand the following chapters in the next chapter.

Chapter 3

Background

3.1 Autonomous Driving

Manufacturers are developing Automated Driving Systems (ADS) to automate dynamic driving tasks (DDT). These developments hold the assurance to enhance vehicle safety and improve mobility. In many instances, this task is often considered as adding a cognitive layer to the existing vehicle platforms [52].

According to the Society of Automotive Engineers (SAE), transferring control from humans to automated vehicles is classified on a scale from 0 to 5, where 0 involves no automation and 5 means a complete full automatic control of the car at all times, under any roadway and environmental condition as explained in the J3016 standard[53]. Figure 3.1 clarifies the role of the human driver on each of these levels of automation.

The ADS architecture defined by the SAE envisions sensing, perception, navigation, control and safety modules. Each one of these work together to share data to achieve Dynamic Driving Tasks (DDT), and assist the system on the reponse of external events defined by the Object and Event Detection and Response (OEDR). Figure 3.2 introduces the functional architecture defined by the SAE on the J3016 standard [53]. This document additionally requires automated driving systems to have a localization system accurate to a minimum of 0.1 m at 95% confidence to be considered as a level 4 (high automation) driving system. This value is considered to be enough so the vehicle can precisely and safely execute maneuvers such as lane change or turn at intersections. Having in mind that this value is purposed for localization, and that localization (navigation) modules are triggered by the sensing

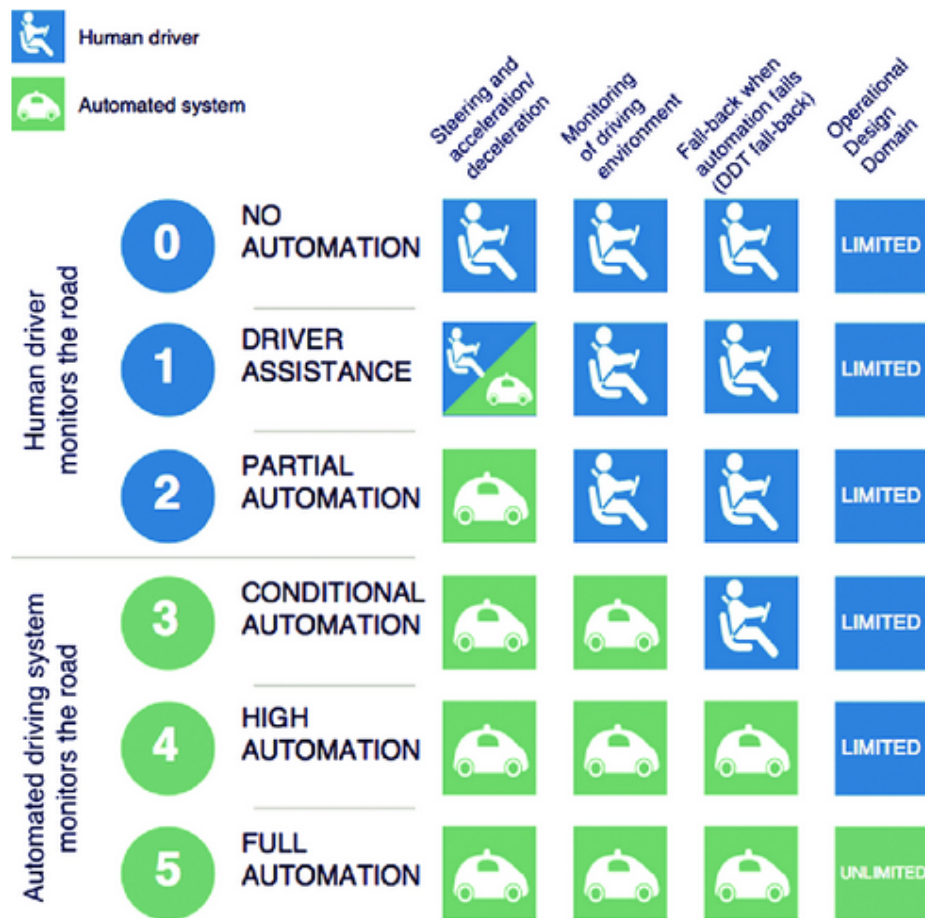


FIGURE 3.1: Levels of Automation According to the SAE J3016 Standard

modules in self-driving systems as shown in Figure 3.2. We can consider that sensing module need to have at least the same or better accuracy, specially knowing that localization, perception and planning modules introduce error [54]

The term for autonomous driving is usually misused to refer to either any level of automation, or just to levels higher than three. For such reason, it is important to differentiate the Advanced Driver-Assistance Systems (ADAS) which provide features such as keeping the ego vehicle within its lane, controlling the cruise speed, or breaking automatically in case of emergency.

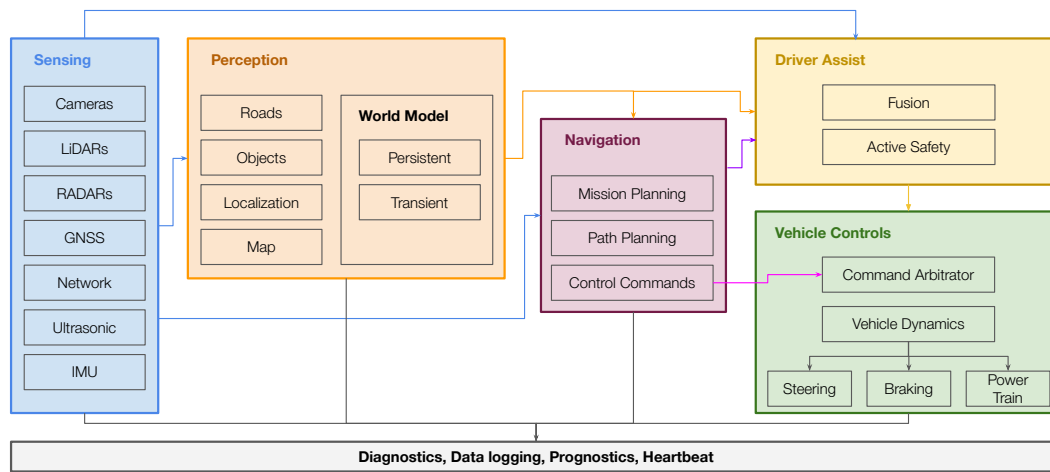


FIGURE 3.2: SAE International Autonomous Mode Functional Architecture Flow Diagram

3.1.1 Operational Design Domain (ODD)

For the above reason, the SAE also introduced the Operational Design Domain (ODD) concept [55, 56]. The ODD defines the limits in which the AVS is designed to operate, it is defined as "Operating conditions under which a given driving automation system, or feature thereof, is specifically designed to function, including, but not limited to, environmental, geographical, and time-of-day restrictions, and/or the requisite presence or absence of certain traffic or roadway characteristics." [53, 55]. The precise definition of the ODD is essential during the design phase, as the system requirements will change according to it. Case in point, a vehicle that should operate only in sunny weather in a limited area will have different requirements than one that should operate in foggy or snowy conditions.

3.1.2 Dynamic Driving Task (DDT)

The dynamic driving tasks define the operation and tactical functions required to operate a vehicle. The operational tasks describe the actions required to drive a car on a selected route and the all the actuator controls, such as steering or braking, while the tactical functions include actions to generate and follow a trajectory or keep the vehicle on a lane. It is important to note that the DDT definition excludes strategic functions such as scheduling or route planning. The American National Highway Traffic Safety Administration (NHTSA) further develops DDT and defines

the following working list [57] of tactical and operational maneuvers to ADS driving control:

- **Parking.** The vehicle comes to a complete stop within a vacant parking spot.
- **Maintain Speed.** The vehicle maintains a safe speed set through longitudinal control with acceptable following distances.
- **Car Following.** The vehicle identifies and follows a target vehicle at acceptable following distance while staying within a lane through longitudinal and lateral control.
- **Lane Centering.** The vehicle stays within a lane through lateral control.
- **Lane Switching/Overtaking.** The vehicle crosses lanes or overtakes an upcoming vehicle based on a projected path or hazard.
- **Enhancing Conspicuity.** The vehicle controls vehicle blinkers, headlights, horn, or other methods used to communicate with other drivers.
- **Obstacle Avoidance.** identifies and responds to on-road hazards, such as pedestrians, debris, animals, etc.
- **Low-Speed Merge.** The vehicle merges into a lane below about 70 kph, for example from an exit ramp, by identifying a vacant lane position and matching speed.
- **High-Speed Merge.** The vehicle merges into a lane above about 80 kph, for example from an exit ramp, by identifying a vacant lane position and matching speed.
- **Navigate On/Off-Ramps.** The vehicle drives on on/off-ramps, which are typically one-way, steeply curved, and banked road segments.
- **Right-of-Way Decisions.** The vehicle obeys directional restrictions; for example, one-way roads and actively managed lanes.
- **Follow Driving Laws.** The vehicle ADS obeys motor vehicle codes and local ordinances; for example, following distances, speed limits, etc. This may include driving norms that vary by region as well.

- **Navigate Roundabouts.** The vehicle determines right-of-way, enters, navigates, and exits a roundabout, and communicates with other road users as necessary.
- **Navigate Intersection.** The vehicle determines right-of-way, enters, navigates, and exits intersections, including signalized, stop signs, 4/3/2-ways, and communicates with other road users as necessary; may include left or right turns across oncoming traffic.
- **Navigate Crosswalk.** The vehicle determines right-of-way, enters, navigates, and exits pedestrian crosswalks, and communicates with other road users as necessary.
- **Navigate Work Zone.** The vehicle determines right-of-way and traffic patterns, enters, navigates and exits work zone, and communicates with other road users as necessary.
- **N-Point Turn.** The vehicle makes a heading adjustment that involves alternating between forward and reverse movement and adjusting steering to reposition the vehicle within a tight space.
- **U-Turn.** The vehicle determines right-of-way, initiates, and completes a U-turn, and communicates with other road users as necessary.
- **Route Planning.** The vehicle uses various information to define (and potentially update) a route network including road segments, turns, etc.

Finally, the DDT fall-back is defined as the response by the user or by an Automated Driving System (ADS) to either perform the DDT task or achieve a safety state after occurrence of a DDT performance-relevant system failure or upon leaving the designated ODD.

3.1.3 Object and Event Detection and Response (OEDR)

The vehicle while performing tactical maneuvers will interact with static and dynamic obstacles that may require the system to change its behavior. For this reason the SAE defines the Object and Event Detection and Response as to "the subtasks of

the DDT that include monitoring the driving environment (detecting, recognizing, and classifying objects and events and preparing to respond as needed) and executing an appropriate response to such objects and events (i.e., as needed to complete the DDT and/or DDT fallback" [53].

The system to achieve these detection and reponse requires the support from the sensing, perception, world modeling, and the navigation and planning systems as shown in Figure 3.2.

3.2 Advanced Driver-Assistance Systems (ADAS)

In the 1950s, the first Advanced Driver-Assistance Systems (ADAS) was introduced in the form of the anti-lock braking system (ABS) [58]. ADAS technologies installed on vehicles not only help to keep drivers and passengers safe, but also other drivers and pedestrians. Some warn the driver if the vehicle is at risk of an impending crash, while others are designed to overtake the driver action to avoid a crash. Every car manufacturer uses different terminology for the technologies and capabilities. However, the NHTSA categorizes these driver assistance technologies in the following groups:

- Collision Warning.
- Collision Intervention.
- Driving Control Assistance.
- Other Assistance.

Each of these driver assistance systems requires prompt and accurate sensing to enable prompt triggering of these assistance features. Cameras provide fast read-outs used in pedestrian, vehicle, and lane detection systems. Algorithms developed for images have been widely studied and provide high detection rates. However, camera performance is greatly reduced on low light scenarios, and cameras cannot provide accurate distance readings at mid or long ranges; LiDARs, on the other hand, provide accurate measurements in 3D space at short, mid, and long ranges. Additionally, LiDAR sensors work well regardless of lighting conditions.

Nevertheless, object detection for these sensors is currently not as developed as its camera counterparts; Using these two sensors in conjunction can help balance each other and bring additional benefits if both sensors are accurately calibrated. Case in point, lane detection could be performed on images captured by the camera, while accurate distance measurement could be back-projected to 3D space from the LiDAR readings. Section 5.3.3 presents a method on how to achieve this.

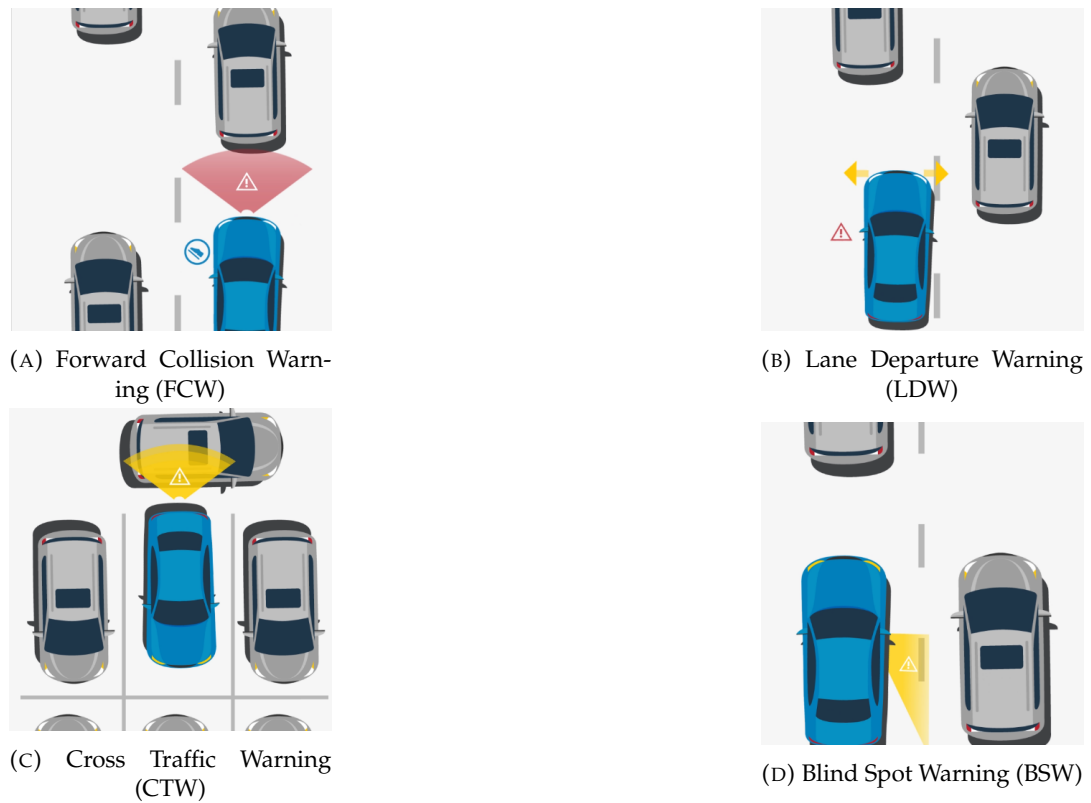
3.2.1 Collision Warning Systems

Collision Warning Systems use different sensors installed to notify the driver of a potential collision. These warnings are often shown to the driver as a visible notification on the board and an auditory alert so the driver can avoid impact. It is important to note that this kind of system only warns the driver and does not avoid a crash. The collision warning systems can be subcategorized as:

- **Forward Collision Warning.** A forward collision warning system monitors the vehicle's speed, the speed of the vehicle in front of it and the distance between the vehicles. If the distance between vehicle gets too close to, the system will warn the driver of an impending crash.
- **Lane Departure Warning.** A lane departure warning system monitors lane markings and alerts the driver when it detects that the vehicle is drifting out of the ego lane. However, this kind of system only provides a warning to the driver and does not take action to avoid a crash.
- **Cross Traffic Warning.** Cross traffic warning alerts the driver of potential collisions, while in moving moving in reverse, or when slowly merging where visibility might be limited.
- **Blind Spot Warning.** The blind spot warning alerts the driver of potential collision with an income vehicle when trying to make a lane change.

Additionally, Figure 3.3 illustrates some situations on which these types of warning systems are used.

FIGURE 3.3: ADAS Collision Warning Systems



3.2.2 Collision Intervention Systems

Similar to the Warning Intervention Systems, these systems use the sensors installed in the car to identify a potential collision. Additionally, these systems are equipped with vehicle controls to overtake the driver's action to avoid an imminent crash. These systems can be categorized as follows:

- **Automatic Emergency Braking (AEB).** Automatic emergency braking systems apply the vehicle's brakes automatically in time to avoid or mitigate an impending forward crash with another vehicle. Dynamic brake support (DBS) and crash imminent braking (CIB) are AEB systems that the SAE considers and the NHTSA to reduce moderate and less severe rear-end common crashes potentially. Additionally, if the system detects a crash and the driver brakes, but not hard enough to avoid the crash, DBS automatically supplements the driver's braking to avoid a crash. If the system detects a crash but the driver does not brake, CIB automatically applies the vehicle's brakes to slow or stop the car, avoiding the crash or reducing its severity [53].

- **Pedestrian Automatic Emergency Braking (PAEB).** A pedestrian automatic emergency braking system uses forward sensors to detect a pedestrian in the vehicle's path. The system will provide automatic braking if the driver has not acted to avoid crashes.
- **Rear Automatic Braking (RAB).** Rear automatic braking uses sensors like ultrasonic sensors and the rear camera to detect objects behind the vehicle. If the system detects a potential collision while in reverse, it automatically applies the brakes.
- **Blind Spot Intervention (BSI).** Blindspot intervention helps prevent a collision with a vehicle in the driver's blind spot. If the driver ignores the blind spot warning and starts to change to a lane where there is a vehicle, the system activates and automatically applies light braking pressure or provides steering input to guide the vehicle back into the original lane. The system monitors for vehicles in the driver's blind spot using rear-facing cameras or proximity sensors.

3.2.3 Driving Control Assistance Systems

Driver Control Assistance Systems help the driver to reduce fatigue while driving, applying automatic actions on steering, the drive train or on the brakes to maintain certain speed or a safe distance from the front vehicle. These systems are subcategorized as:

- **Adaptive Cruise Control (ACC).** Adaptive cruise control automatically adjusts the vehicle's speed to keep a safe distance between it and the vehicle in front of it.
- **Lane Centering Assistance (LCA).** Lane centering assistance employs a vision system designed to monitor the vehicle's lane position and automatically and continuously apply steering inputs needed to keep the vehicle centered within the ego-lane.

- **Lane Keeping Assistance (LKA).** Lane-keeping assistance helps prevent the vehicle from unintentionally drifting out of the ego-lane. The system uses information provided by lane departure warning sensors to determine whether the vehicle is about to move out of the ego-lane unintentionally. In these cases, the system activates and corrects the steering, braking, or accelerates one or more wheels, so the vehicle returns to the intended lane of travel.

3.2.4 Other Assistance Systems

Systems that do not fit on the categories presented on Sections 3.2.1 to 3.2.3 are considered as other systems. These that might help or provide enhanced conspicuity to the driver. Some examples of these systems are:

- **Automatic High Beams (AHB).** Automatic high beams automatically switch the vehicle's headlights between the lower and higher beams, based on lighting conditions and traffic, when an oncoming vehicle approaches. This technology, also known as semi-automatic beam switching headlamps, uses photometric sensors or onboard cameras to detect when to switch between high and low beams.
- **Backup Camera.** A backup camera, also known as a rearview video system, helps prevent backover crashes and protect vulnerable people such as children and the elderly. This camera provides an image of the area behind the vehicle, and helps the driver see behind the vehicle while in reverse. This system is not intended to replace mirrors or turn around to look.
- **Automatic Crash Notification.** An automatic crash notification system is designed to notify emergency responders of a crash and provide its location. In most cases, when the system detects that an airbag has deployed or that there has been a dramatic and sudden deceleration, the system automatically connects to an operator, who will then communicate with the driver. The operator can also collect basic information from the vehicle, without driver input, to provide to emergency responders to quickly locate the scene of the crash.

3.3 Sensing Systems

After introducing Autonomous Driving systems in the previous section, we want to focus on the sensing modules. Sensing modules are vital in new measurement techniques and instrumentation systems. Essential qualities of a suitable sensor system are high resolution, high reliability, low cost, appropriate output for a given input (good sensitivity), rapid response time, small random error in results, and small systematic error.

The sensing modules used in navigation robots and autonomous vehicles are often composed of an extensive array of multiple sensors such as LiDAR, camera, GNSS, ultrasonic, and RADAR. In this work, we focus on cameras and LiDARs. It is important to note that each sensor "sees" the world from its perspective. In other words, each sensor has its coordinate system and, therefore, its origin. In order to combine the data from multiple sensors, it is required first to obtain their relative position and the conversion required to associate the data among sensors to integrate and improve the reliability of the sensing system. Case in point, cameras produce a dense representation of the world, including color, texture, and shape. However, cameras cannot provide reliable depth information at longer distances. On the other hand, LiDARs capture sparse but highly accurate range information at short, middle, and often at long range regardless of the lighting conditions.

The simultaneous integration of data from multiple sensors is known as fusion, and it is used to overcome weaknesses in each sensor. The following sections will elaborate on how these sensors work, store data, and how to preprocess and obtain valuable data, as well as how to perform fusion when using cameras and LiDARs.

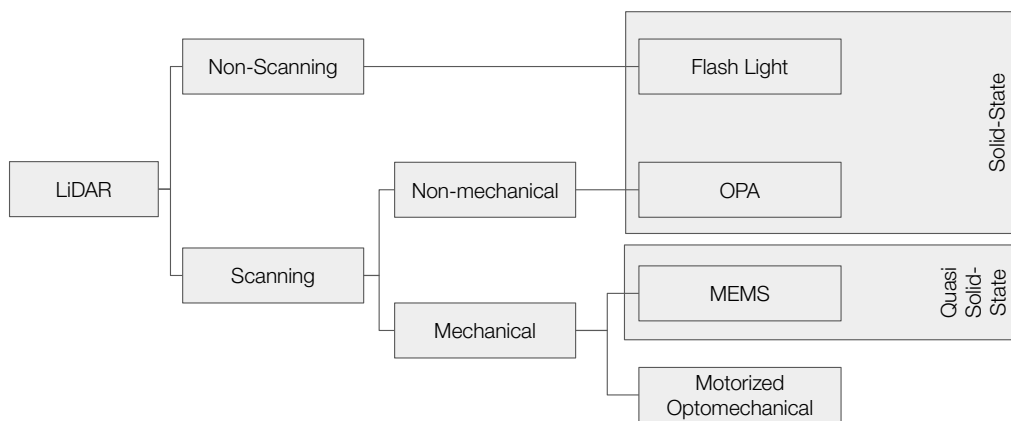
3.4 LiDAR

LiDAR, which stands for Light Detection and Ranging, is a remote sensing method that uses light in the form of a pulsed laser to measure ranges. LiDAR can also be used to make digital 3D representations of areas, due to differences in laser return times, and by varying laser wavelengths. LiDAR is also referred as 3D laser scanning, a special combination of 3D scanning and laser scanning. This sensor has

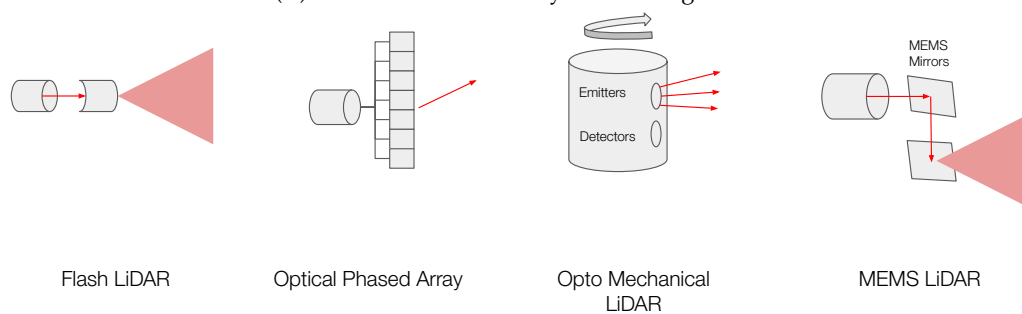
many applications such as surveying, geodesy, geomatics, archeology, among others. In the recent years, thanks to its high accuracy and cost being reduced year after year, the technology is also used in the sensing, perception, and localization modules of some ADS and ADAS systems.

LiDAR devices targeting robotics and automotive applications that use lasers with a wavelength between 800 to 1550 nanometers. These wavelengths are considered eye-safe since the human eye does not get strongly absorbed. Additionally to its laser wavelength, LiDARs are classified according to its scanning method as Figure 3.4 shows, while Figure 3.4 illustrates the scanning methods used internally by the sensors.

FIGURE 3.4: LiDAR classifications by scanning method.



(A) LiDAR classification by its scanning method



(B) LiDAR scanning methods

Motorized optomechanical LiDAR sensors use arrays of multiple infra-red(IR) lasers paired with IR detectors to measure distances to objects. Sensors manufacturers mount the device components within a compact and weather-resistant casing. The arrays of sensors and detectors spin rapidly, often generating a 3D-rich and

360-degree view of the surroundings. A digital signal processor (DSP) analyzes the waveform readings and calculates highly accurate range information. Additionally to the distance, LiDARs often provide the laser return intensity as read by the detector. The intensity information is helpful to identify light reflecting or absorbing surfaces such as retro-reflectors, lane markings, or vegetation. The number of laser arrays installed on 3D LiDARs devices ranges from 4 and 128 at writing. Additionally, sensor-equipped with 128 laser-detector pairs can return millions of points per second, creating a vibrant environments.

Micro-electromechanical systems (MEMS) LiDAR is called quasi-solid-state LiDAR as their moving parts move the laser beam in free space without moving any optical component. MEMS LiDARs have a modulated laser incident on a MEMS mirror that scans the laser beam to an object. A photodetector then picks up the echoed laser signal from the object, and the time of flight can be used to extract the distance. The mirrors inside a MEMS sensor can steer, modulate, and switch light, as well as control phase [59]. For this reason, compared to motorized optomechanical LiDARs, these are smaller, have faster readings, and their cost is usually lower.

Non-scanning LiDAR is also known as Flash LiDAR. The word "Flash" refers to the idea that the 2D FoV of interest is entirely illuminated by the laser source, like a camera with a flashlight, while an array of photodetectors at the image plane simultaneously picks up the time-of-flight (ToF) information of individual pixels in the 2D FoV [60]. Flash LiDARs use only solid-state components, which have the advantages of having no moving parts, being resistant to vibrations, enjoying a compact size, and a reduced price. However, the flood illumination required by each pixel of the photodetector array causes to receive only a tiny fraction of the returning laser power, leading to a low signal-to-noise ratio (SNR), which significantly limits the distance measurement range or demands very high laser power [61]. Moreover, the resolution of the detector array-based non-scanning LiDAR is constrained by the size and density of the detectors.

Optical phased arrays (OPAs) are a typical solid-state beam steering technology that enables the non-mechanical steering of optical beams [62]. OPAs benefit from high stabilization, random-access pointing, and good optical power handling capability. The laser power is split into an array of transmitters whose phases can be

individually controlled. By dynamically adjusting the relative phase shifts among the transmitters, a laser beam can be formed and steered [63]. The OPAs with several phase modulators have been reported using liquid crystals, MEMS, or silicon photonics based on different approaches.

Regardless of the technology LiDARs used to scan and generate the 3D representation of its surroundings, all the LiDARs sensors produce a point cloud, which represents the 3-dimensional position of each scan. Additionally to the position often the sensor also includes information such as the laser return intensity, noise, echo, and the time-of-flight.

3.4.1 Point Clouds

A point cloud is a set of discrete points in space. Each point position has its set of Cartesian coordinates (X, Y, Z). Point clouds are generally produced by 3D scanners or photogrammetry techniques, which measure discrete points on the object's external surfaces. Point clouds are ideal for representing LiDAR scans because these sensors create discrete and sparse readings of an object's surface. Moreover, point clouds additionally to the reading coordinates, these data structures can have extra custom fields to store information such as intensity, range, azimuth and specify the data format. Thus, these can be accommodated to fit the data given by different sensors and manufacturers.

There are multiple storage formats for point cloud, such as the Polygon File Format (PLY), the object file format developed by Wavefront (OBJ), the Recap Scan format developed by Autodesk (RCS), and some others. Nevertheless, these formats suffer from shortcomings, as they were created for a different purpose and before current laser scanners were widely used [64]. For these reasons, the Point Cloud Library (PCL) implemented a new format, the Point Cloud Data (PCD) format. This format aims to fast data processing and a dynamic storage container that can handle different data types. Additionally, this format is integrated within the PCL library, allowing fast processing of point clouds.

The PCD format stores the data in three different storage types: In *ASCII format*, a human-readable format. It stores each point in a new line and each field separated by a space. Points with invalid values are represented with the "nan" string; In *Binary*

Format, a complete memory dump from the PointCloud array of the C++ object. This format allows the fastest read and writes access to the data; Finally, the PCD format can also store the compressed data binary data. This format compresses the binary format using Lehmann's LZF algorithm [65]. PCL selected this method, not due to its size reduction capabilities but its compression-decompression speeds [64].

3.5 Camera

Cameras have become ubiquitous thanks to their low cost, high quality, and ability to represent the world with dense and feature-rich images. The images created by these devices resemble our vision, depicting objects located at different distances with different apparent dimensions. Mathematically speaking, projection geometry is the way cameras are commonly modeled. Projective geometry alters the angles, distance, and ratios of distances when projecting objects. The only property maintained is straightness when considering lens distortion. The camera can be modeled using homogeneous coordinates. A point in Euclidean 2-space is represented by an ordered pair of real numbers (x, y) , and a coordinate is added. We now take the critical conceptual step of asking why the last coordinate must be 1. After all, the other two coordinates are not so constrained. Formally, points are represented by equivalence classes of coordinate triples. Two triples are equivalent when they differ by a common multiple. These are called the homogeneous coordinates of the point.

3.5.1 The Camer Pinhole Model

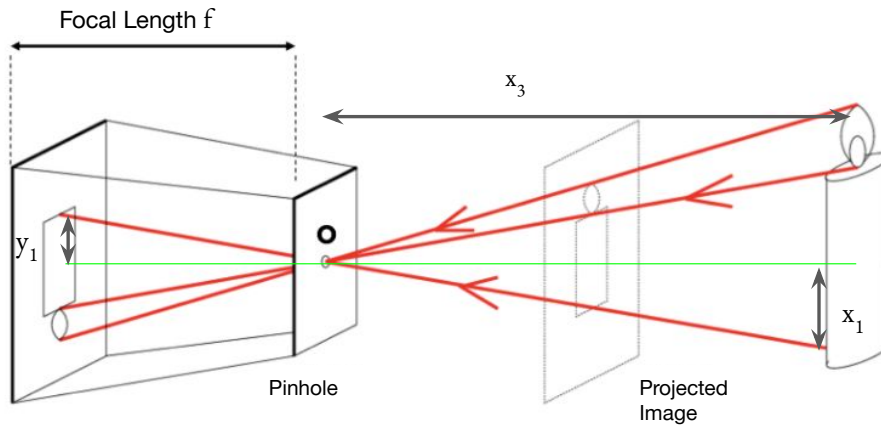
The pinhole camera model describes the mathematical relationship between the coordinates of a point in three-dimensional space and its projection onto the image plane of an ideal pinhole camera. The camera aperture is represented as a point, and no lenses are used to focus light. The model does not include, for example, geometric distortions or blurring of unfocused objects caused by lenses and finite-sized apertures. This model uses homogeneous coordinates to enable the projection of the 3-Dimensional world to a plane.

The mapping from the coordinates of a 2d in euclidean space point $P = (x_1, x_2)$ to the 2D image coordinates of the point's projection onto the image plane $p =$

(y_1, y_2) , according to the pinhole camera model, illustrated in Figure 3.5 and given by

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \frac{f}{x_3} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

FIGURE 3.5: Diagram of Camera Pinhole Model



In computer vision, the projection's origin is located not in the center as presented in the model above but at the left-topmost coordinate of the projection plane. For this reason, the longitudinal and transversal offset is known as the Center Point (C_x, C_y) . Additionally, the distance between the projective plane and the focusing point is labeled as the focal length (f_x, f_y) . The mapping from 3D coordinates of points in space to 2D image coordinates is encoded in a matrix known as the Camera Matrix, usually denoted by K , and arranged as:

$$K = \begin{bmatrix} f_x & 0 & C_x \\ 0 & f_y & C_y \\ 0 & 0 & 1 \end{bmatrix}$$

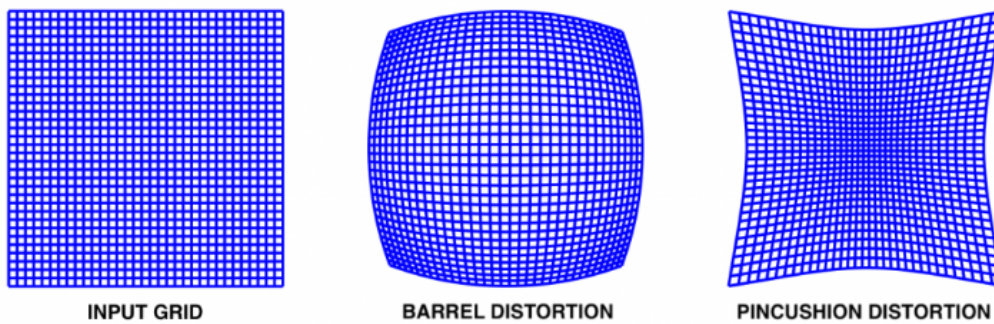
This implies that the left and right hand sides are equal up to a non-zero scalar multiplication. A consequence of this relation is that also K can be seen as an element of a projective space; two camera matrices are equivalent if they are equal up to a scalar multiplication. This description of the pinhole camera mapping, as a linear transformation K instead of as a fraction of two linear expressions, makes it possible

to simplify many derivations of relations between 3D and 2D coordinates.

3.5.2 Camera Plumb Bob Model

The Pinhole model can represent an ideal camera with perfect lens and no distortion. Nevertheless, cameras and lenses inherently distort the light, and therefore the image. Radial and tangential distortions are modeled using the "Plumb Bob" introduced by Brown [66]. Radial distortion bends straight lines into circular arcs, violating the main invariance preserved in the pinhole camera model, in which straight lines in the world map to straight lines in the image plane. Radial distortions are predominant in current projective lenses, and it may appear as a "barrel" or "pincushion distortion" as illustrated in Figure 3.6.

FIGURE 3.6: Types of Radial Distortion



The radial distortions are represented as:

$$x_{radial_distorted} = x(1 + k_1r^2 + k_2r^4 + k_3r^6)$$

$$y_{radial_distorted} = y(1 + k_1r^2 + k_2r^4 + k_3r^6)$$

Tangential distortions occur when the lens and the image plane are not parallel, causing some areas to look nearer than expected. Tangential distortions are represented by:

$$x_{tangential_distorted} = x + [2p_1xy + p_2(r^2 + 2x^2)]$$

$$y_{\text{tangential_distorted}} = y + [p_1(r^2 + 2y^2) + 2p_2xy]$$

Merging both distortions we get:

$$x_{\text{distorted}} = x + (1 + k_1r^2 + k_2r^4 + k_3r^6) + [2p_1xy + p_2(r^2 + 2x^2)]$$

$$y_{\text{distorted}} = y + (1 + k_1r^2 + k_2r^4 + k_3r^6) + [p_1(r^2 + 2y^2) + 2p_2xy]$$

In both, the radial and tangential distortions r^2 is calculated as:

$$r^2 = x_{\text{distorted}}^2 + y_{\text{distorted}}^2$$

k_1, k_2, k_3 are the radial distortion coefficients, while p_1, p_2 are the tangential distortion coefficients. Higher order radial coefficients can be added. Nevertheless, it has been shown that higher-degree polynomials does not improve much the distortion model [67].

All the camera parameters ($C_x, C_y, f_x, f_y, k_1, k_2, k_3, p_1, p_2$) are obtained through a method known as camera calibration, which will be further analysed in Chapter 4.

3.5.3 Images

Digital cameras store the projections they generate in digital images, also known as raster images. These images are composed of a finite number of picture elements (pixels), each with finite, discrete quantities and a finite size defined by the camera sensor. Each pixel represents the intensity or gray level that the digital sensor read at the time of capture. Additionally, these are stored as a rectangular matrix or a grid of pixels. The dimension of the matrix is also known as image resolution.

The fundamental strategy underlying the images is the tessellation of a plane, into a two dimensional array of squares. A single numeric value is then stored for each pixel. For most images, this value is a visible color, but other measurements are possible, even numeric codes for qualitative categories. Each raster grid has a specified pixel format, the data type for each number. Common pixel formats are binary, gray scale, palettized, and full color, where color depth determines the fidelity of the colors represented and color space determines the range of color coverage (which is often less than the full range of human color vision). Most modern color raster

formats represent color using 24 bits (over 16 million distinct colors), 8 each (0-255) for red, green, and blue.

3.6 Sensor Fusion

Sensor fusion is the process of merging data from multiple sensors to reduce the amount of uncertainty involved in a robot navigation motion or perception task. The three fundamental ways of combining sensor data are [68]:

- Redundant sensors: All sensors provide the same type of reading about the world.
- Complementary sensors: The sensors gather independent (disjoint) data.
- Coordinated sensors: The sensors collect information about the world sequentially.

Moreover, these sensor can communicate using the following schemes:

- Decentralized: No communication exists between the sensor nodes.
- Centralized: All sensors provide measurements to a central node.
- Distributed: The nodes interchange information at a predefined communication rate.

Regardless of the type of combination or synchronization method, each sensor stores perceives the world in its own format, and coordinate system. For the case of a single camera and a single LiDAR, as detailed in Sections 3.4 and 3.5 we need to obtain additionally to the camera projection matrix, the transformation matrix between both sensors in the form:

$$\mathbf{p}_{\text{cam}} = \mathbf{P}_{\text{lidar,cam}} \cdot \mathbf{R} \cdot \mathbf{t} \cdot \mathbf{p}_{\text{lidar}}$$

where p_{lidar} represents the 3D point in euclidean coordinates in the LiDAR coordinate frame; R and t represent the rotation and translation matrices required to convert the LiDAR coordinate system to the camera one; $P_{\text{lidar,cam}}$ represents the

camera projection matrix after distortion correction. Leaving p_{cam} containing the projected LiDAR 3D points onto the image. In Chapters 4 and 5 we will present the methods we developed to obtain these transformation between coordinate systems and projective spaces.

Chapter 4

Automatic Single-Shot Camera Calibration

4.1 Problem

Navigation robots, such as autonomous vehicles, require a highly accurate representation of their surroundings to navigate and reach their target safely. Sensors such as cameras, radars, and LiDARs (Light Detection and Ranging) are commonly used to provide rich perception information. Each of these sensors can complement each other to supply reliable and accurate data. For example, cameras produce a dense representation of the world, including color, texture, and shape. However, cameras cannot provide reliable depth information at longer distances. On the other hand, LiDARs capture dense and highly accurate range information at short, middle, and often at long range regardless of the lighting conditions.

The simultaneous integration of data from multiple sensors is known as fusion, and it is used to overcome weaknesses in each individual sensor. State-of-the-art perception algorithms utilize fused data inside deep neural networks to improve detection accuracy. For example, some of these networks require the image, the point cloud data, and accurate camera intrinsic and camera-lidar extrinsic parameters to enable training and inference [39–42]. Another common application that requires precise calibration is camera-based localization, also known as visual SLAM (Simultaneous Localization and Mapping) [69, 70]. On the other hand, applications that do not require fusion and only operate on images might not be significantly affected by small errors in the intrinsic camera parameters. Recent advances in deep

learning [71, 72] apply data augmentation techniques to increase resilience to image distortions. Regardless of whether camera data is used independently or as part of a fusion methodology, any application involving 3D geometry will require accurate and careful sensor calibration. Fundamental to this is how a camera is modeled in terms of its intrinsic parameters.

Cameras have become ubiquitous thanks to their low cost, high quality, and ability to represent the world with dense and feature-rich images. The images created by these devices resemble our own vision, depicting objects located at different distances with different apparent dimensions. The mathematical model commonly used to project the three-dimensional world is the pin-hole camera model. In addition, the plumb-bob model, also known as the Brown-Conrady model, represents the distortion caused by the lens attached to the camera[1]. Model parameters can be estimated using a method known as camera calibration (also referred to as geometric camera calibration or camera re-sectioning). This method requires capturing images while moving either the camera itself or a calibration target, with identifiable features and known dimensions, aiming to cover the entire camera's field of view. The targets used in this process depend on the calibration algorithm. Methods such as the one presented by Zhang [73] use one-dimensional targets in the form of a stick with beads attached to it separated by a known distance. On the other side, methods using three-dimensional targets employ two or three planes orthogonal to each other or a plane undergoing a pure translation. Nevertheless, the calibration methods used in computer vision applications traditionally use two-dimensional targets [74]. A large flat checkerboard is often used for this purpose. Correspondences between feature points on the target among all of the frames are determined in order to calculate the intrinsic parameters. This process can be tedious, especially on robots or vehicles featuring large arrays of cameras. While there are instructions on how to perform the data capture procedure, such as taking multiple images while keeping the focus and focal length fixed, there are no clear instructions on setting the checkerboard pose to obtain accurate parameters. After finishing the data acquisition and estimating the parameters, their validity is not apparent until they are applied to project 3D points. The metric commonly used to evaluate the accuracy of the parameters is the re-projection error. It involves calculating the error between

the detected and the corresponding re-projected feature points. However, this metric uses the same points that were used to estimate the parameters, which reduces its reliability.

Given the importance of accurate camera intrinsic parameters for 3D applications, we aim to: 1) define clear guidelines to calibrate monocular cameras accurately; and 2) create a method that allows us to calibrate accurately using a single-shot having a predefined setting with multiple checkerboards. To accomplish this, we employ a realistic simulator to generate, calibrate and evaluate hundreds of combinations to obtain the minimum number of checkerboards, their positions, and rotations that would provide accurate intrinsic parameters. We additionally evaluate the corner detection accuracy, which is an integral part of the calibration process, and often overlooked in other work. To overcome the weakness of the re-projection error metric, we intentionally project virtual 3D points, labeled as Control Points, on the camera field of view edges. We then use these Control Points to verify the distortion correction qualitatively and select the best checkerboard arrangements based on score combinations. Finally, we test the top-performing checkerboard poses to calibrate real cameras and project the point cloud generated by a 3D LiDAR sensor using the estimated intrinsic parameters to validate our calibration guidelines findings. To the best of our knowledge, this is the first work to carry out an in-depth study using simulations to provide an optimized set of guidelines for one-shot calibration.

4.2 Previous work

There exists a considerable amount of work dedicated to developing techniques for estimation of camera intrinsic parameters. Notable mentions include the work by Zhang [31], Kannala and Brandt [32], and Heikkila and Sliven [33]. Despite being published more than twenty years ago, these methods provide consistent and reliable results. Moreover, the widely-used open-source computer vision library OpenCV [34] and the proprietary Matlab [35] platform use these methods in

their camera calibration toolboxes due to their proven accuracy. More recent approaches use deep learning methods to estimate the camera intrinsic parameters using neural networks trained on large datasets of images with known intrinsic parameters [36][37]. These methods are convenient since they do not require any targets or calibration datasets. Nevertheless, these approaches are still far from matching the accuracy achieved by target-based techniques.

Zhang [31] used synthetic data while testing his calibration method to evaluate resilience against noise. He obtained good results with as few as three checkerboards, without aiming to use the parameters in 3D applications. However, he only used the checkerboard corners to measure the error, which might result in over-fitted parameters. Moreover, he did not consider the error introduced by the corner detection phase.

The work dedicated to the extrinsic calibration of LiDARs, radars, and cameras explicitly states that accurate intrinsic camera parameters are required [3, 6–8, 38–42, 75]. These methods find shared features between the 2D perspective space on the images generated by the camera and the 3D Euclidean space employed by radars and LiDARs. The shared features are then input to an optimizer to estimate the extrinsic parameters (relative position \mathbf{t} , and rotation \mathbf{R}), which attempts to reduce the projection error of the 3D features ($\mathbf{p}_{\text{lidar}}$) while using the given camera intrinsic parameters ($\mathbf{P}_{\text{lidar,cam}}$) in the form $\mathbf{p}_{\text{cam}} = \mathbf{P}_{\text{lidar,cam}} \cdot \mathbf{R} \cdot \mathbf{t} \cdot \mathbf{p}_{\text{lidar}}$. This equation illustrates the importance of having high-quality camera intrinsic parameters in order to obtain accurate sensor extrinsic parameters.

There are a limited number of published studies about verifying estimated camera intrinsic parameters for use in 3D applications. Basso et al. [4] stressed the requirement of accurate intrinsic parameters for 3D applications such as SLAM, and introduced a method for the intrinsic and extrinsic calibration optimizer for short-range time-of-flight (ToF) sensors such as the Microsoft Kinect. Geiger et al. [5] presented a single-shot calibration method for short and long-range LiDARs and cameras. Their approach uses multiple checkerboards in a single frame to accelerate and simplify the data acquisition. However, they did not analyze or demonstrate why these positions are optimal. We extend this research direction to create clear guidelines on how to achieve consistent and accurate intrinsic parameters and introduce

validation metrics to verify them.

4.3 Method

we decided to use a real camera as our baseline. We therefore first needed to calibrate it, obtain the intrinsic parameters, and verify that these are appropriate for 3D applications. To calibrate our camera, we decided to use planar checkerboards with checkered patterns since they are widely available, are low cost, and have established corner detection methods that provide high accuracy [5], [76]. To make the simulation closer to reality, we decided to model and simulate the checkerboard used to calibrate our real camera.

Additionally, to calibrate our real baseline camera and the virtual cameras generated by the simulator, we re-implemented the corner detection method presented by Geiger et al. [5], based on Ha's algorithm [77]. This is due to its simplicity and proven advantage in noisy and blurry environments when compared to the Harris [78], and Shi-Tomasi [79] corner detectors included in OpenCV.

4.3.1 Baseline Calibration

We intrinsically calibrated a 5.4 MP Lucid Vision Labs machine vision camera (TRI054S) paired with an 8 mm focal length Fujinon lens. We used an 800 mm by 600 mm planar checkerboard printed on 4 mm thick aluminum, with an eight by six pattern, and a 100 mm square size. A total of 292 checkerboard poses were used to generate a baseline. We then used the OpenCV [34] camera calibration toolkit based on Zhang's method [31], and MATLAB's [80] Adaptive Thresholding to obtain the camera intrinsic parameters (principal point, focal length, axis skew), three radial distortion coefficients, and two tangential distortion coefficients. We projected the point cloud generated by a 3D LiDAR sensor, a Hesai Pandar 64, extrinsically calibrated using the method by Zhou et al. [7] to validate that these parameters are accurate for 3D applications.

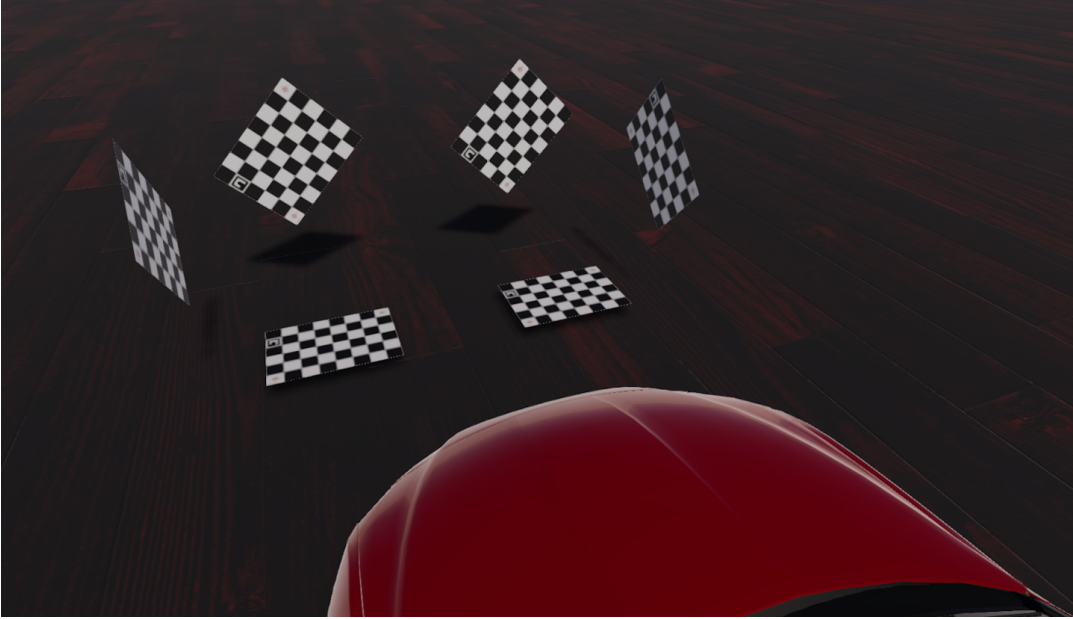


FIGURE 4.1: Our modeled checkerboard simulated in the LGSVL.

4.4 Simulation

Having a baseline defined by a real camera calibrated with a checkerboard, we created a 3D model with the help of Blender [81], based on the printed planar checkerboard mentioned in Section 4.3.1. We then converted the checkerboard model for use within the LGSVL (LG Silicon Valley) simulator [82] as a controllable object.

The LGSVL Simulator allows the creation of virtual locations, weather scenarios, obstacles, and one ego-vehicle. Any number of sensors can be attached to the ego-vehicle, such as cameras, LiDARs, and GNSS. With the help of the simulator API, we generated an empty scene with an ego vehicle, one camera with the parameters introduced in Section 4.3.1. We then dynamically generated multiple instances of our checkerboard controllable object as illustrated in Figure 4.1. The camera simulated by the LGSVL simulator rendered images using the plumb-bob model with the given intrinsic parameters. The simulator API allowed us to save these renders as image files.

4.4.1 Checkerboard Coordinate System

We defined the checkerboard coordinate system to be right-handed. The Z-axis is normal to the checkerboard plane, the X-axis is parallel to the checkerboard's short

side, and the Y-axis is parallel to the long side of the checkerboard. The origin is located at the center of the checkerboard.

4.4.2 Simulator Coordinate System

The simulator coordinate system is left-handed. The X-axis faces to the right, the Y-axis points upwards, and the Z-axis is normal to the camera plane and faces forward.

4.5 Checkerboard Corner Detector Evaluation

Before starting to simulate multiple checkerboards, we decided to initially evaluate the limits of our re-implemented version of the corner detector (based on Geiger et al. [5], and Guiming and Jidong [76]) inside the simulator. We used this information to decide the pose and distance intervals that will have a higher probability of detecting the checkerboard corners, and therefore produce more accurate intrinsic parameters. This step is of utmost importance since the detected checkerboard corners are the inputs for the optimizer. If they contain significant errors, the estimated parameters will be inaccurate.

4.5.1 Corner Detector Metrics

To experimentally obtain the intervals at which the checkerboard corner detector will fail, we located the checkerboard at the center of the camera's field of view in the simulated world. We rotated the checkerboard with respect to its X (roll, α), Y (pitch, β), and Z (yaw, γ) axes on a $[-90^\circ, 90^\circ]$ interval with one-degree steps for the roll and pitch and five degrees steps for the yaw; to obtain the maximum distance at which the detector would fail, we moved the checkerboard away from the camera in 1 m steps, until the detector failed. Additionally, to evaluate the corner detector, we defined the following variables and statistics:

- *Ground Truth 3D corners* \mathbf{C}_{3D_t} in camera space define the checkerboard plane normal.
- *Corner RMSE* is calculated between the true 2D corners \mathbf{C}_{2D_t} in distorted images generated by the simulator, and the corners computed when running the

corner detection C_c . The true 2D corners are obtained by projecting the true 3D corners as: $\mathbf{P} \cdot \mathbf{K} \cdot \mathbf{C}_{3Dt}$, where \mathbf{P} and \mathbf{K} , are the projection and intrinsic calibration matrices respectively.

- *Inner Checkerboard Area* is calculated by obtaining the area of the two triangles formed by the corners in the checkerboard. Area calculation would be exact in an undistorted image but is not precise in a distorted one. For this reason, we use two triangles to estimate the area since we propose that this produces better results than using a parallelogram.
- *Checkerboard-Image Plane Angle* is defined as the angle between the checkerboard plane normal (as defined by the corners) and the image plane normal (camera z-axis).

4.5.2 Experiments

Rolling Experiment. For this experiment, we positioned the origin of the checkerboard at the same height as the camera origin, and set the checkerboard 4 m away along the Z-axis, then varied the roll angle between 0 and 90 degrees in one-degree steps.

Pitching Experiment. In this experiment, we aligned the checkerboard and camera origins, placed the checkerboard 4 m away from the camera, and varied the pitch rotation in the $[0^\circ, 90^\circ]$ interval with one-degree steps.

Yaw Experiment. For this experiment, we aligned the checkerboard and camera origins and positioned the checkerboard to be 4 m away from the camera on the Z-axis. We then rotated the checkerboard w.r.t the checkerboard's Z-axis between $[-90^\circ, 90^\circ]$ in five degrees steps.

Simultaneous Rolling and Pitching Experiment. In this experiment, we examined the effects of simultaneously varying the pitch and roll on the corner detector. We set the checkerboard origin height to match the camera's and placed the checkerboard 4 m away from the camera along the Z-axis. Additionally, we fixed the yaw rotation to 53.14 degrees. This angle allowed us to align the checkerboard longer diagonal to the vertical axis; this condition helped us simulate the same circumstances that we

would use in an actual camera-3D sensor extrinsic calibration [7]. We then simultaneously varied the roll (α) and pitch (β) over the intervals of $[-80^\circ, 80^\circ]$ and $[-60^\circ, 60^\circ]$ respectively.

Range Experiment. In this experiment, we set the camera origin and the checkerboard origin to have the same height and initially separated them by 4 m along the Z-axis. To verify the maximum detection distance of the checkerboard corner detector, we moved the checkerboard away from the camera in 1 m steps until it failed. Additionally, once we obtained the checkerboard corner detector failure range, we repeated the experiment focusing on the working area with 0.5 m steps to understand better the detector's performance.

4.5.3 Results

Figure 4.2 and Figure 4.3 show the results of the corner detector experiments in the simulator. From these, we can draw the following guidelines regarding the corner detector:

- We found that the corner detector peak performance w.r.t roll rotation between the camera plane normal and the checkerboard normal is between 0 and 60 degrees, as we present in Figure 4.2a. However, rotations below 70 degrees also obtain reliable corner detection performance. On the other hand, we can see that the performance quickly decreases at angles larger than 70 degrees and the detector completely fails for angles larger than 78 degrees.
- We observed in Figure 4.2b that the corner detector performs best between 20 and 60 degrees when varying the pitch angle between the camera plane normal and the checkerboard normal. The performance degrades at angles larger than 60 degrees, until it cannot detect any corners at all after 78 degrees.
- From Figure 4.2c, we can appreciate that the corner detector performs best between 20 and 60 degrees when simultaneously varying the pitch and roll between the camera plane normal and the checkerboard normal. Similarly, we see a reduction in accuracy when the rotations surpass 78 degrees.

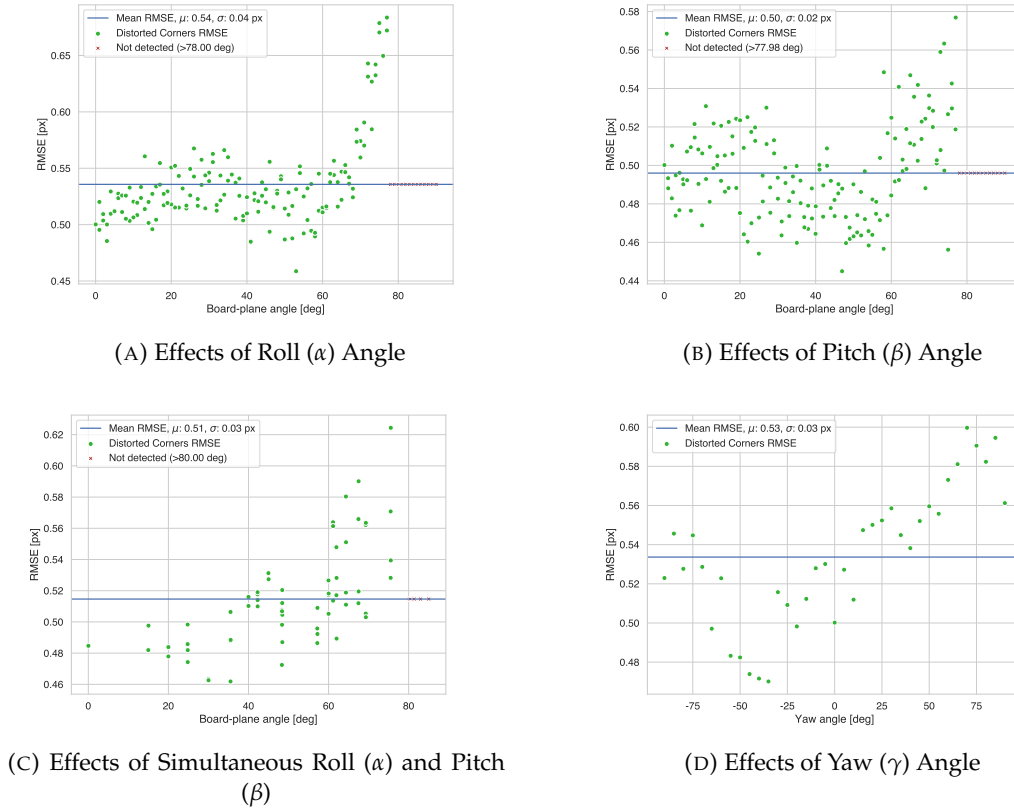


FIGURE 4.2: Effects of roll (α), pitch (β) and yaw (γ) rotations on the checkerboard corner detector.

- From Figure 4.3a we can see that the corner detector can detect corners reliably up to 35 m. However, in Figure 4.3b we can observe a tendency in the corner detector RMSE that leads us to the conclusion that the distance between the camera and checkerboard considerably affects its performance. For this reason, we suggest that the checkerboard's visible inner area should be at least $20,000 \text{ px}^2$. This value for the area is resolution independent, so we propose it as a guideline for perspective cameras and lenses that produce a different field of view.

4.6 Simulated Calibration Experiments

With the knowledge obtained about the impacts of distance and rotation on the corner detector, we investigated the checkerboard positions in the image frame and their influence on the camera intrinsic parameters. To achieve this, we needed to define the metrics to assist evaluation for each set of checkerboard poses and positions.

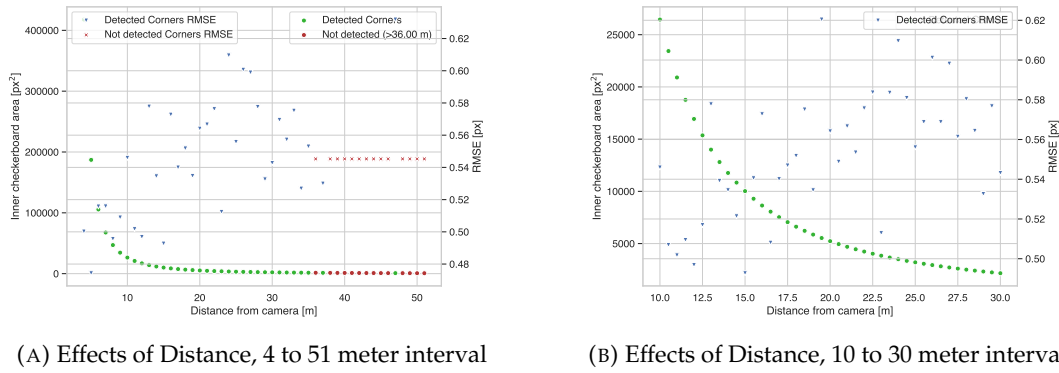


FIGURE 4.3: Effects of distance between the checkerboard and the camera on the checkerboard corner detector.

4.6.1 Checkerboard Pose Metrics

To verify if a set of checkerboard poses provides a better estimation of the camera's intrinsic parameters, we calculated the Root Mean Square Error (RMSE) between the ground truth parameters and the those estimated from the corners of the checkerboards detected on the image using OpenCV [34]. We measured the following parameters:

- Focal length (f_x, f_y).
- Center point (c_x, c_y).
- Distortion coefficients: three radial (k_1, k_2, k_3), and two tangential (p_1, p_2).

In addition to the intrinsic parameters, we also obtained:

- The RMSE between the ground truth corner positions and the projected corner points using the estimated intrinsic parameters.
- The checkerboard corner re-projection error, which is the distance between the detected corners in a calibration image, and the corresponding 3D corner points projected into the same image.
- The Control Points re-projection error, which is the distance between the projections of a 3D Control Point when using the estimated and the ground truth intrinsic parameters. In Section 4.6.2, we introduce and describe the "Control Points" in more detail.

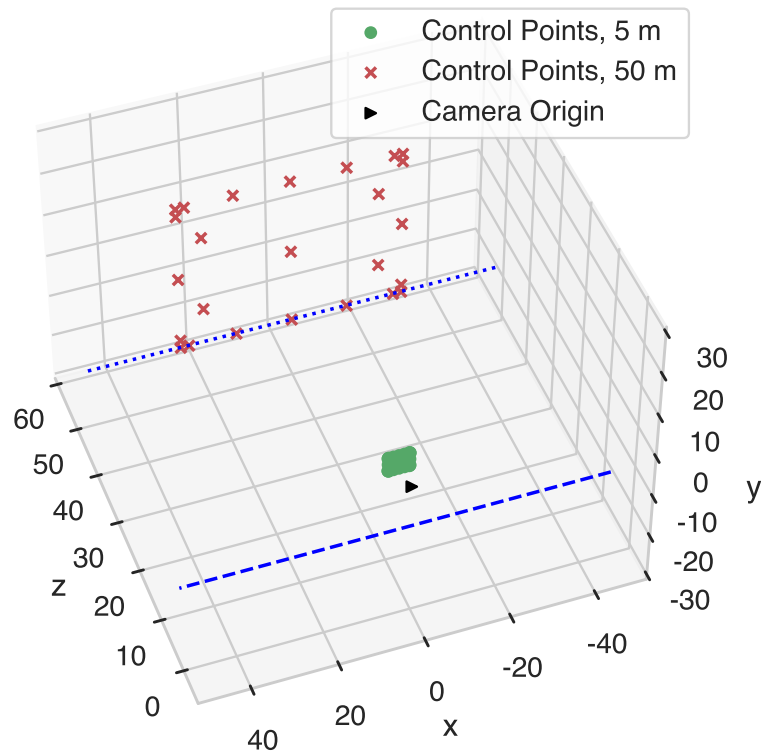


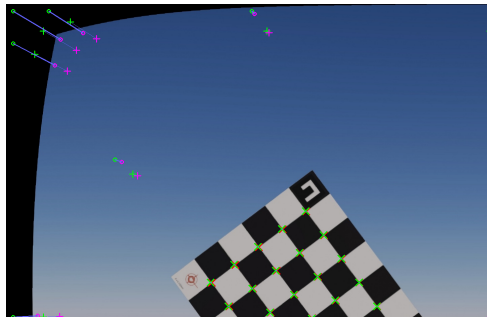
FIGURE 4.4: Control Points systematically located inside the camera frustum.

4.6.2 Control Points

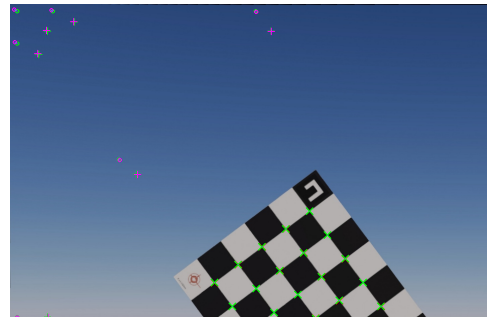
The re-projection error is a metric used to quantify the distance between the projection estimate of a 3D point and its actual projection. This metric is widely used to estimate the performance of the camera intrinsic parameters, measuring the detected corners and their 3D estimated counterparts. However, since these points only contain areas belonging to the checkerboard, this metric is unreliable on other parts of the image. For this reason, we decided to purposely insert 3D virtual points in the simulation to assist in measuring the performance on the edges of the image.

The Control Points are 3D virtual points strategically positioned in the camera frustum. They target the areas of the camera field of view which are prone to project points incorrectly due to lens distortions. We located these points systematically as shown in Figure 4.4 at 5 m and 50 m from the camera origin. We defined those two distances to test the performance at close and long-range.

Figure 4.5 shows two simulated checkerboards with an almost identical re-projection



(A) Control Points showing a significant projection error, even when the checkerboard corners are projected with low error.



(B) Control Points and checkerboard corners showing a reduced projection error.

FIGURE 4.5: Control points as an auxiliary metric. Green marks represent the "Control points" projected with the ground truth intrinsic parameters, while purple marks represent the projection of the "Control Points" using the estimated intrinsic parameters. Both (a) and (b) have the same subpixel checkerboard corner re-projection error value, and corners in the checkerboard are correctly re-projected in both cases. However, the estimated intrinsic parameters have a large error in (a), correlating to Control Point re-projection error.

error value when using only the checkerboard corners. The ground truth checkerboard corners are drawn with a red crosshair, while the re-projected corners are drawn with a green crosshair. The corner points for both checkerboards are re-projected after un-distortion with sub-pixel accuracy. However, when using the Control Points to calculate the re-projection error in Figure 4.5a, the error metric increases considerably. The checkerboard in Figure 4.5b, on the other hand, has lower control point re-projection error due to the intrinsic parameters being closer to the ground truth. We use this new metric to help us determine whether a checkerboard pose is adequate or not.

4.6.3 Dual Checkerboard Calibration

In this series of experiments, we evaluated the effect of varying the position and rotation of each checkerboard. This chapter aims to formulate guidelines that will help to narrow down the number of combinations required when increasing the number of checkerboards. To do this, we must determine the poses that result in a minimized error.

Having explained the proposed method for corner detector evaluation and defined the required metrics, we also aim to answer the following questions for multiple checkerboards:

- How do the checkerboard rotation angles affect the calibration results?
- How do the vertical and horizontal checkerboard positions impact the intrinsic parameter estimation?
- How does the distance between the camera and the checkerboard influence the intrinsic parameters?
- What is the minimum number of checkerboards or poses required to accurately calibrate intrinsically a camera?

Dual Checkerboard Rotation Experiments

In these experiments, we investigate the checkerboard pose performance when varying the roll (α) and pitch (β) angles in different rotational combination patterns: individual, simultaneous, symmetric, and asymmetric. For all these experiments, we fixed the checkerboard yaw angle to 53.14 degrees because this aligns the checkerboard longer diagonal to the vertical-axis, which allowed us to simplify the camera-LiDAR calibration [7]. This condition was required to simulate the same settings to use in the real-world scenario.

We used the roll and pitch rotation intervals we obtained in Section 4.5 for the following experiments, since we previously determined that these would provide accurate checkerboard corner detections.

We prepend all of the experiments in this section by the letter **A**, denoting the angle variation. The following list summarizes the experiments we carried out, with the rotation experiments being additionally summarized in Table 4.1:

A1. Varying the pitch and roll of the right checkerboard, while keeping the left checkerboard static. We simulated all the roll (α) and pitch (β) combinations over the $[-60^\circ, 60^\circ]$ interval in 10 degree steps.

A2. Varying the pitch and roll of the left checkerboard, while keeping the right checkerboard static. We varied the roll and pitch over the $[-60^\circ, 60^\circ]$ interval in 10-degrees steps.

A3. Varying the pitch and roll of the left checkerboard and right checkerboard simultaneously over the $[-60^\circ, 60^\circ]$ interval in 10-degree steps. The yaw angle of both checkerboards was set to -53.14 degrees.

TABLE 4.1: Summary of the rotation experiments with dual checkerboards and their results. The best poses are defined by the top 2 % lowest error of the projected control points. The subscript letter identifies the left (l) and right (r) checkerboards respectively. The # symbol in parenthesis represents the index of the experiment.

Experiment	Left Checkerboard Parameters	Right Checkerboard Parameters	Top 2%
			Poses and its Control Points projection Error (ϵ)[px]
A1 Left Fixed Right Rotate	$x, y, z: (-0.6, 1.0, 5.0)\text{m}$ $\alpha: 0^\circ$ $\beta: 0^\circ$ $\gamma: -53.14^\circ$	$x, y, z: (0.6, 1.0, 5.0)\text{m}$ $\alpha: [-60, 60, 10]^\circ$ $\beta: [-60, 60, 10]^\circ$ $\gamma: -53.14^\circ$	(#98) $\alpha_r: 10^\circ, \beta_r: 10^\circ, \epsilon: 1.807$
			(#72) $\alpha_r: -10^\circ, \beta_r: 10^\circ, \epsilon: 1.8260$
			(#110) $\alpha_r: 20^\circ, \beta_r: 0^\circ, \epsilon: 1.8867$
			(#30) $\alpha_r: -40^\circ, \beta_r: -20^\circ, \epsilon: 1.8930$
A2 Left Rotate Right Fixed	$x, y, z: (-0.6, 1.0, 5.0)\text{m}$ $\alpha: [-60, 60, 10]^\circ$ $\beta: [-60, 60, 10]^\circ$ $\gamma: -53.14^\circ$	$x, y, z: (0.6, 1.0, 5.0)\text{m}$ $\alpha: 0^\circ$ $\beta: 0^\circ$ $\gamma: -53.14^\circ$	(#52) $\alpha_l: -20^\circ, \beta_l: -60^\circ, \epsilon: 1.1576$
			(#132) $\alpha_l: 40^\circ, \beta_l: -40^\circ, \epsilon: 1.2514$
			(#94) $\alpha_l: 10^\circ, \beta_l: -30^\circ, \epsilon: 1.3005$
			(#92) $\alpha_l: 10^\circ, \beta_l: -50^\circ, \epsilon: 1.3379$
A3 Left Rotate Right Rotate	$x, y, z: (-0.6, 1.0, 5.0)\text{m}$ $\alpha: [-60, 60, 10]^\circ$ $\beta: [-60, 60, 10]^\circ$ $\gamma: -53.14^\circ$	$x, y, z: (0.6, 1.0, 5.0)\text{m}$ $\alpha: [-60, 60, 10]^\circ$ $\beta: [-60, 60, 10]^\circ$ $\gamma: -53.14^\circ$	(#132) $\alpha_l: 40^\circ, \beta_l: -40^\circ, \alpha_r: 40^\circ, \beta_r: -40^\circ, \epsilon: 1.5329$
			(#58) $\alpha_l: -20^\circ, \beta_l: 0^\circ, \alpha_r: -20^\circ, \beta_r: 0^\circ, \epsilon: 1.6313$
			(#157) $\alpha_l: 60^\circ, \beta_l: -50^\circ, \alpha_r: 60^\circ, \beta_r: 50^\circ, \epsilon: 1.8264$
			(#36) $\alpha_l: -40^\circ, \beta_l: 40^\circ, \alpha_r: -40^\circ, \beta_r: 30^\circ, \epsilon: 1.8902$
A4 Left Rotate Right Rotate Mirror Yaw	$x, y, z: (-0.6, 1.0, 5.0)\text{m}$ $\alpha: [-60, 60, 10]^\circ$ $\beta: [-60, 60, 10]^\circ$ $\gamma: 53.14^\circ$	$x, y, z: (0.6, 1.0, 5.0)\text{m}$ $\alpha: [-60, 60, 10]^\circ$ $\beta: [-60, 60, 10]^\circ$ $\gamma: -53.14^\circ$	(#28) $\alpha_l: -40^\circ, \beta_l: -40^\circ, \alpha_r: -40^\circ, \beta_r: -40^\circ, \epsilon: 1.5135$
			(#98) $\alpha_l: 10^\circ, \beta_l: 10^\circ, \alpha_r: 10^\circ, \beta_r: 10^\circ, \epsilon: 1.6489$
			(#73) $\alpha_l: -10^\circ, \beta_l: 20^\circ, \alpha_r: -10^\circ, \beta_r: 20^\circ, \epsilon: 1.9723$
			(#68) $\alpha_l: -10^\circ, \beta_l: -30^\circ, \alpha_r: -10^\circ, \beta_r: -30^\circ, \epsilon: 2.1704$
A5 Left Rotate Right Rotate Mirror Roll/Pitch	$x, y, z: (-0.6, 1.0, 5.0)\text{m}$ $\alpha: [-60, 60, 10]^\circ$ $\beta: [-60, 60, 10]^\circ$ $\gamma: 53.14^\circ$	$x, y, z: (0.6, 1.0, 5.0)\text{m}$ $\alpha: [60, -60, -10]^\circ$ $\beta: [60, -60, -10]^\circ$ $\gamma: -53.14^\circ$	(#74) $\alpha_l: -10^\circ, \beta_l: 30^\circ, \alpha_r: 10^\circ, \beta_r: 30^\circ, \epsilon: 1.7785$
			(#73) $\alpha_l: -10^\circ, \beta_l: 20^\circ, \alpha_r: 10^\circ, \beta_r: 20^\circ, \epsilon: 1.7820$
			(#21) $\alpha_l: -50^\circ, \beta_l: 20^\circ, \alpha_r: 50^\circ, \beta_r: 20^\circ, \epsilon: 1.7852$
			(#93) $\alpha_l: 10^\circ, \beta_l: -40^\circ, \alpha_r: -10^\circ, \beta_r: 40^\circ, \epsilon: 2.0763$

A4. This experiment was similar to A3. It varied the pitch and roll of the left checkerboard and right checkerboard simultaneously over the $[-60^\circ, 60^\circ]$ interval in 10-degree steps. However, in this experiment, we mirrored the yaw angle between checkerboards, setting the left γ_l angle to 53.14, and the right γ_r angle to -53.14 degrees.

A5. Similar to the A3 experiment, we varied simultaneously the roll and pitch angles in the $[-60^\circ, 60^\circ]$ interval, in 10-degree steps. But in this experiment, we mirrored all the rotations ($\alpha_l = -\alpha_r, \beta_l = -\beta_r$, and $\gamma_l = -\gamma_r$).

Dual Checkerboard Horizontal Positioning Experiments

In these experiments, we investigated the checkerboard pose performance when varying the horizontal positioning for two simultaneous checkerboards in the camera field-of-view. We used different combinations while keeping the vertical positioning fixed.

From Section 4.5.3, we learned that the closer the checkerboard is to the camera plane, the better the checkerboard corner detector performance. For this reason, we positioned both checkerboards five meters from the camera plane along the camera's Z-axis. Additionally, for these experiments we selected the roll and pitch rotations that provided the best corner detection results. With these initial conditions, we simulated the following experiments involving variation of the horizontal positioning of the checkerboards. We prepend all of the experiments in this section by the letter **H**, denoting the horizontal variation. We also summarize the experiment parameters and results in Table 4.2:

H1. In this experiment, we modified the horizontal position of both checkerboards simultaneously along the camera X-axis, while each checkerboard was facing the camera with different rotation angles. We positioned the left checkerboard at the left edge of the image, with the right checkerboard next to its right side separated by 0.9 m. Additionally, we set the right checkerboard to have -40 , -20 , and -53.14 degrees for the roll, pitch, and yaw, respectively; and 0 , 0 , and -53.14 degrees for the left checkerboard. We then moved both checkerboards with 0.02 m steps until the right checkerboard reached the image edge.

H2. Similar to H1, in this experiment we changed only the horizontal position until the right checkerboard reached the image edge, in 0.02 m steps. However, the checkerboard normals had the rotation angles mirrored. The left checkerboard roll, pitch, yaw rotations were set to 50 , -20 , and -53.14 degrees, respectively, while the right ones were set to -50 , 20 , and -53.14 degrees.

H3. Contrary to the H1 and H2, in this experiment we fixed the initial position of both checkerboards to the center of the image and moved the horizontal position of each checkerboard simultaneously towards the left and right image edges in 0.01 m steps. We set the right checkerboard roll, pitch, and yaw rotations to -40 , -20 , and -53.14 degrees, respectively, while setting the left checkerboard rotations to 0 , 0 , and -53.14 degrees.

H4. In this experiment we initially positioned both checkerboards at the center of the frame, separated by 0.8 m. We then moved each checkerboard towards the left and right edges respectively in 0.01 m steps. We also set the left and right checkerboards to point towards opposite directions. The right checkerboard roll was set to -50 , the

pitch to 20, and the yaw to -53.14 degrees; the left one was set to 50, -20 , -53.14 degrees in roll, pitch, and yaw, respectively.

H5. In this experiment, we set both checkerboards at the center of the frame separated by 0.8 m, with their rotation angles mirrored. We then moved the left checkerboard towards the left edge of the image and simultaneously moved the right checkerboard to the right side of the frame. Both checkerboards were moved in 0.01 m steps. The right checkerboard roll, pitch, and yaw rotations were set to 50, -20 , and -53.14 degrees, respectively, while the left checkerboard rotations were set to -50 , 20, and -53.14 degrees.

Dual Checkerboard Vertical Positioning Experiments

In these experiments, we examined the checkerboard pose performance when varying symmetrically and asymmetrically the vertical positioning of the two checkerboards in the camera field-of-view. Similar to the experiments of Section 4.6.3, we selected the most performant rotations for the checkerboard detector from Section 4.5.3. The following list explains the vertical positioning experiments. We prepend the experiments in this section with **V**, denoting the vertical variation. We additionally include a summary of these experiments in Table 4.2:

V1. In this experiment, we initially positioned the left checkerboard at the bottom left edge and the right checkerboard at the lower right of the frame. We fixed the rotations of both checkerboards to -40 , -20 , and -53.14 degrees for the roll, pitch, and yaw, respectively. We then moved both checkerboards simultaneously to the top of the image in steps of 0.02 m, without modifying the horizontal position.

V2. In this experiment we initially positioned the left checkerboard to the top-left edge and the right checkerboard to the lower right of the frame. We fixed both checkerboard rotations to -40 , -20 , and -53.14 degrees for the roll, pitch, and yaw, respectively. We then moved the left checkerboard towards the bottom left and the right checkerboard to the top right in steps of 0.02 m, without modifying the horizontal position.

TABLE 4.2: Summary of the horizontal and vertical positioning experiments with dual checkerboards and their results. The best poses are defined by the top 2% lowest error of the projected control points. The subscripts l and r identify the left and right checkerboards respectively. The # symbol in parenthesis represents the index of the experiment.

Experiment	Left Checkerboard Parameters	Right Checkerboard Parameters	Top 2% Poses and its Control Points Re-projection Error(ϵ)[px]
H1 Left Perpendicular Horizontally Together	$\alpha, \beta, \gamma: (0, 0, -53.14)^\circ$ $x: [-1.2, 0.4, 0.02]$ m $y: 1.0$ m $z: 5.0$ m	$\alpha, \beta, \gamma: (-40, -20, -53.14)^\circ$ $x: [-0.3, 1.3, 0.02]$ m $y: 1.0$ m $z: 5.0$ m	(#74) $x_l: 0.28$ m, $x_r: 1.18$ m, $\epsilon: 1.0277$ (#78) $x_l: 0.36$ m, $x_r: 1.26$ m, $\epsilon: 1.0783$ (#15) $x_l: -0.89$ m, $x_r: 0.0$ m, $\epsilon: 1.3312$ (#5) $x_l: -1.10$ m, $x_r: -0.02$ m, $\epsilon: 1.3568$
H2 Left/Right Mirrored Horizontally Together	$\alpha, \beta, \gamma: (50, -20, -53.14)^\circ$ $x: [-1.2, 1.3, 0.2]$ m $y: 1.0$ m $z: 5.0$ m	$\alpha, \beta, \gamma: (-50, 20, -53.14)^\circ$ $x: [-0.3, 0.4, 0.02]$ m $y: 1.0$ m $z: 5.0$ m	(#14) $x_l: -0.92$ m, $x_r: -0.2$ m, $\epsilon: 1.1834$ (#75) $x_l: 0.3$ m, $x_r: -1.2$ m, $\epsilon: 2.0962$ (#17) $x_l: -0.86$ m, $x_r: 0.04$ m, $\epsilon: 2.1201$ (#18) $x_l: -0.84$ m, $x_r: 0.06$ m, $\epsilon: 2.1817$
H3 Left Perpendicular Horizontally Separate	$\alpha, \beta, \gamma: (0, 0, -53.14)^\circ$ $x: [-0.4, -1.2, -0.01]$ m $y: 1.0$ m $z: 5.0$ m	$\alpha, \beta, \gamma: (-40, -20, -53.14)^\circ$ $x: [0.4, 1.2, 0.01]$ m $y: 1.0$ m $z: 5.0$ m	(#74) $x_l: -1.14$ m, $x_r: 1.14$ m, $\epsilon: 1.0383$ (#65) $x_l: -1.05$ m, $x_r: 1.05$ m, $\epsilon: 1.0566$ (#73) $x_l: -1.13$ m, $x_r: 1.13$ m, $\epsilon: 1.0842$ (#78) $x_l: -1.18$ m, $x_r: 1.18$ m, $\epsilon: 1.1105$
H4 Left/Right Mirrored Horizontally Separate	$\alpha, \beta, \gamma: (50, -20, -53.14)^\circ$ $x: [-0.4, -1.2, -0.01]$ m $y: 1.0$ m $z: 5.0$ m	$\alpha, \beta, \gamma: (-50, 20, -53.14)^\circ$ $x: [0.4, 1.2, 0.01]$ m $y: 1.0$ m $z: 5.0$ m	(#20) $x_l: -0.6$ m, $x_r: 0.6$ m, $\epsilon: 1.7851$ (#43) $x_l: -0.83$ m, $x_r: 0.83$ m, $\epsilon: 1.9118$ (#54) $x_l: -0.94$ m, $x_r: 0.94$ m, $\epsilon: 1.9452$ (#31) $x_l: -0.71$ m, $x_r: 0.71$ m, $\epsilon: 2.0131$
H5 Left/Right Mirrored Inverted Horizontally Separate	$\alpha, \beta, \gamma: (-50, 20, -53.14)^\circ$ $x: [-0.4, 1.2, 0.01]$ m $y: 1.0$ m $z: 5.0$ m	$\alpha, \beta, \gamma: (50, -20, -53.14)^\circ$ $x: [0.4, -1.2, 0.01]$ m $y: 1.0$ m $z: 5.0$ m	(#6) $x_l: -0.92$ m, $x_r: -0.2$ m, $\epsilon: 4.084$ (#14) $x_l: 0.3$ m, $x_r: -1.2$ m, $\epsilon: 4.7277$ (#13) $x_l: -0.86$ m, $x_r: 0.04$ m, $\epsilon: 4.7709$ (#23) $x_l: -0.84$ m, $x_r: 0.06$ m, $\epsilon: 5.43005$
V1 Left/Right Identical Rotations Vertically Together	$\alpha, \beta, \gamma: (-40, -20, -53.14)^\circ$ $x: -1.13$ m $y: [0.35, 1.65, 0.02]$ m $z: 5.0$ m	$\alpha, \beta, \gamma: (-40, -20, -53.14)^\circ$ $x: 1.13$ m $y: [0.35, 1.65, 0.02]$ m $z: 5.0$ m	(#42) $y_l: 1.19$ m, $y_r: 1.19$ m, $\epsilon: 0.6763$ (#37) $y_l: 1.09$ m, $y_r: 1.09$ m, $\epsilon: 0.7285$ (#55) $y_l: 1.45$ m, $y_r: 1.45$ m, $\epsilon: 0.7542$ (#57) $y_l: 1.49$ m, $y_r: 1.49$ m, $\epsilon: 0.8238$
V2 Left/Right Identical Rotations Vertically Opposite	$\alpha, \beta, \gamma: (-40, -20, -53.14)^\circ$ $x: -1.13$ m $y: [1.65, 0.35, -0.02]$ m $z: 5.0$ m	$\alpha, \beta, \gamma: (-40, -20, -53.14)^\circ$ $x: 1.13$ m $y: [0.35, 1.65, 0.02]$ m $z: 5.0$ m	(#34) $y_l: 0.97$ m, $y_r: 1.03$ m, $\epsilon: 0.8916$ (#46) $y_l: 0.73$ m, $y_r: 1.27$ m, $\epsilon: 0.9385$ (#37) $y_l: 0.91$ m, $y_r: 1.09$ m, $\epsilon: 1.0684$ (#64) $y_l: 0.37$ m, $y_r: 1.63$ m, $\epsilon: 1.1009$

Dual Checkerboard Distance Experiments

Although we know that the corner detector performs better the closer the checkerboard is to the camera, in these experiments we evaluated the impacts of checkerboard scale on the accuracy of the intrinsic parameters. Similar to the horizontal and vertical experiments, we selected the most performant poses from Section 4.5.3. Additionally, from Section 4.6.3 we chose to keep the checkerboards near the left and right edges since this positioning provided better parameter estimation. The following list describes the distance experiments. We prepend the experiments in this section with **D**, denoting the distance. We additionally include a summary of these experiments in Table 4.3:

D1. In this experiment, we initially positioned both checkerboards at the camera height but distanced the left checkerboard 5 m from the camera and the right one at 10 m. We kept the left checkerboard static during the whole experiment on the left image edge, while we reduced the right checkerboard distance in 0.1 m steps until it reached 4.6 m. The roll, pitch, and yaw for both checkerboards was set to -40 , -20 , and -53.14 degrees, respectively.

D2. In this experiment, we initially positioned both checkerboards at the camera height, with one on the left image edge and the other on the right. We set the initial distance of both checkerboards to 10 m, and then moved them towards the camera until they reached a distance of 5 m from the camera. We fixed the roll, pitch, and yaw for both checkerboards to -40 , -20 , and -53.14 degrees, respectively.

4.6.4 Dual Checkerboard Calibration Results

Looking at the results from Table 4.1, Table 4.2, and Table 4.3 we can infer the following guidelines:

1. It is essential to locate the checkerboards near to the image edge in order to generate distortion parameters closer to the ground truth.
2. Having a single checkerboard pointing directly to the camera while the other is rotated seems to produce better overall parameters. We hypothesize that this dual checkerboard combination provides enough information to estimate

TABLE 4.3: Summary of the distance between camera and checkerboard experiments with dual checkerboards and their results. The best poses are defined by the top 2% lowest error of the projected control points. The subscript l and r identify the left and right checkerboards, respectively. The # symbol in parenthesis represents the index of the experiment.

Experiment	Left Checkerboard Parameters	Right Checkerboard Parameters	Top 2%
			Poses and its Control Points Re-projection Error (ϵ)[px]
D1 Left Fixed Right Approach	$\alpha, \beta, \gamma: (-40, -20, -53.14)^\circ$ x: -1.13 m y: 1.0 m z: 5.0 m	$\alpha, \beta, \gamma: (-40, -20, -53.14)^\circ$ x: 1.13 m y: 1.0 m z: [10.0, 4.6, -0.1] m	(#32) $z_l: 6.8$ m, $\epsilon: 1.2415$
			(#43) $z_r: 5.7$ m, $\epsilon: 1.4161$
			(#52) $z_r: 4.8$ m, $\epsilon: 1.4630$
			(#54) $z_r: 4.6$ m, $\epsilon: 1.4680$
D2 Left/Right Approach	$\alpha, \beta, \gamma: (-40, -20, -53.14)^\circ$ x: -1.13 m y: 1.0 m z: [10.0, 4.6, -0.1]	$\alpha, \beta, \gamma: (-40, -20, -53.14)^\circ$ x: 1.13 m y: 1.0 m z: [10.0, 4.6, -0.1]	(#49) $z_l: 5.1$ m, $z_r: 5.1$ m, $\epsilon: 1.3418$
			(#30) $z_l: 7.0$ m, $z_r: 7.0$ m, $\epsilon: 1.5207$
			(#50) $z_l: 5.0$ m, $z_r: 5.0$ m, $\epsilon: 1.5672$
			(#46) $z_l: 5.4$ m, $z_r: 5.4$ m, $\epsilon: 1.5840$

the center point and the focal length accurately while also producing sufficient data to calculate the distortion parameters.

3. Keeping the checkerboard rotation inside the $[-60^\circ, 60^\circ]$ interval, as we defined in Section 4.5.3, is essential to increase the corner detection accuracy.
4. Checkerboards located on the image corners are not required, as long as the user can position the checkerboards near the image edge.
5. Checkerboards with varied rotation relative to the camera plane are more beneficial than those located farther away, since closer checkerboards produce more accurate corners. We found this to be true because rotated checkerboards near the acceptable rotation limits also provide scale information.

4.6.5 Multiple Checkerboards Calibration

Based on the guidelines we defined in Section 4.6.4, we defined the poses of up to 10 checkerboards in the image frame. Since we narrowed down the combinations, all these experiments contain only fixed positions and rotations, instead of simulation intervals. Additionally, for the top-score poses, we manually picked other performant poses to form two setups: one with six and the other with seven checkerboards. To form these "custom" setups, denoted by subscript **C**, we chose what we considered "easy" to replicate positions in the real world. For instance, we preferred (but did not prioritize) checkerboard positions closer to the ground or separated

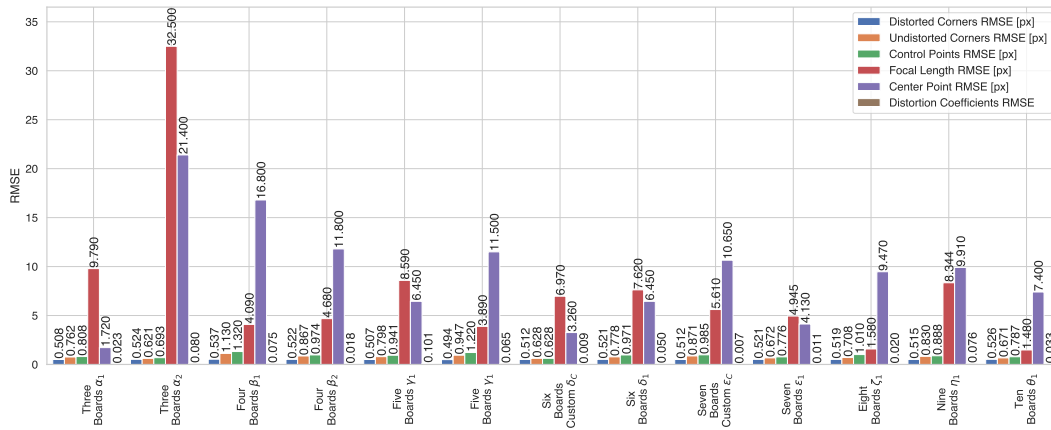


FIGURE 4.6: Quantitative result summary for the simulation experiments with multiple checkerboards.

from each other; this would help accelerate checkerboard positioning setup in a real-world scenario. However, we always followed the guidelines we defined in Section 4.6.4.

The summary of the simulation experiments with multiple checkerboards is in Table 4.4. We decided to limit the number of checkerboards to 10 because adding more checkerboards would prove a challenge to set up in the real and simulated worlds while maintaining all the checkerboards at a close range to maximize the performance of the corner detector.

4.6.6 Multiple Checkerboards Calibration Results

Table 4.5 and Figure 4.6 show that it might be feasible to calibrate with as few as three checkerboards and obtain good calibration results. However, in Figure 4.6 we can see that the α_1 experiment, with three checkerboards, presents a larger than average focal length error, which might produce scaling problems and an irregular projection at different distances.

Additionally, on the same Section 4.6.6, we can see that having more checkerboards produces more accurate calibration intrinsic parameters. However, we can also infer that using six or seven checkerboards can provide enough information to estimate highly accurate calibration parameters for 3D applications. This number of checkerboards gives a realistic and balanced approach since having more than seven simultaneous checkerboards in the frame might cause difficulties while replicating

TABLE 4.4: Summary of the simulation experiments with multiple checkerboards. The Experiment column contains the number of checkerboards and the details for each checkerboard per line. Position is expressed in meters, while rotations are expressed in degrees.

Experiment (xyz),(rpy)	Experiment (xyz),(rpy)
Three Checkerboards α_1 (-0.05, 1.0, 4)m, (0, 0, -53.14) $^\circ$ (1.05, 1.4, 4.9)m, (10.07, 51.04, 40.53) $^\circ$ (-1.35, 0.65, 5.6)m, (34.04, -27.33, -35.99) $^\circ$	Three Checkerboards α_2 (-0.05, 1, 4)m, (0, 0, -53.14) $^\circ$ (1.05, 1.4, 4.9)m, (36.84, 42.16, -62.15) $^\circ$ (-1.35, 0.65, 5.6)m, (-33.57, 18.33, -6.1) $^\circ$
Four Checkerboards β_1 (-0.45, 1.5, 5)m, (4.44, -14.28, -64.04) $^\circ$ (0.3, 0.7, 4.5)m, (-11.53, -38.45, -60.02) $^\circ$ (1.05, 1.4, 4.8)m, (10.07, 51.04, 40.53) $^\circ$ (-1.35, 0.65, 5.6)m, (34.04, -27.33, -35.99) $^\circ$	Four Checkerboards β_2 (-0.45, 1.5, 5.0)m, (-11.53, -38.45, -60.02) $^\circ$ (0.3, 0.7, 4.5)m, (0, 0, -53.14) $^\circ$ (1.05, 1.4, 4.8)m, (10.07, 51.04, 40.53) $^\circ$ (-1.35, 0.65, 5.6)m, (34.04, -27.33, -35.99) $^\circ$
Five Checkerboards γ_1 (0.05, 1, 4.1)m, (0, 0, -53.14) $^\circ$ (0.95, 1.6, 4.6)m, (-40.17, -20.68, -1.58) $^\circ$ (0.95, 0.65, 4.6)m, (10.07, 51.04, 40.53) $^\circ$ (-0.95, 1.5, 4.8)m, (-11.53, -38.45, -60.02) $^\circ$ (-0.75, 0.6, 4.3)m, (34.04, -27.33, -35.99) $^\circ$	Five Checkerboards γ_2 (0.05, 1, 4.3)m, (0, 0, -53.14) $^\circ$ (0.95, 1.6, 4.8)m, (-40.17, -20.68, -1.58) $^\circ$ (0.95, 0.65, 4.8)m, (10.07, 51.04, 40.53) $^\circ$ (-0.95, 1.5, 5)m, (-11.53, -38.45, -60.02) $^\circ$ (-0.75, 0.6, 4.5)m, (34.04, -27.33, -35.99) $^\circ$
Six Checkerboards δ_1 (-0.05, 1.4, 4.4)m, (0, 0, -53.14) $^\circ$ (0.1, 0.6, 4.2)m, (-36.16, -43.53, -61.14) $^\circ$ (0.95, 1.6, 4.6)m, (20.17, -20.68, -1.58) $^\circ$ (0.95, 0.65, 4.6)m, (10.07, 51.04, 40.53) $^\circ$ (-0.95, 1.5, 4.7)m, (-11.53, -38.45, -60.02) $^\circ$ (-0.75, 0.6, 4.3)m, (34.04, -27.33, -35.99) $^\circ$	Seven Checkerboards ϵ_1 (-0.05, 1.7, 4.9)m, (-36.16, -43.53, 10.14) $^\circ$ (0.15, 1.0, 5.2)m, (0, 0, -53.14) $^\circ$ (0.1, 0.4, 4.7)m, (64.74, 8.31, -24.57) $^\circ$ (0.95, 1.6, 4.5)m, (20.17, -20.68, -1.58) $^\circ$ (0.95, 0.65, 4.5)m, (10.07, 51.04, 40.53) $^\circ$ (-0.95, 1.5, 4.7)m, (-11.53, -38.45, -60.02) $^\circ$ (-0.75, 0.6, 4.3)m, (34.04, -27.33, -35.99) $^\circ$
Eight Checkerboards ζ_1 (-1.15, 1.1, 5)m, (-11.53, -38.45, -60.02) $^\circ$ (-0.5, 1.7, 4.9)m, (-36.16, -43.53, 10.14) $^\circ$ (-0.6, 0.4, 5)m, (34.04, -27.33, -35.99) $^\circ$ (-0.35, 1.0, 5.4)m, (0, 0, -53.14) $^\circ$ (0.55, 0.95, 5)m, (40.93, 38.59, 36.58) $^\circ$ (0.5, 0.5, 4.4)m, (64.74, 8.31, 24.57) $^\circ$ (0.55, 1.59, 4.5)m, (20.17, -20.68, -1.58) $^\circ$ (1.35, 0.98, 5.05)m, (10.07, 51.04, 89.53) $^\circ$	Nine Checkerboards η_1 (-1.37, 0.9, 5.1)m, (-11.53, -58.45, -80.02) $^\circ$ (-0.8, 1.7, 4.9)m, (-36.16, -43.53, 10.14) $^\circ$ (-0.6, 0.4, 5.0)m, (34.04, -27.33, -35.99) $^\circ$ (-0.53, 1.08, 4.8)m, (-20.84, 26.11, -4.00) $^\circ$ (0.03, 1.65, 5.2)m, (0, 0, -53.14) $^\circ$ (0.4, 0.98, 4.65)m, (40.93, 38.59, 36.58) $^\circ$ (0.5, 0.5, 4.4)m, (64.74, 8.31, 24.57) $^\circ$ (0.75, 1.59, 4.5)m, (20.17, -20.68, -1.58) $^\circ$ (1.35, 0.98, 5.05)m, (10.07, 51.04, 89.53) $^\circ$
Ten Checkerboards θ_1 (-1.41, 1.05, 5.2)m, (-11.53, -58.45, -80.02) $^\circ$ (-0.8, 1.75, 5.0)m, (-36.16, -43.53, 10.14) $^\circ$ (-1.06, 0.32, 5.1)m, (39.04, -37.33, -15.99) $^\circ$ (-0.48, 1.0, 4.4)m, (-20.84, 26.11, -4.00) $^\circ$ (0.03, 1.65, 5.2)m, (0, 0, -53.14) $^\circ$ (0.4, 0.98, 4.65)m, (30.93, 28.59, 36.58) $^\circ$ (0.9, 0.4, 4.7)m, (64.74, 8.31, 24.57) $^\circ$ (0.89, 1.69, 4.9)m, (20.17, -20.68, -1.58) $^\circ$ (1.40, 0.95, 5.2)m, (10.07, 51.04, 89.53) $^\circ$ (-0.06, 0.3, 4.95)m, (64.74, 8.31, -24.57) $^\circ$	
Six Checkerboards δ_C (0.05, 1.5, 5.1)m, (0, 0, -53.14) $^\circ$ (0.05, 0.30, 5.2)m, (10, -5, 0) $^\circ$ (-1.18, 1.3, 4.9)m, (9.23, -4.70, -1.58) $^\circ$ (1.18, 1.3, 5.1)m, (-4.72, 23.40, -15.88) $^\circ$ (1.29, 0.24, 5.2)m, (63.74, 18.31, -12.57) $^\circ$ (-1.38, 0.32, 5.3)m, (-22.92, -58.33, 5.44) $^\circ$	Seven Checkerboards ϵ_C (0.05, 1.90, 5.5)m, (0, 5, 0) $^\circ$ (0.05, 1.05, 5.4)m, (0, 0, -53.14) $^\circ$ (0.05, 0.28, 5.0)m, (10, -5, 0) $^\circ$ (-1.18, 1.3, 4.9)m, (9.23, -4.70, -1.58) $^\circ$ (1.18, 1.3, 5.1)m, (-4.72, 23.40, -15.88) $^\circ$ (1.29, 0.24, 5.2)m, (63.74, 18.31, -12.57) $^\circ$ (-1.38, 0.32, 5.3)m, (-22.92, -58.33, 5.44) $^\circ$

TABLE 4.5: Summary of the results for the simulation experiments with multiple checkerboards. The subscript C denotes the experiments of which the poses were manually selected following our guidelines.

Experiment	Distorted Corners RMSE [px]	Undistorted Corners RMSE [px]	Control Points RMSE [px]	Focal Length Error [px]	Center Point Error [px]	Distortion Error
Ten Checkerboards θ_1	0.526	0.671	0.787	1.48	7.40	0.033
Nine Checkerboards η_1	0.515	0.830	0.888	8.344	9.91	0.076
Eight Checkerboards ζ_1	0.519	0.708	1.01	1.58	9.47	0.025
Seven Checkerboards ϵ_1	0.521	0.672	0.776	4.945	4.13	0.011
Six Checkerboards δ_1	0.521	0.778	0.971	7.62	6.45	0.050
Five Checkerboards γ_2	0.494	0.947	1.22	3.89	11.5	0.065
Five Checkerboards γ_1	0.507	0.798	0.941	8.59	6.45	0.101
Four Checkerboards β_2	0.522	0.867	0.974	4.68	11.8	0.018
Four Checkerboards β_1	0.537	1.13	1.32	4.09	16.8	0.075
Three Checkerboards α_2	0.524	0.651	0.693	32.5	21.4	0.0803
Three Checkerboards α_1	0.508	0.762	0.808	9.79	1.72	0.0232
Six Checkerboards δ_C	0.512	0.628	0.628	6.97	3.26	0.0092
Seven Checkerboards ϵ_C	0.512	0.871	0.985	5.61	10.65	0.007

the setups presented in Table 4.4 when using real checkerboards and tripods. For this reason, in the next section, we decided to investigate and test these setups with real cameras, project the point cloud from a 3D LiDAR, and verify the quality of the camera intrinsic parameters.

4.7 Real-world Calibration Verification

For real-world calibration verification, we used the Lucid Labs machine vision camera that was introduced in Section 4.3.1, and an additional FLIR Blackfly camera (PGE-23S6C), paired with an 8 mm μ Tron lens. This camera has a lower resolution (1920x1200 pixels, approximately 2.3 MP) and a larger image sensor, leading to a wider field of view even if it uses a lens with the same focal length. Testing the calibration setups with this camera helped us to validate the checkerboard poses at a different image scale. We proceeded to replicate the checkerboard positions we obtained in Section 4.6.6 for the setups with six and seven checkerboards: $\delta_1, \delta_C, \epsilon_1, \epsilon_C$, for the Lucid and the FLIR camera, a total of eight experiments.

4.7.1 Multiple checkerboard verification experiments

The checkerboard poses we obtained from the simulator are defined by their roll, pitch, and yaw rotations. These angles are numbers that we, as humans, might find

TABLE 4.6: Summary of the extrinsic parameters between the cameras and the 3D LiDAR for the outdoors dataset: x, y, z are in meters; roll, pitch and yaw are in radians. We truncated the floating-point values to 3 digits for formatting purposes.



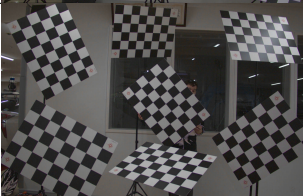





Experiment	x	y	z	roll	pitch	yaw
Lucid δ_1	0.031	-0.14	0.03	-1.515	0.005	-1.628
Lucid δ_C	0.031	-0.14	0.03	-1.474	0.01	-1.625
Lucid ϵ_1	0.031	-0.14	0.05	-1.513	-0.01	-1.619
Lucid ϵ_C	0.031	-0.12	0.05	-1.51	-0.01	-1.623
FLIR δ_1	0.031	-0.14	0.03	-1.5	0.005	-1.564
FLIR δ_C	0.031	-0.14	0.03	-1.48	0.005	-1.564
FLIR ϵ_1	0.031	-0.14	0.03	-1.5	0.005	-1.564
FLIR ϵ_C	0.031	-0.12	0.03	-1.504	-0.02	-1.563

difficult to replicate in the real world with such accuracy. For this reason, and to simplify and accelerate the checkerboard positioning in a garage, we created transparent image overlays of the checkerboard poses rendered by the simulator. We rendered these on top of the camera stream and projected the composed image on a large screen while replicating the checkerboard poses defined by the $\delta_1, \delta_C, \epsilon_1, \epsilon_C$ experiments in a garage. Figure 4.7 shows the setup we used while matching the checkerboards poses with the help of tripods.

Having matched the checkerboard positions with our simulation setups, we captured a single image for the $\delta_1, \delta_c, \epsilon_1$ and ϵ_c experiments from Section 4.6.5 with the Lucid and the FLIR cameras; we then proceeded to detect the checkerboard corners with the corner detector introduced in Section 4.3.1, and obtained the intrinsic parameters for each camera setup using OpenCV [34].

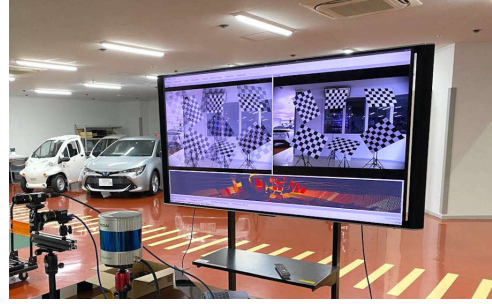
In addition to the one-shot calibration dataset, we captured camera and LiDAR data outdoors. With the help of this dataset, we obtained the extrinsic calibration parameters between the Lucid and FLIR cameras and the 3D LiDAR, which we summarized in Table 4.6. We then individually projected the 3D LiDAR point cloud into the same image captured by the camera using the four sets of intrinsic parameters ($\delta_1, \delta_C, \epsilon_1, \epsilon_C$). We repeated this procedure for the second camera.

TABLE 4.7: Summary of the real-world experiments with multiple checkerboards on two different cameras.

Experiment	Image
Lucid Six Checkerboards α_1	
Lucid Six Checkerboards α_C	
Lucid Seven Checkerboards α_1	
Lucid Seven Checkerboards α_C	
FLIR Six Checkerboards α_1	
FLIR Six Checkerboards α_C	
FLIR Seven Checkerboards α_1	
FLIR Seven Checkerboards α_C	



(A) Image overlays used to help the checkerboard positioning.



(B) Experiment Setup in the garage.

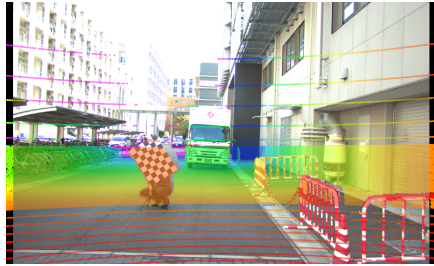
FIGURE 4.7: Multiple checkerboard verification experiments in a garage. (a) shows the composed image by the camera stream and the checkerboards overlays. (b) shows the camera and the screen we used during the experiments to help us match the checkerboards poses.

4.7.2 Real-world Calibration Results

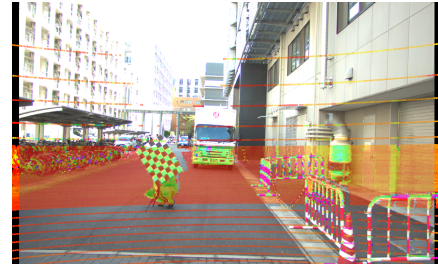
Having estimated the intrinsic parameters for the setups outlined in Table 4.7 using the single-shot setups, we obtained the absolute error between these intrinsic parameters and our baseline values. Additionally, we also calculated the checkerboard corner re-projection error, and summarized the results in Figure 4.10. In this figure, we can verify that both of the seven checkerboard setups provided the most reliable intrinsic parameters, as we expected. Finally, as a qualitative examination, we projected the point cloud from a Hesai Pandar 64 LiDAR on the rectified image by each set of estimated parameters from the $\delta_1, \delta_C, \epsilon_1, \epsilon_C$ experiments and compiled them in Figure 4.8 and Figure 4.9.

Both of the ϵ experiments (seven checkerboards) show better overall performance when compared with their equivalent δ counterparts (six checkerboards), as can be observed in Figure 4.10a and Figure 4.10b. This holds true for both the Lucid and the FLIR cameras, which is consistent with our simulation results in Figure 4.6 and the qualitative results in Figures 4.8 and 4.9. The best performing experiment in our simulation (ϵ_C) obtained the lowest error in our real-world experiments. Moreover, the corresponding qualitative results from both cameras also exhibit an excellent point cloud projection as illustrated by Figures 4.8g, 4.8h, 4.9g and 4.9h.

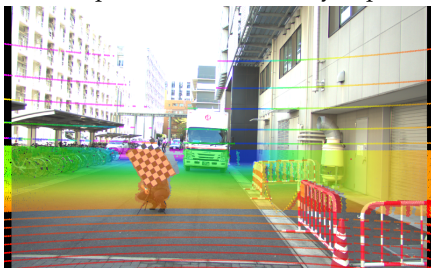
Interestingly, the worst performing simulated experiment (δ_C) also matched with our real experiment results. While obtaining the extrinsic parameters, we noted that the roll rotation for both cameras in the δ_C test is slightly different from the rest of



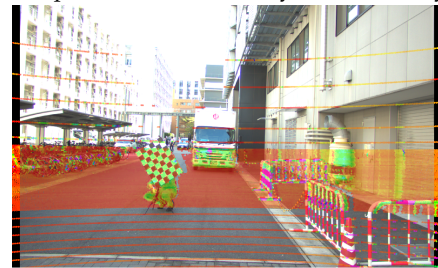
(A) Projection using δ_1 's intrinsic parameters, point cloud colored by depth.



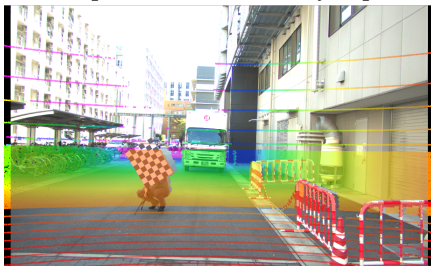
(B) Projection using δ_1 's intrinsic parameters, point cloud colored by laser intensity.



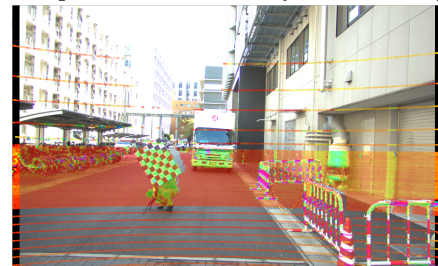
(C) Projection using δ_C 's intrinsic parameters, point cloud colored by depth.



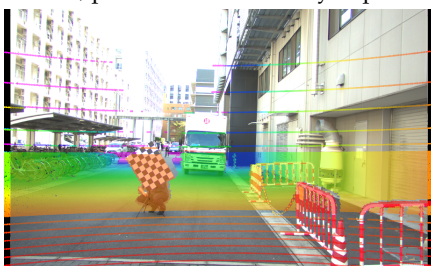
(D) Projection using δ_C 's intrinsic parameters, point cloud colored by laser intensity.



(E) Projection using ϵ_1 's intrinsic parameters, point cloud colored by depth.



(F) Projection using ϵ_1 's intrinsic parameters, point cloud colored by laser intensity.

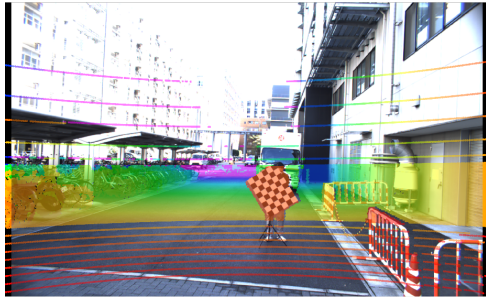


(G) Projection using ϵ_C 's intrinsic parameters, point cloud colored by depth.

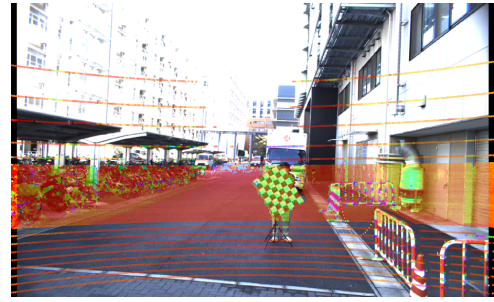


(H) Projection using ϵ_C 's intrinsic parameters, point cloud colored by laser intensity.

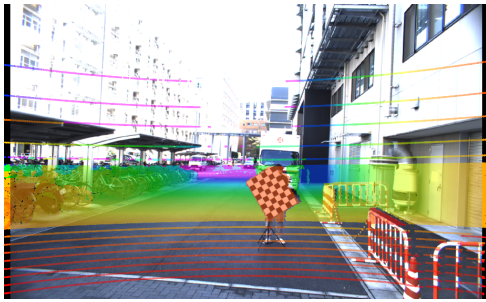
FIGURE 4.8: Point cloud projection on the Lucid camera using the intrinsic parameters by the one-shot experiments, replicating the $\delta_1, \delta_C, \epsilon_1, \text{ and } \epsilon_C$ experiments. In the left column the projected point cloud is colored by distance. In the right column, the projected point cloud is colored by the laser intensity of each return value.



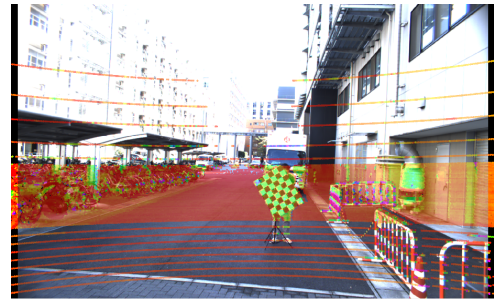
(A) Projection using δ_1 's intrinsic parameters, point cloud colored by depth.



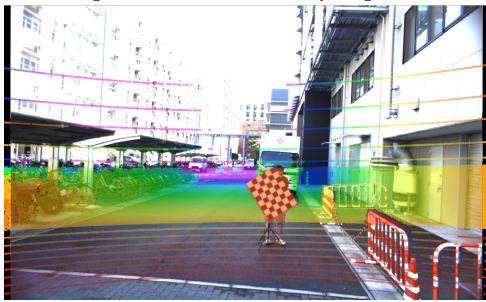
(B) Projection using δ_1 's intrinsic parameters, point cloud colored by laser intensity.



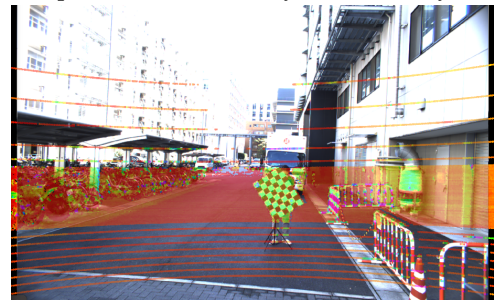
(C) Projection using δ_C 's intrinsic parameters, point cloud colored by depth.



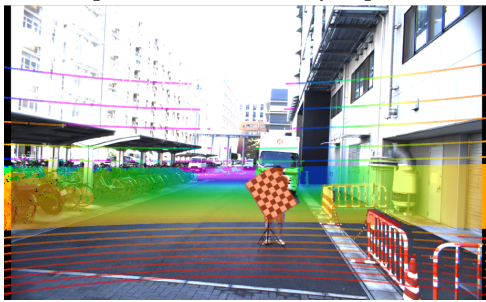
(D) Projection using δ_C 's intrinsic parameters, point cloud colored by laser intensity.



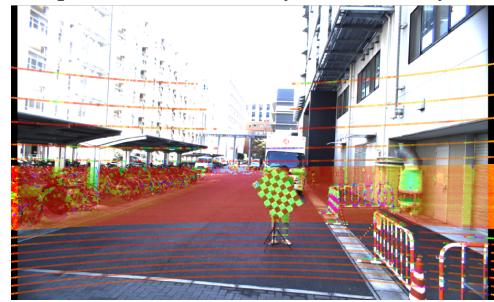
(E) Projection using ϵ_1 's intrinsic parameters, point cloud colored by depth.



(F) Projection using ϵ_1 's intrinsic parameters, point cloud colored by laser intensity.



(G) Projection using ϵ_C 's intrinsic parameters, point cloud colored by depth.



(H) Projection using ϵ_C 's intrinsic parameters, point cloud colored by laser intensity.

FIGURE 4.9: Point cloud projection on the FLIR camera using the intrinsic parameters by the one-shot experiments, replicating the $\delta_1, \delta_C, \epsilon_1, \epsilon_C$ experiments. Figures on the left column, colored the projected point cloud by distance. In the left column the projected point cloud is colored by distance. In the right column, the projected point cloud is colored by the laser intensity of each return value.

the experiments as we can see in Table 4.6. The extrinsic calibration method searches for shared features in the image and LiDAR domain and uses these as an input to an optimizer to minimize the re-projection error. It utilizes the detected 3D features and projects them using the given camera intrinsic parameters. However, if the intrinsic parameters erratically project the 3D features, the optimizer will estimate an incorrect relative camera-LiDAR position, leading us to infer that the camera-LiDAR extrinsic calibration algorithm incorrectly converged due to inaccurate intrinsic parameters. Additionally, after analyzing the qualitative results for the δ_C experiment, we noticed an adequate projection at long range, but on the right-bottom section of Figures 4.8c, 4.8d, 4.9c and 4.9d we found that the point cloud hitting the barricade located closer to the camera is incorrectly projected, pointing out a large error in the focal length. This can be confirmed in the quantitative results presented in Figure 4.10a and Figure 4.10b.

4.8 Conclusion

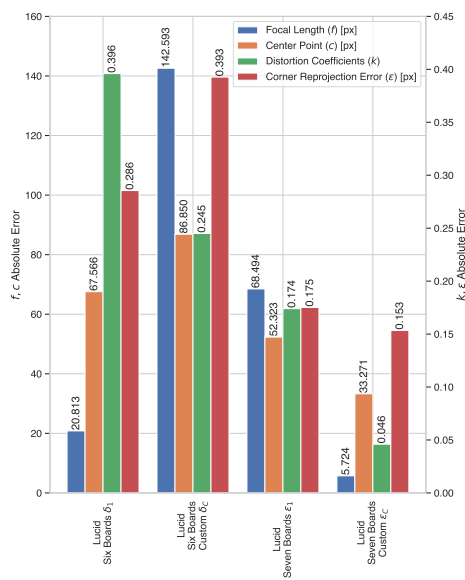
We presented a first-of-its-kind method to generate clear guidelines for single shot camera intrinsic calibration using multiple checkerboards, suitable for use in 3D applications. With the help of a simulator, we defined the position and rotation intervals that allow a corner detector to obtain optimal detections; we then generated thousands of multiple checkerboard poses and evaluated them to obtain position and rotation intervals that maximize the probability of estimating accurate camera intrinsic parameters. These results gave us enough information to generate checkerboard pose guidelines. Using these guidelines, we developed sets of multiple checkerboard poses and evaluated them synthetically and in the real world using two different cameras with different resolutions. We found consistent results between our simulations and our real-world evaluations. This enabled us to confirm that if our guidelines are followed, accurate intrinsic parameters for 3D applications can be obtained by using seven checkerboards.

The overall results show that the camera simulations helped us to accelerate the camera modeling process, its evaluation, and ultimately the creation of guidelines to obtain accurate intrinsic parameters. We can also infer that even if the simulations

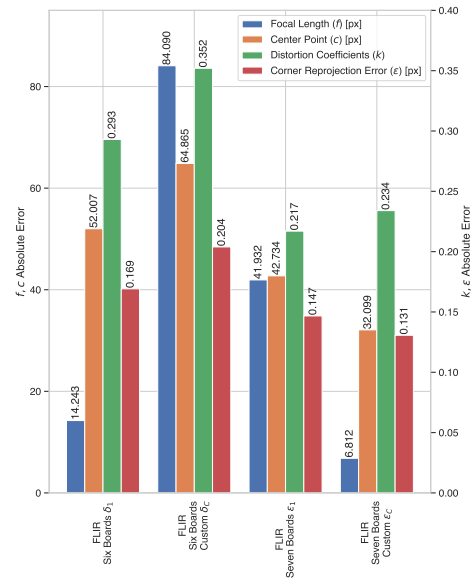
create ideal image conditions, i.e., images without chromatic aberration, vignetting, and so on, we can still transfer the lessons learned to the real world. It would have been challenging and costly to replicate the simulated experiments in the real world since they require specialized equipment to position and rotate the checkerboards. Moreover, to obtain the ground truth corner coordinates, a team of labelers would be necessary to identify each corner at the pixel level, extending the time required to complete this work.

In this chapter, we simulated and evaluated a single plumb-bob camera synthetically. For this reason, the distances we mentioned might not apply to other focal length lenses. Nevertheless, as an additional finding, we learned in Section 4.5.3 that if the checkerboard guidelines we defined in Section 4.6.6 are to be used with a different field of view camera, instead of following the recommended checkerboard distance, the user should aim to project the checkerboard area with at least $20,000 \text{ px}^2$ to produce precise corner detection. We validated this experimentally when testing on the FLIR camera in Section 4.7, which has a wider field of view and less than half the resolution of the Lucid camera. However, it is important to note that our results might not be applied to cameras paired with fish-eye lenses since these lenses are normally modeled with a non-perspective projection model, which we aim to test in future work.

Finally, the checkerboard pose guidelines we defined in Section 4.6.4 and Table 4.4 can also be used with the original method presented by Zhang [31] which involves using a single checkerboard and capturing multiple images with random orientations. Instead of moving the checkerboard randomly around the whole camera field of view, the user might aim to replicate the checkerboard poses we defined for the δ_1 , ϵ_C experiments, or even the ζ , η or θ experiments, and use the images as an input for either the OpenCV or Matlab calibration toolboxes. This would simplify and accelerate the calibration process, while helping to estimate accurate intrinsic parameters for 3D applications such as robotics, autonomous driving, or other applications that require high quality parameters.



(A) Absolute error comparison for the Lucid camera.



(B) Absolute error comparison for the FLIR camera.

FIGURE 4.10: Real-world results for the multi-checkerboard experiments. The left axis denotes the error in pixels for the sum of both focal lengths (f) and the sum of the center point (c). The right axis denotes the absolute error for the sum of distortion coefficients (k_1, k_2, k_3, p_1, p_2) and checkerboard corner re-projection error (ϵ).

Chapter 5

Multi-Sensor Fusion Toolbox for Autonomous Driving

5.1 Introduction

The field of autonomous vehicles has undergone amazingly fast development in recent years. Demonstrations from software giants such as Google, NVIDIA and Intel show that we are closer than ever to achieve the deployment of fully autonomous driving systems on general roads. Nevertheless, these solutions are completely closed, not providing feedback to the research community.

The importance of the benefits to society of the deployment of autonomous vehicle technology has been widely discussed[83]. Reduction of road accidents, safe mobility for the elderly, independent transportation for the disabled, and reduction of traffic congestion are just a few of the potential benefits promised by the adoption of autonomous driving technology.

An autonomous vehicle requires several sensors to understand its surroundings, and act accordingly in different scenarios[84]. Images from camera devices, range data from LiDARs, speed information from radars, and other sensor data is fused to achieve single-digit centimeter-level accuracy of the object detection. Thanks to the fast development of autonomous driving technologies, the cost of these sensors is rapidly reducing.

A lot of research has been carried out in the calibration, fusion and classification field. However, most existing work focus on a unique sensor type (i.e. rotating LiDAR), or are not designed to function with multiple sensors simultaneously.

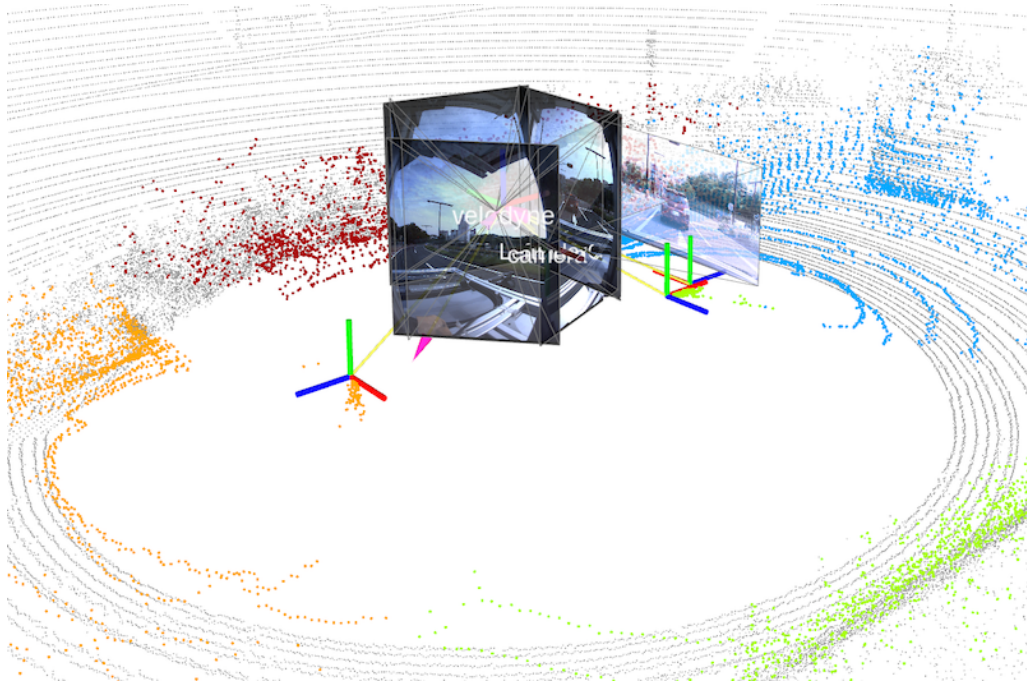


FIGURE 5.1: Successful calibration result of six wide-angle cameras, one rotating LiDAR and four MEMS (Micro Electro-Mechanical Systems) LiDARs. Each LiDAR is shown in different color.

The integration of multiple LiDARs and cameras allows the system to benefit from redundant, complementary and timely information. Moreover, multiple sensors distributed around the vehicle improve the field of view, and hence the safety of the system and its users. Nevertheless, increasing the number of sensors requires better synchronization, fast and reliable fusion techniques, and optimized processing methods due to the increased bandwidth. Figure 5.1 shows a successful integration of multiple LiDAR and camera sensors of different types using this work.

In this paper, we present an open and easy-to-use perception framework for multi-LiDAR, multi-camera calibration, fusion, and a point cloud ground classifier. Our experience with autonomous vehicles has showed that these phases are the foundation for any higher level perception system, whether it is based on traditional machine learning [85] or deep learning. New approaches such as [86], [87], [88] are good examples of state-of-the-art systems that require a solid calibration and fusion framework. In summary, the main contributions of this work are as follows:

- The proposal of a reliable LiDAR-to-LiDAR extrinsic calibration based on 3D shape matching.

- Implementation of an easy-to-use "target-less" Camera-to-LiDAR extrinsic calibration method built around a user interface.
- A real-time pixel to point cloud fusion algorithm.
- The proposal of an accurate, and real-time ground classifier for point cloud that works regardless the type and number of LiDARs.

5.2 Related Work

Extrinsic calibration can be defined as the process of calculating the relative position in space between sensor coordinate frames. This can be achieved by looking for co-observable features in the data from both sensors. Calibrating different sensors may use similar optimization methods. However, the features to search for are dependent on the sensor. In the following subsections we will summarize some of the most recent developments in each area.

5.2.1 LiDAR-LiDAR extrinsic calibration

To the best of our knowledge, at the time of writing, very few publications can be found that address the calibration of multiple multi-layer LiDAR sensors. In [43] a method to calibrate a single-layer LiDAR is presented. This work maximizes the mutual information entropy, through the estimation of the projection coefficients between the spaces.

The work presented in [44], shows a semi-automatic extrinsic calibration method for multiple LiDARs and cameras. In this work, features are inserted in the field of view of the sensors with the help of spheres. These are detected with the help of the PointCloud Library (PCL)[89] segmentation and sphere fitting toolbox. The rigid body transformation is calculated using the Iterative Closest Point (ICP) algorithm[45].

5.2.2 Camera-LiDAR extrinsic calibration

While there is extensive work in this field, most of it focuses on the calibration of a single LiDAR and a camera. In this subsection we can classify the available methods into two:

1. The target category requires predefined and specific setups to ease the identification of shared features between the sensors. Under this class we can find notable mentions such as [90], [46], [47], [48], [44], and [91], which take advantage of the purposely inserted features, fixing the number of targets and optimizing the calibration algorithm under these constraints. The work presented by [44], [90], and [91] require the construction of a setup consisting of several specific targets, and scatter them across the shared field of view of the camera and the LiDAR. This approach works well in laboratories and closed environments. However, its application is difficult in the field. Moreover, all the mentioned work, except in [91], are not open-sourced, reducing their impact and reachability.

2. The target-less methods focus on finding inherent features to the scene in both sensors. In this category [49] is a notable mention. It requires an initial rough position estimate between the sensors, provided by the user. In this position, it generates an image projection of the LiDAR reflectance values using the given camera projection matrix. It then slowly modifies the transformation to try to match the generated reflectivity image with the camera gray-scale image. To measure the error, it compares the camera brightness histogram, and the LiDAR's reflectivity histogram. Since this method relies completely on the reflectivity values, it requires manual pre-calibration of the LiDAR unit[50], instead of using the parameters given by the manufacturer. This extra step involves the use of specific equipment, making this method difficult to deploy and test in practical situations.

Table 5.1 presents a feature summary comparison of the Camera-LiDAR calibration methods mentioned. From this we can compare characteristics such as integration, required target type, and sensing devices. After a quick analysis, we can identify the need of a method that works with different types of cameras and LiDARs, while maintaining the target-less property. Target-less methods offer the possibility to calibrate without the need of a special setup, or target. This feature allows users to reduce the time required to obtain the calibration parameters.

Method	Open-source	Data Available	ROS Compatible	Target Type	LiDAR Type	Camera Type
Geiger, et.al [51]	X	O	X	Chessboard	Rotating	Single
Naroditsky, et.al.[46]	X	X	X	Blackboard	Rotating	Single
Velas, et.al. [47]	O	X	O	Custom board	Rotating	Single
Weimin, et.al.[91]	O	O	X	Chessboard	Rotating	Single
Pandey, et.al.[49]	O	O	X	Target-less	Rotating	Omni
This work	O	O	O	Target-less	any	any

TABLE 5.1: Feature comparison among tested Camera-LiDAR algorithms.

5.2.3 Camera-LiDAR fusion

Having both sensors extrinsically calibrated, we can perform fusion at a point-to-pixel level. This enables both sensors to complement each other. The camera can integrate distance information for each of the 3D points projected on the image, while the LiDAR can extract color information to each of the points projected onto the camera field of view. The work presented in [92] achieves fusion using distinct shots taken with a single camera pointing towards the LiDAR field of view. Having the camera intrinsics known and using photogrammetry techniques, they obtain the colored point cloud. However, this approach requires several pre-computations and is not capable of real-time performance. A different approach can be found in [47]. In this work, the authors integrate the intrinsic and extrinsic calibration over several phases. Nevertheless, they neither present the fusion method nor their findings on computation time.

5.2.4 Point Cloud Ground Classifier

The last part of our calibration framework involves the ground removal. This step is essential to ease the identification of obstacles on the road. Most work on this field ([93], [94], [95], and [96]) is focused on airborne applications, with LiDARs attached to a flying drone and pointing towards the ground. These methods can also be applied to both the rotating and solid-state LiDARs usually found in autonomous vehicles. However, these methods are not designed to be executed in real time.

Ground classification is often confused with road classification. This kind of research is focused on autonomous driving applications. The work by [97], and [98]

present the current state-of-the-art developments. However, these require large labeled datasets, a frequently ignored limitation for its real application. For this reason, most of this work does not perform well when the resolution of the LiDAR sensor is changed, requiring a new dataset.

5.3 Theory and Implementation

In this section, we will explain each of the contributing parts of the framework, the theory behind each one, and its corresponding open-source implementation.

During the remaining sections we will employ the following coordinate frame conventions, which observe the ROS (Robot Operating Systems)[27] standard. In summary:

- All systems are right handed.
- LiDAR coordinates: x axis points forward, y axis extends to the left, and z faces upwards.
- Camera frames follow: z points forward, x extends to the right, and y faces downwards.

5.3.1 LiDAR-LiDAR extrinsic calibration

LiDAR sensors obtain the distance to an object by illuminating it with a pulsed laser, and measuring the time required to reflect and return. The 3D version of these sensors are offered by manufacturers as multi-layered, electro mechanical based, or solid-state devices. In all these cases, the device is equipped with one or more laser transceivers. Either by moving a single or multiple lasers, or pulsing the array constantly, the device generates a 3D point cloud, instead of a single 2D scan. This allows the device to generate 3D shapes with high resolution. Taking advantage of this feature, our LiDAR-LiDAR extrinsic calibration is based on a shape-based approach. Moreover, as originally shown in [99], the transformations obtained by this approach produce considerably lower error, while also performing up to three times faster than point-to-point based algorithms (i.e. ICP).

LiDAR-to-LiDAR calibration can be applied in a variety of sensor configurations. For example, new LiDAR technologies provide faster readings, reduced cost, non-mechanical solutions, velocity reading included on each point, etc. However, these sensors tend to have a narrower field of view, compared to their traditional mechanical counterparts. For this reason, extrinsic calibration is required when using several of these sensors. Another setting for multi-LiDAR application would be the integration of multiple LiDARs on bigger vehicles, such as SUVs, trucks, buses, etc., where a single rotating LiDAR would not be able to cover the entire field of view.

Theory

In order to extrinsically calibrate the sensors, both must have shared features as suggested by [43]. Our strategy employs the Normal Distributions (ND) method [100], [99] to match shapes, instead of a direct PDF (Probability Density Function) projection matching as presented in [43]. The ND algorithm tries to successively find the rigid transformation (Equation 5.1) between the b and a point cloud coordinate frames where R_{ab} is a rotation matrix, represented by the rotation on the three coordinate axes as shown in Equation 5.2. The translation vector between these two spaces is represented by t as shown in Equation 5.3.

$$p_a = R_{ab} * p_b + t_{ab} \quad (5.1)$$

$$R(\alpha, \beta, \gamma) = R(\alpha) \times R(\beta) \times R(\gamma) = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{pmatrix} \quad (5.2)$$

$$t = (t_x \ t_y \ t_z) \quad (5.3)$$

To solve this problem, and assuming we have an initial estimate pose (x , y , z , $roll$, $pitch$, and yaw), the ND algorithm uses the Newton algorithm to maximize Equation 5.4, when comparing the point clouds we wish to align, where w , p_i , and Σ_i are the desired pose (matching parameter), the mean vector, and the covariance matrix respectively.

$$Fitness(\mathbf{X}, w) = \sum_i^{N-1} \exp\left(\frac{-(x_i - p_i)^T \Sigma_i^{-1} (x - p_i)}{2}\right) \quad (5.4)$$

This calculation involves: 1) the partitioning of the point cloud P_a in smaller k regions, called Voxels. Voxels are quantized versions of a point cloud, obtained by dividing the space into cubic lattices. The points are assigned to the appropriate Voxel, by rounding the coordinate value of each point p_i ; 2) The computation of the normal vectors $n_{k,i}$ for each of the M_k points inside the Voxels; 3) The probability distribution for the resulting normal vectors, which requires the calculation of the average p and covariance matrix Σ_k as shown in Equations 5.5 and 5.6, respectively.

$$p_k = \frac{1}{M_k} \sum_{i=0}^{M_k-1} x_{ki} \quad (5.5)$$

$$\Sigma_k = \frac{1}{M_k} \sum_{i=0}^{M_k-1} (x_{ki} - p_k)(x_{ki} - p_k)^T \quad (5.6)$$

Finally, the normal distribution for each Voxel is estimated as:

$$e(x)_k \approx e^{\left(\frac{(x - p_k)^t \Sigma_k^{-1} (x - p_k)}{2}\right)} \quad (5.7)$$

Implementation

Our implementation follows the original design by [99], integrated in the PCL (Point Cloud Library) library [89]. The parameters we used are: 0.1 and 1.0 meters for the voxel sizes on the downscaling step, for the point clouds to be aligned; the termination value for the score as 0.01 (ϵ); the step size for the optimization as 0.1; and 400 as the maximum number of iterations before forcing the termination of the algorithm when the algorithm cannot converge. As defined in [99], the algorithm requires a rough estimated initial pose. In summary, the inputs required by our tool are:

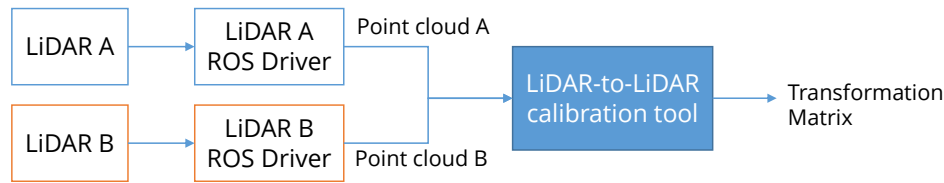


FIGURE 5.2: High-level diagram of the LiDAR-to-LiDAR calibration tool.

- The point clouds generated by the LiDAR sensors to be aligned. These can be obtained directly from the sensor, in an online fashion, or from previously recorded data.
- An optional voxel size for both point clouds. A larger value will accelerate the process, but will decrease the accuracy due to a larger quantization.
- A rough estimation of the transformation between the sensors, in the form $(x, y, z, roll, pitch, yaw)$.

The LiDAR-to-LiDAR calibration tool is implemented as a ROS node. This approach allows us to add an abstraction layer between the sensors and the algorithm, enabling our node to support any kind of LiDAR sensor, as long as we can add its respective driver. Figure 5.2 shows a high-level diagram of the node.

The resulting transformation matrix is then registered in the ROS Transformations Tree (TF). The TF eases the conversion between the coordinate frames in a synchronized manner. In the specific case of LiDAR-to-LiDAR calibration, this transformation is static. This means that the transformation will not change over time, since both sensors are fixed to the vehicle’s chassis.

5.3.2 Camera-LiDAR extrinsic calibration

We decided to implement our method as a semi-automatic calibration tool. The user is required to select the corresponding points between the image and point cloud, using a point-and-click approach. This method removes the necessity to setup a special room, or create particular markers or targets to identify the corresponding points. Moreover, this allows our method to work with not only RGB cameras, but also other types of projective cameras, such as infrared ones.

Theory

The Camera-LiDAR extrinsic calibration follows the same idea as the LiDAR-LiDAR extrinsic calibration. Find the relative transformation between both spaces with the help of shared features between spaces. However, in the case of cameras and LiDARs, these do not represent the data in the same number of dimensions. To find the relationship between them, we need to add an extra step. Knowing that LiDARs represent the points in an orthogonal 3D space using euclidean coordinates (x, y, z) ; and that cameras represent 2D points in a perspective space (u, v) , we can relate these spaces through a linear transformation in the form:

$$p_{cam} = P * R * t * p_{lidar} \quad (5.8)$$

where:

p_{cam} represents the final projected points in image space.

P is the camera projection matrix, also known as the camera intrinsics, or the intrinsics parameters.

R is the rotation matrix on 3D space as mentioned in Equation 5.2.

t is the translation vector between the camera and the LiDAR, described on Equation 5.3.

p_{lidar} is the LiDAR point cloud in 3D space.

Multiplying all of these, we derive the following matrix:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \quad (5.9)$$

Equation 5.9 represents the complete relationship between the LiDAR and the camera spaces, where u and v represent the coordinates in pixel space, and x, y and z represent the 3D space in the LiDAR frame.

To obtain the transformation parameters m , we need to solve the generated system of equations. However, it is important to note that knowing in advance the intrinsic parameters of the setup helps to simplify this problem. Knowing the projection parameters, we would only need to find the Euler rotations (α, β, γ) and the

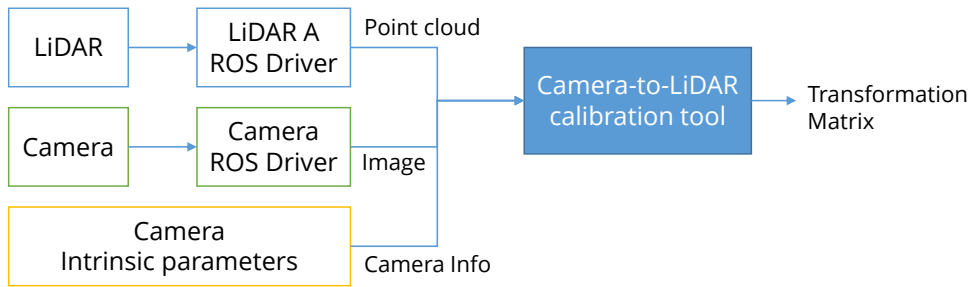


FIGURE 5.3: High-level diagram of the camera-to-LiDAR calibration tool.

translation vector t . Since our objective is to obtain the extrinsic calibration we will assume the intrinsic matrix P is known. To obtain these parameters we followed the well-established intrinsic calibration methods integrated in OpenCV[101].

Having simplified the parameters calculation, we proceed to estimate the pose between the intrinsically calibrated camera to the LiDAR sensor. This problem is commonly known as Perspective-n-Point (PnP). To solve it we employed the Efficient PnP method[102]. This algorithm estimates the R and t matrices minimizing the re-projection error using the Gauss-Newton method, given the corresponding 2D-3D points (in our case, p_{cam} and p_{lidar}), and the camera intrinsics (P). Even if the method requires as low as five points to calculate the parameters, we ask the user to provide at least nine corresponding points, as shown in [102].

Implementation

As previously mentioned, to feed the point correspondences between the points in LiDAR space and camera space, we developed a simple, yet effective point-and-click user interface (UI) solution. This UI shows the image and the point cloud in parallel. The user can quickly identify the correspondences between image and the projected point cloud by clicking on the screen. This approach removes any specific setup or marker prerequisite, and enables our method to work virtually anywhere, as long as the user is able to identify features between the image and point cloud.

Similarly to the LiDAR-LiDAR calibration tool, the camera-LiDAR extrinsic calibration is a ROS node. It can connect to any camera and LiDAR supported by the system. Figure 5.3 shows the components of the calibration tool.

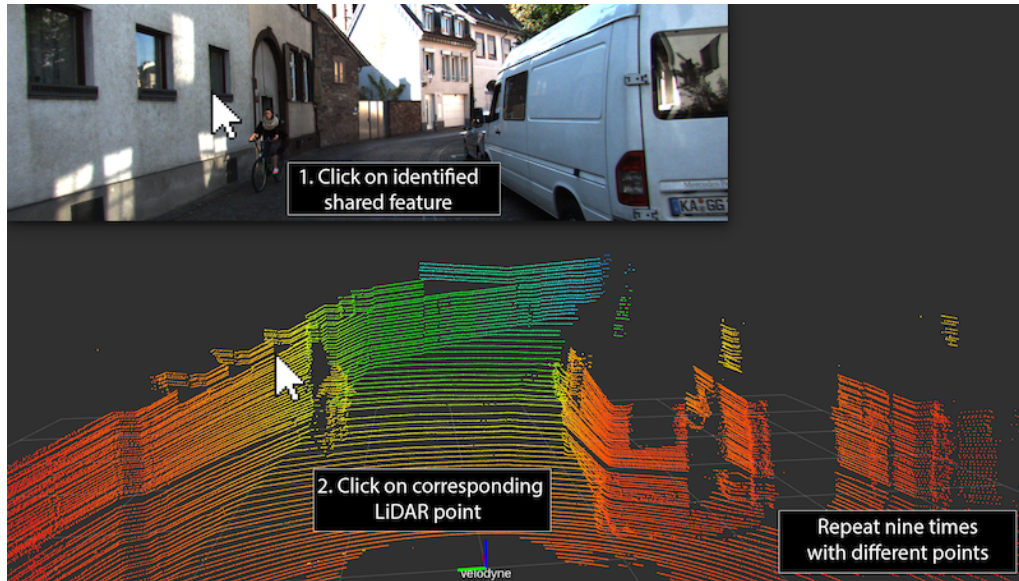


FIGURE 5.4: UI presented to the user to ease the selection of corresponding points.

Figure 5.4 presents the user interface displayed to allow the selection of the corresponding points between 3D and 2D space.

Finally, the node is designed to work with both a real sensor or use previously recorded data. This feature allows the selection of points located at different instants in time, easing the selection of better matching features between both sensors.

5.3.3 Image-Cloud fusion

With the Camera-LiDAR extrinsic calibration complete, we can fuse the data from both sensors. The range information from the LiDAR can be projected to the 2D space. Similarly, the color information contained in the image can be back-projected to the LiDAR space. Figure 5.5 illustrates this concept. The point cloud projected to the image is displayed using a color map representing the distance between the LiDAR and the object being observed. The red color represents a closer distance, while a blue color represents a further one. This kind of point cloud representation is specially useful when detecting and tracking objects [86], [87], [88]. For this reason, we decided to present an open real-time solution to this problem. Our approach uses a hashing function to temporarily store the 3D-2D correspondences. This enables us to back-project the color information in constant time $O(1)$, at the cost of $O(n)$ storage space.



FIGURE 5.5: Result of the real time Image-Cloud fusion using a Velodyne HDL-64 and a wide-angle camera.

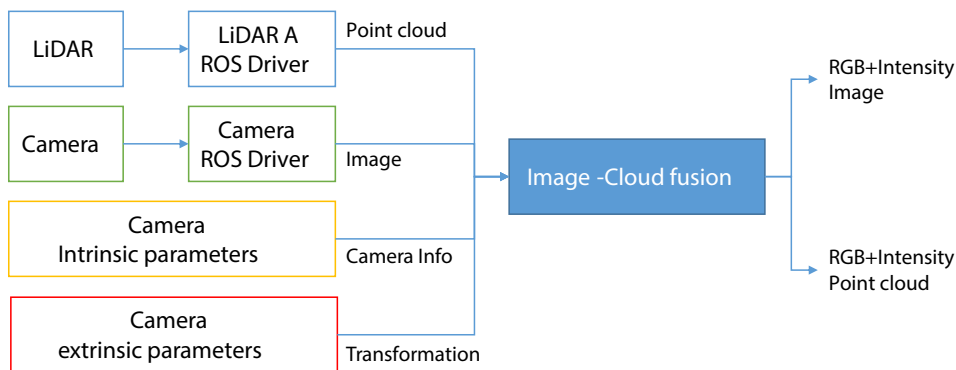


FIGURE 5.6: High-level diagram of the image-cloud fusion.

Theory

With the help of Equation 5.8 and having known the matrices P , R and t from the calibration steps, we can obtain the correspondence between the LiDAR points and the image pixels. The distance for each point is calculated on the ground plane as $\sqrt{x^2 + y^2}$.

In order to back-project the color information from the image to the LiDAR's point cloud, we use an unordered map data structure. We define the unordered map as a list of ordered pairs $\langle (u, v), (x, y, z) \rangle$, that use the point coordinates in both spaces as the hashing function for indexing. Defining the map this way, allows us to find the correspondences in constant time $O(1)$.

The implementation of the node was done following the ROS messaging standard. An outline of the node is shown in Figure 5.6.

5.3.4 Ground classification

The final component of the framework presented in this work is a ground classifier for point cloud data. We call it the Ray Ground Filter. It is a binary classifier (*Ground* or *No-Ground*) for the points in the point cloud. As previously mentioned in Section 5.1 and 5.2, this step is important to ease the implementation of higher level perception methods, since it determines the points considered as an obstacle for its classification and tracking.

Theory

The first phase of the Ray Ground Filter algorithm involves converting the point cloud from Euclidean space to an alternative quantized polar space. The new space is partitioned into a fixed number of radial dividers or rays. The steps involved in this conversion are:

1. Remove the points which are above the height of the LiDAR h_{lidar} . It is important to remove objects above the vehicle that are not considered obstacles (i.e. bridges, signals, etc.).

$$P_e = \{(x, y, z) \in \mathbb{R} \times \mathbb{R} \times \mathbb{R} \mid z < h_{lidar}\} \quad (5.10)$$

2. Convert the points from 3D Euclidean space (x, y, z) to 2D Polar space (r, θ) , while also storing the point's height (z) and a reference to the original point, where $\theta \in [0, 2\pi)$.

$$\begin{aligned} P_p &= \{(\theta, r, z)\}, \text{ such that} \\ \theta &= \text{atan}(y, x) \\ r &= \sqrt{x^2 + y^2} \\ z &= z \end{aligned} \quad (5.11)$$

3. Quantize θ into N radial dividers.

$$N = \text{ceil}(2\pi/\alpha * i) \mid \alpha \in [0, 2\pi] \quad (5.12)$$

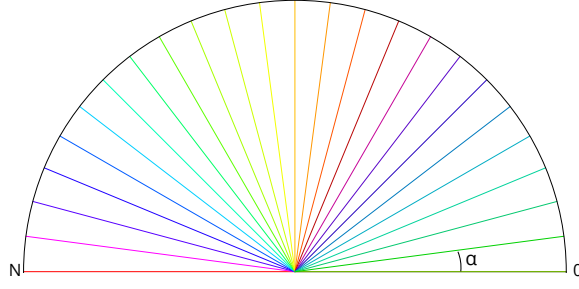


FIGURE 5.7: Diagram of the radial dividers (rays) used in the Ray Ground Algorithm. Showing only 0 to π for simplicity.

where α is a value decided by the user. We use a 100th of a degree ($1.74 \text{ e-}04$ rad).

4. Create a list for each of the N radial dividers.
5. Assign the points to the quantized angle and add it to the list. Figure 5.7 shows each the radial dividers generated and differentiated by color.

$$\forall p \in P_p, n = \text{floor}(p_\theta/\alpha) \quad (5.13)$$

6. Order all the points inside of each of the rays (unordered lists).

At this point we achieve a new point cloud formed by N lists. Each one of these lists contain points of a new type $(x, y, z, i, r, \theta, n)$, where (x, y, z) are the original coordinates; i represents the intensity of the laser reflectivity; θ is the angle in polar coordinates; and n is the index of the quantized radial divider or ray to which this points belong; and k is the number of points in each of the ordered rays.

With all the points organized and ordered, we proceed to classify them into two classes *Ground* and *No-Ground* using a triangle geometry, as shown in Figure 5.8. The adjacent side of the triangle is defined as the distance (d_i) between two consecutive points in the ray.

$$d_i = r_i - r_{i-1}, i \in (0, k] \quad (5.14)$$

The first point in the ray is classified using the same geometry shape. However, the initial point is a virtual one located below the LiDAR sensor h_{lidar} . The opposite

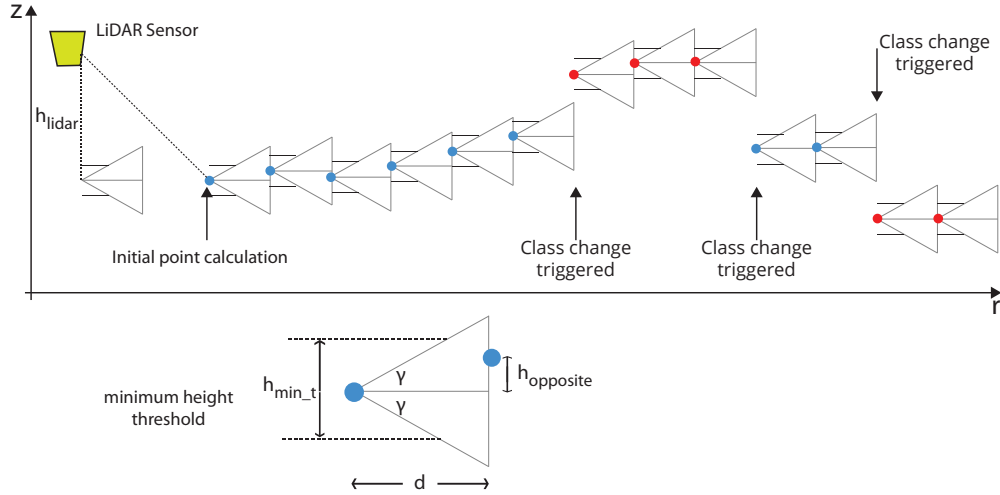


FIGURE 5.8: Ray Ground Filter classification diagram following triangle geometries inside a ray. The points shown in blue color were classified as ground, while the red ones as obstacles.

side is calculated according to a given angle threshold (γ), while d and γ define the maximum height difference to which the next point in the ray might be ($h_{opposite}$).

If the next point falls inside the triangle geometry, the point is considered to be of the same class as the previous point.

$$\begin{aligned}
 h_{i,opposite} &= \tan(\gamma) * d_i \\
 \text{if } h_{i,opposite} &< h_{min,t}, \\
 \text{then } h_{i,opposite} &= h_{min,t}
 \end{aligned} \tag{5.15}$$

Otherwise, it triggers a change of class. However, the points located closer to the sensor tend to be adjacent due to the laser arrangement, triggering a class change on uneven terrain. For this reason, if $h_{opposite}$ is less than a given value $h_{min,t}$, the change is not generated (Figure 5.8).

If a change of class is detected when the previous class was *No-Ground*, then the point needs to be re-classified to ensure it is *Ground*. To achieve this, we re-calculate the geometry of the triangle using the distance between the last point classified as *Ground* and the current one.

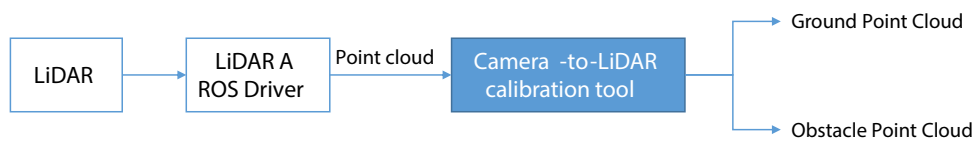


FIGURE 5.9: High-level diagram of the Ray Ground Classifier component.

Implementation

The classifier was implemented as a ROS node. The node can receive a point cloud from any LiDAR sensor. It requires as inputs:

- A point cloud of type (x, y, z) or (x, y, z, i) .
- The height of the LiDAR sensor h_{lidar} .
- The quantization angle for the radial divisions α .
- The angle γ which defines the maximum height of a consecutive point.
- The minimum height threshold between points h_{min} .

To accelerate the processing, the conversion from (x, y, z) to (θ, r, z) , and the quantization is performed in parallel. The resulting radial dividers are stored in N vectors. If the host architecture supports OpenMP, the node will perform conversion and sorting on each one of the CPU cores, reducing the load and accelerating execution.

Figure 5.9 presents the high-level architecture design of the classifier. Finally, the result of ground classification on the KITTI dataset on a Velodyne HDL-64 can be seen in Figure 5.10. Blue colored points denote the ones classified as ground, while the red points indicate those belonging to an obstacle.

Due to nature of the algorithm, it can virtually work with any type and/or number of LiDARs. It also does not require a large dataset for a training phase. Moreover, the algorithm was implemented completely using the CPU, allowing it to work on low-power and embedded systems without a GPU.

5.4 Evaluation

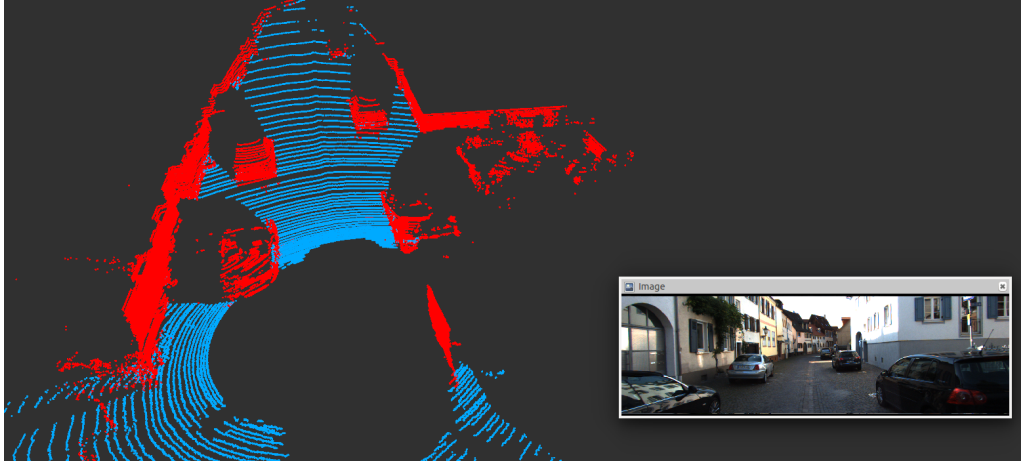


FIGURE 5.10: Ray ground filter on KITTI dataset. Image shown only for reference.

5.4.1 LiDAR-LiDAR extrinsic calibration

LiDAR to LiDAR evaluation using well-established resources is difficult. Currently there are no public datasets including the assessment of multiple LiDARs. For this reason, we decided to prepare two vehicles with different settings, obtain the ground truth, and evaluate our method.

The first setup is built on top of a Toyota Prius sedan. We mounted a rotating LiDAR (Velodyne HDL-64, with a 360 degrees field of view), and four low-cost smaller Micro-Electro-Mechanical-based Scanning LiDARs (MEMS). Each MEMS LiDAR had 140 degrees of horizontal field of view. To obtain the position and Euler rotations, we used a chessboard pattern as a guide to manually identify points between each of the LiDAR frames using the changes in intensity. Having identified the corresponding points, we calculated the rigid-transformation matrix between the rotating LiDAR, and each of the solid-state ones using the Singular-value decomposition (SVD) method. The manually obtained measurements are shown in Figure 5.11.

The second setup consisted of five rotating Velodyne HDL-16 LiDARs mounted on a Toyota Alphard minivan. The ground truth was calculated in the same way as in the Prius setup. Figure 5.12 shows the setup and the obtained measurements.

Using our LiDAR-to-LiDAR extrinsic calibration tool, we collected a database of 20 different experiments. The average resulting parameters are shown in Table 5.2 and Table 5.3, for the Prius and Alphard setups respectively. Figure 5.13 displays

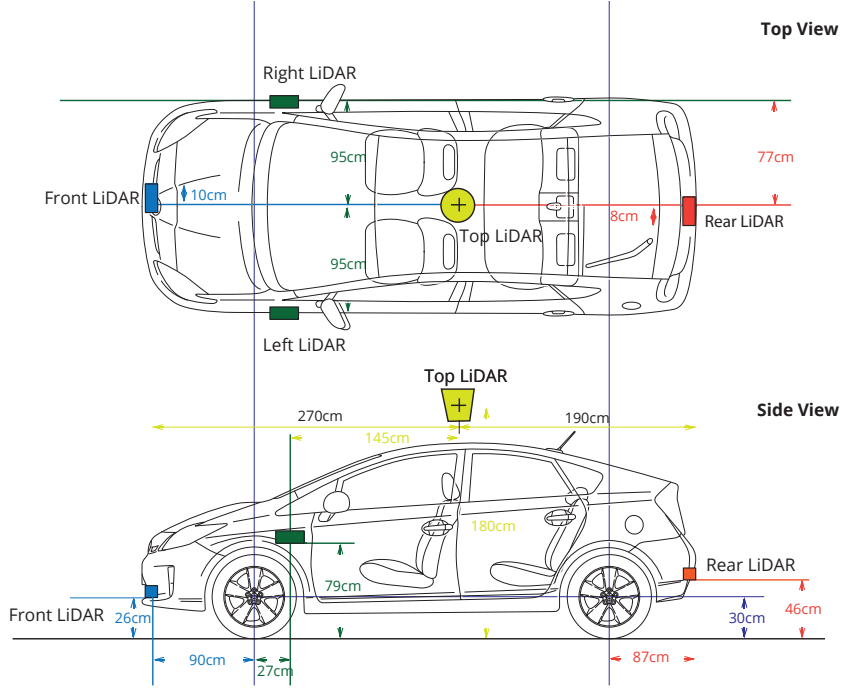


FIGURE 5.11: Manually obtained measurements used as ground truth to evaluate our method. Prius setup, Velodyne HDL-64 and four narrow MEMS LiDARS.

the average accuracy and standard deviation for the calculations compared to the manually obtained points.

TABLE 5.2: Prius Setup. LiDAR-LiDAR extrinsic calibration quantitative results between each of the narrow field of view LiDARs, and a rotating LiDAR. All units are in meters. Reporting absolute error.

Parameter	Rotating to Front LiDAR			Rotating to Left LiDAR		
	Ground Truth	Our method	Error	Ground Truth	Our method	Error
X [meters]	2.70	2.67724	0.0228	1.450	1.4387	0.0113
Y [meters]	-0.10	-0.10886	0.0089	0.950	0.9574	0.0074
Z [meters]	-1.54	-1.56121	0.0212	-1.100	-1.0274	0.0726
α [rads]	1.57	1.5708	0.0008	3.140	3.096	0.0440
β [rads]	0.00	0.0074	0.0074	0.000	-0.0537	0.0537
γ [rads]	1.57	1.5708	0.0008	1.570	1.5243	0.0457
Parameter	Rotating to Right LiDAR			Rotating to Rear LiDAR		
	Ground Truth	Our method	Error	Ground Truth	Our method	Error
X [meters]	1.450	1.44314	0.0069	-1.90	-1.8782	0.0218
Y [meters]	-0.950	-0.9678	0.0178	0.08	0.0737	0.0063
Z [meters]	-1.100	-1.1095	0.0095	-1.34	-1.3665	0.0265
α [rads]	3.140	3.13085	0.0091	1.57	1.5708	0.0008
β [rads]	-3.140	-3.1368	0.0032	3.13	3.1153	0.0147
γ [rads]	-1.570	-1.5773	0.0073	-1.57	-1.5708	0.0008

Finally, the extrinsic calibration obtained using our method can be seen in Figure 5.14 and 5.15 for the Prius and Alphard setups respectively. For the Prius Setup, white points represent the point cloud belonging to the rotating LiDAR, while the

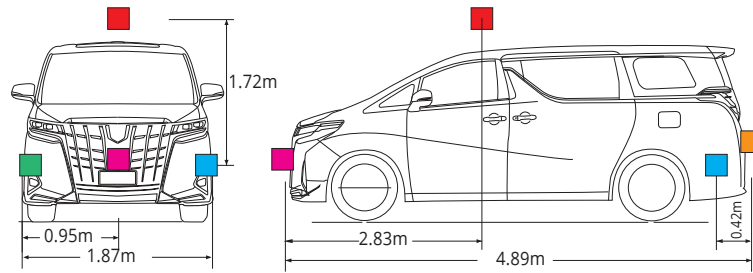


FIGURE 5.12: Manually obtained measurements used as ground truth to evaluate our method. Alphard setup, Five Velodyne VLP-16.

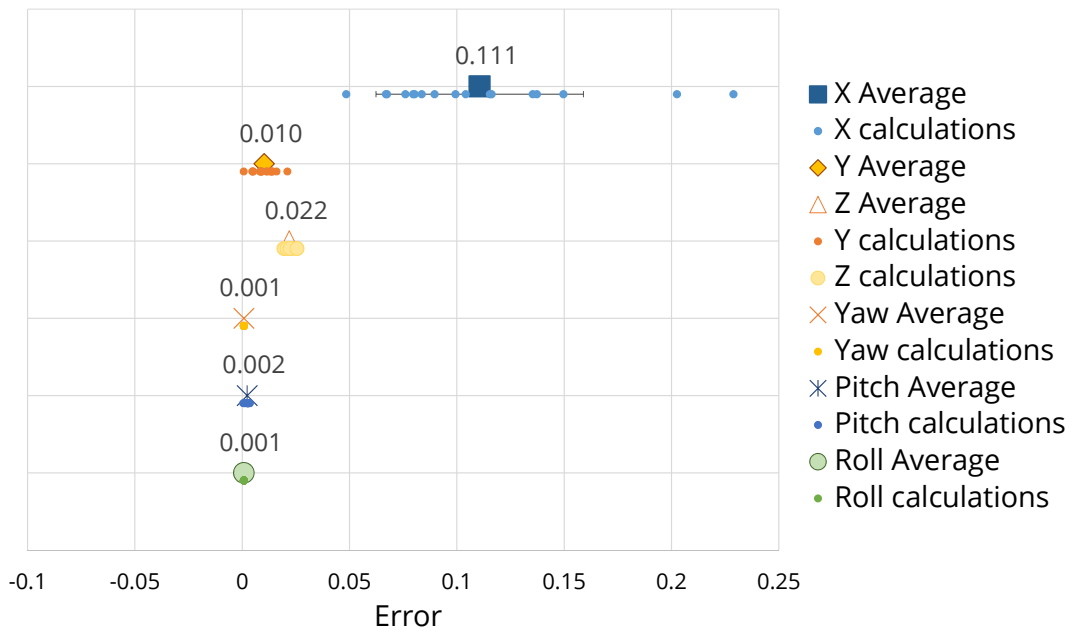


FIGURE 5.13: Prius Setup. Average absolute error for 20 repeated calculations of the extrinsic parameters between LiDARs. X, Y, Z error is reported in meters, while Yaw, Pitch, Roll in radians.

colored ones belong to the other low-resolution MEMS LiDARs. In a similar way, for the Alphard setup, white points represent the point cloud generated by the top rotating lidar, while the colored ones belong to the other rotating sensors located to the front, left, right, and rear of the vehicle.

5.4.2 Camera-LiDAR extrinsic calibration

To evaluate our method, we took as reference the regarded KITTI dataset[51]. The authors of the dataset calibrated their sensors using the method described in [90]. This approach is used in other work as standard baseline.

TABLE 5.3: Alphard Setup. LiDAR-LiDAR extrinsic calibration quantitative results between each the top LiDAR, and a five rotating LiDAR covering blind spots. All units are in meters. Reporting absolute error.

Parameter	Top to Front LiDAR			Top to Right LiDAR		
	Ground Truth	Our method	Error	Ground Truth	Our method	Error
X [meters]	2.75	2.75793	0.00793	-1.70	-1.7198	0.0198
Y [meters]	0.05	0.04227	0.01227	-0.95	-0.95033	0.00033
Z [meters]	-1.31	-1.31587	0.00587	-1.3500	-1.39294	0.04294
α [rads]	0.00	0.00584	0.00584	0.03	0.0394	0.0094
β [rads]	0.05	0.06284	0.01284	0.00	-0.0032	0.0032
γ [rads]	0.00	0.01505	0.01505	-1.57	-1.5782	0.0082
Parameter	Top to Left LiDAR			Top to Rear LiDAR		
	Ground Truth	Our method	Error	Ground Truth	Our method	Error
X [meters]	-1.74	-1.74005	0.00005	-2.14.	-2.1429	0.0029
Y [meters]	0.93	0.9391	0.0091	-0.05	-0.0453	0.0153
Z [meters]	-1.45	-1.46752	0.016752	-1.23.	-1.23756	0.00756
α [rads]	0.0	-0.0273	0.0273	0.	-0.0267	0.0267
β [rads]	0.0	-0.02206	0.02206	0.	0.0206	0.0206
γ [rads]	1.58	1.58036	0.00036	3.1415	3.14124	0.00034

To calibrate the LiDAR and the camera we randomly selected a frame, as shown in Figure 5.4. Then, we proceeded to point and click the correspondent points in the image and the point cloud. Table 5.4 shows our results compared to the measurements reported by [51]. However, it is important to note that our approach lets the user to select data from as many frames as needed over the time. This helps to ease the matching process over the image field, and at different ranges.

TABLE 5.4: Camera-LiDAR extrinsic calibration results comparison. Units in meters and radians.

Parameter	KITTI Ground Truth	Geiger, et.al.[90]	Our method	Absolute Error
X [meters]	0.27	0.2717	0.3017	0.0317
Y [meters]	-0.06	-0.076	-0.089	0.029
Z [meters]	-0.08	-0.04	-0.0937	0.0137
Yaw [rads]	1.57	-1.5632	-1.5625	0.039
Pitch [rads]	0.0	0.0006	0.0012	0.0356
Roll [rads]	-1.57	-1.5559	-1.5622	0.358

5.4.3 Image-Cloud Fusion

Due to the nature of the back-projection calculation, the results of Image-Cloud fusion depend on the accuracy of both the camera intrinsics, and extrinsics parameters. For this reason, to evaluate this section of the framework, we decided to carry out instead a performance evaluation. We measured the average performance of our tool on four different camera and LiDAR setups. Table 5.5 shows the arrangements

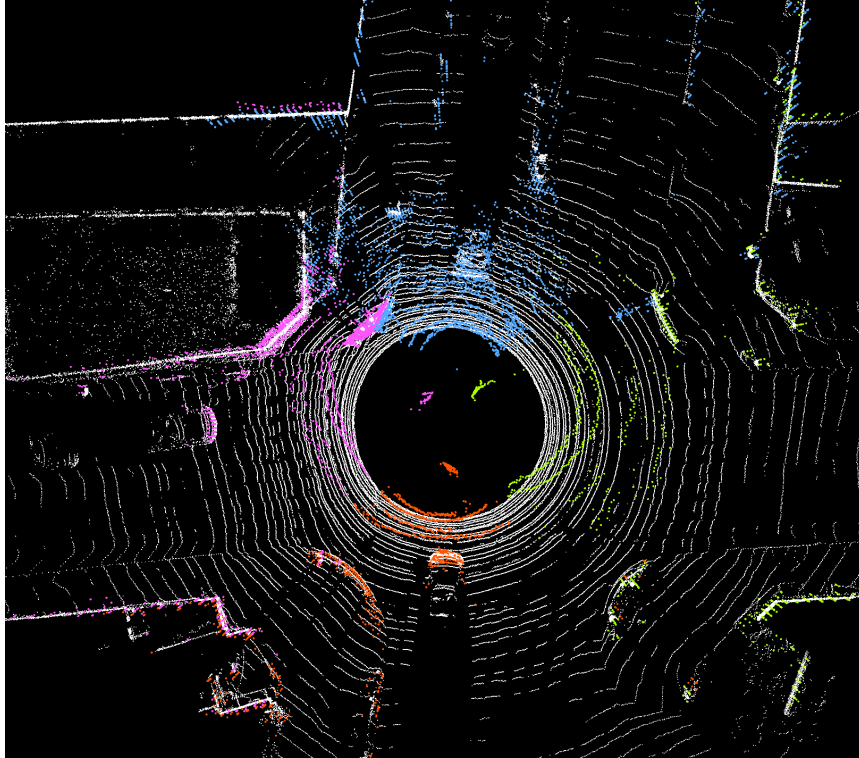


FIGURE 5.14: Prius Setup. Result of extrinsic calibration between rotating LiDAR (shown in white), and four MEMS narrow view LiDAR sensors (displayed in blue, green, pink, and red).

used for this purpose. Figure 5.18 presents qualitative fusion results for each arrangement. All these tests were performed on a desktop computer running Ubuntu 16.04, ROS Kinetic, with an Intel Core i7-6700K CPU with 4 cores, and 16GB of RAM. The node was compiled with OpenMP to distribute the execution on the multi-core system.

TABLE 5.5: Fusion measurements for different sensors and cameras on an Intel Core i7-6700K CPU with 4 cores. A. Baumer VLG-22C@ 20 Hz; B. PointGrey Grasshopper3@15Hz; C. PointGrey Grasshopper3@15 Hz; D. PointGrey Flea2 @ 15Hz

Camera	Image Resolution	Velodyne LiDAR	Point cloud size	Execution (ms)
A	1288x960	HDL-64	~ 120,000	116
B	800x600	HDL-32	~ 70,000	78
C	1384x1036	VLP-16	~ 25,000	50
D	1384x1032	HDL-64	~ 120,000	128

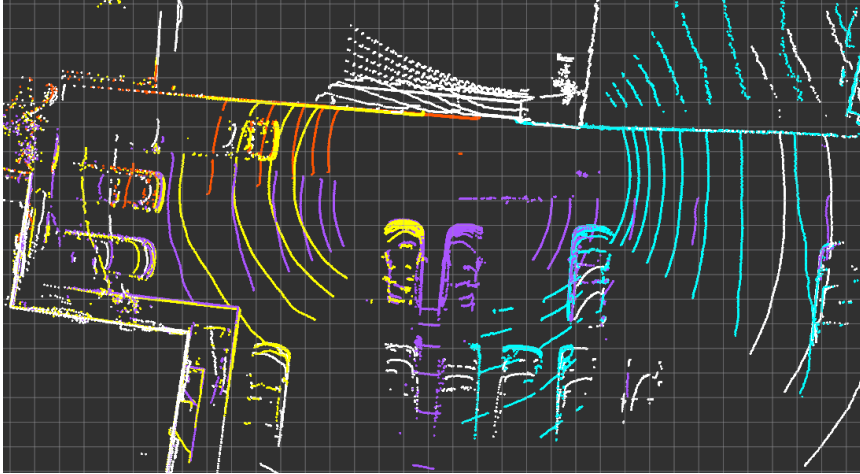


FIGURE 5.15: Alphard Setup, Result of extrinsic calibration of five rotating LiDARs (Top: white; Front: Cyan; Right: Purple; Left: Orange; and Rear: Yellow).

5.4.4 Ray Ground Classifier

To evaluate the classification accuracy, we decided to manually label 12 point clouds. We chose four random point clouds for each of the most commonly used Velodyne sensors: HDL-64, HDL-32 and VLP-16. Each scan was taken from previously recorded log data from various urban settings. For the Velodyne HDL-64 case, we used the point clouds provided by the KITTI dataset. The average results for the point clouds classified can be seen in Table 5.6. In this table we also included the average execution time for each sensor model. The execution time was measured on a desktop computer running on an Intel Core i7-6700K CPU with 4 cores, and 16GB of RAM with OpenMP enabled.

TABLE 5.6: Classification performance of the Ray Ground Classifier on the point clouds by sensor type.

Measurement	HDL-64	HDL-32	VLP-16
Accuracy	0.8247	0.8429	0.8461
TP	0.8429	0.7737	0.6954
FP	0.1952	0.1025	0.0653
TN	0.8047	0.8974	0.9346
FN	0.1570	0.2262	0.3045
Precision	0.8258	0.8560	0.8619
Execution time (ms)	55	20	11

Finally, qualitative results on three different sensors can be seen on Figure 5.19.

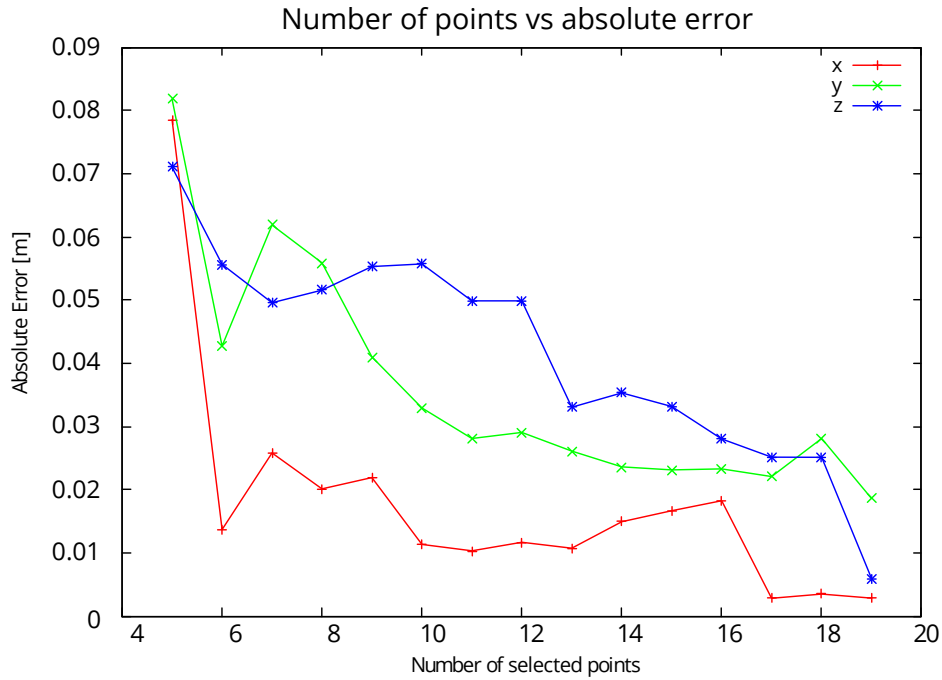


FIGURE 5.16: Translation absolute error compared with number of selected points on the Alphard Setup.

The last sub figure shows the Ray Ground classifier working on four merged Velodyne VLP-16 LiDARs. These were previously calibrated using our method presented in Section 5.3.1.

5.5 Discussion

5.5.1 LiDAR-LiDAR Extrinsic Calibration

Like any other approximation algorithm, the ND algorithm may fail to converge under the maximum number of iterations. This is due to the absence of differentiable features, leading to the cost-function to become insensitive to the rotational parameters. In Figure 5.13, we can observe that when providing a feature rich environment, the ND algorithm can obtain highly accurate results. The error on the x-axis, even if it is considerably low, presents a higher value than the rest of the parameters. While analyzing this difference, we found out that this was caused by an inaccurate internal laser calibration from the manufacturer. Figure 5.2 shows that the points hitting the wall (the x-axis), display an uneven wall surface.

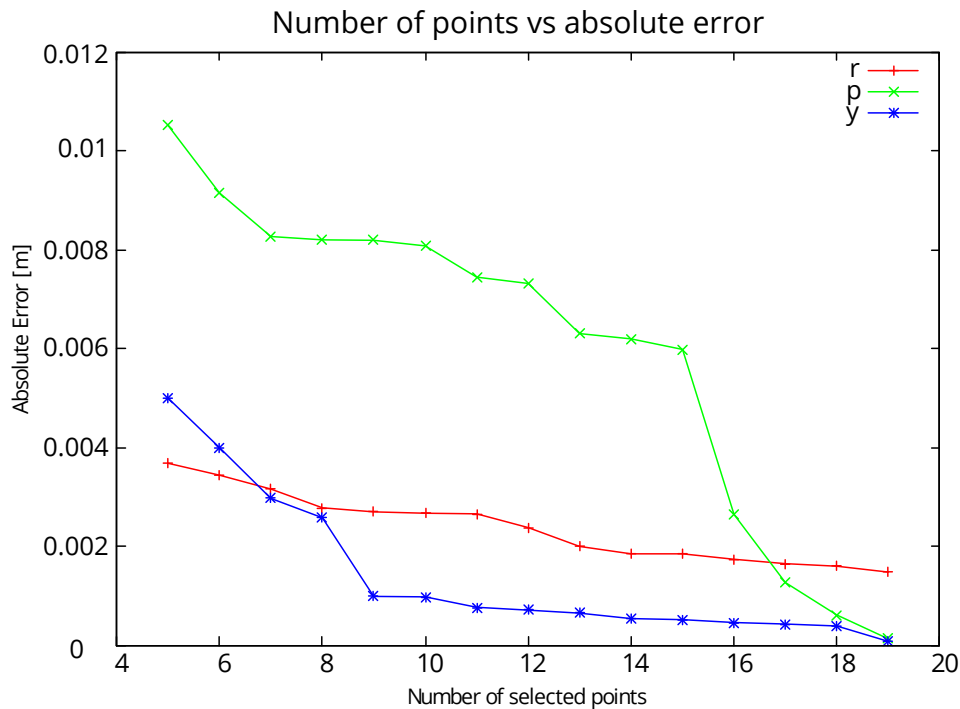


FIGURE 5.17: Rotation absolute error compared with number of selected points on the Alphard Setup.

During our indoor and outdoor experiments, we realized that selecting a complex and/or asymmetric environment leads to a faster convergence. Since our tool shows the calibration result after completing each set of iterations in real-time, the user can quickly identify if the selected setting is appropriate for calibration. Thus, from our experiments we conclude that it is important to have large objects (i.e. walls or other vehicles) in the shared field of view of the LiDARs being calibrated. It is important to note, that our method requires features to be shared between the sensor views, otherwise calibration will not be possible.

When calibrating more than two LiDAR sensors that all share an overlapping field of view, the calibration must be performed $N-1$ times, where N is the number of sensors. In situations when multiple LiDARs only share a field of view with some of the other sensors, the obtained solution might not converge to a local optima. This is because the NDT algorithm is designed to match only two point clouds at a time. To overcome this, one approach would be to complete calibration $N-1$ times, and afterwards repeat calibration $N-1$ times using a different order. If the same solution is obtained, the current solution is very likely to be a global optima. Nevertheless, this cannot be confirmed due to the lack of extra information from other sensors.

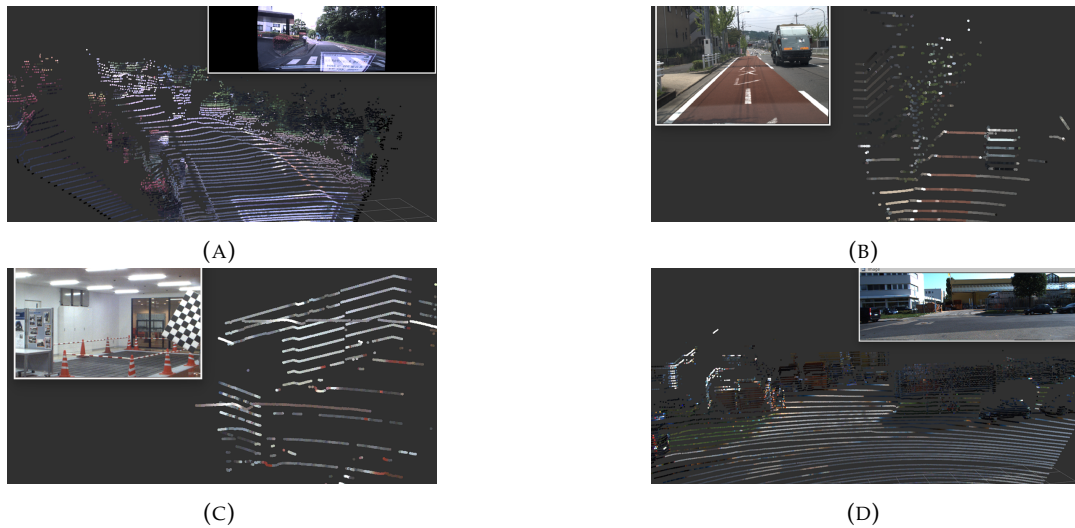


FIGURE 5.18: Fusion results on different setups a) BaumerVLG-22 and HDL-64; b) Grasshopper3 and HDL-32; c) Grasshopper and VLP-16; d) Flea2 and HDL-64 (KITTI)

If a different solution is obtained, then an average between the solutions might be used. To further improve the optimization of the calibration parameters, a bundle adjustment-like approach could be performed. This would allow to tune the individual calibration results to maximize global consistency.

5.5.2 Camera-LiDAR Extrinsic Calibration

As mentioned in Section 5.3.2, our method provides an easy and practical way to quickly obtain the extrinsic calibration between the Camera and LiDAR. It does not require the preparation of a specific setting, or marker of any kind. For this reason, it allows calibration of the sensors while directly connected to them, or the use of previously recorded data. Since our method depends completely on the points fed to the algorithm, its accuracy highly depends on the ability of the user to find proper correspondences. During our experiments we found it is easier to observe the shared features between sensors when there are objects nearby. This is due to the nature of the multi-layered laser and the perspective camera model. After several experiments we defined the following guidelines to ease the calibration process. Users familiar with these, were able to obtain the parameters in as short as one minute.

- 1) Capture a single scene with different objects scattered around the field of view.
- 2) It is easier to identify objects' corners, points hitting lowest and highest points of an identified object in the point cloud.
- 3) Since our method allows the selection of

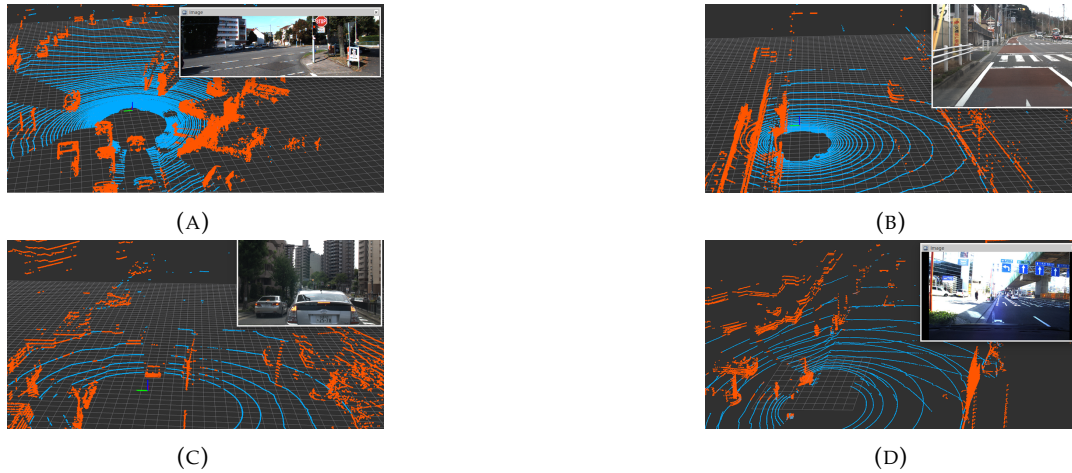


FIGURE 5.19: Ground Classification, using our Ray Ground Algorithm on a) Velodyne HDL-64 (KITTI); b) Velodyne HDL-32; c) Velodyne VLP-16; d) Four Velodyne VLP-16 calibrated. The points belonging to the road are shown in blue, while obstacles are painted in red.

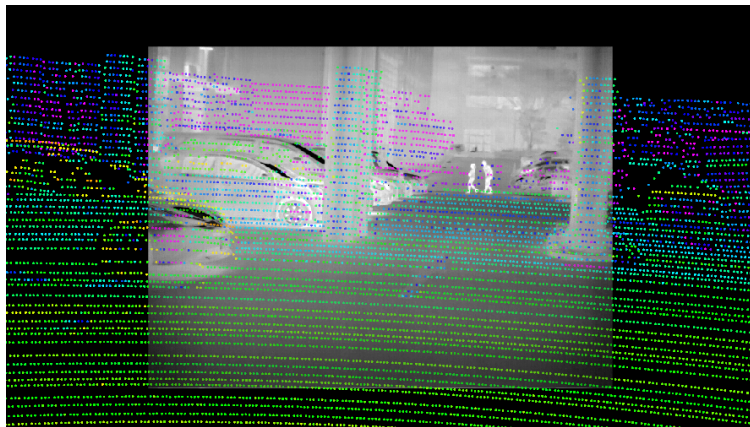


FIGURE 5.20: Successful calibration of a thermal vision camera (FLIR ADK), and a Velodyne HDL-64.

points regardless time of the the frame, using recorded log data eases the selection of these points while the car is running in a urban scenario. Taking advantage of this attribute is highly recommended in cases where a static scenario does not contain enough identifiable features. 4) As shown in Figure 5.16 and 5.17, selecting more points will obtain a reduced error on the translation parameters. However, if the selected features are not easily identifiable, selecting more points will increase the final absolute error.

Finally, an additional advantage of our method is that it also allows the calibration regardless of the image format. Figure 5.20 shows a successful extrinsic calibration between a thermal vision camera and a Velodyne HDL-64.

5.5.3 Image-Cloud Fusion

The image-cloud fusion node is an example of an application of a successful integration of sensor fusion. It complements and integrates distance and color data between the sensors, providing extra information to other perception applications. We consider that this tool will help other researchers in the development of real-time perception systems. As previously mentioned, recent developments based on deep learning require the post-fusion data to be fed to the network [87], [86], [88]. All these techniques perform the data fusion off-line, forwarding it at a later phase. Our method provides a real-time solution to this problem in many applications. It is important to note that the performance might be reduced when using higher resolution images and LiDAR sensors.

5.5.4 Ray Ground Classifier

The Ray Ground Classifier presents an elegant and effective way to classify the ground on complex point clouds. Being a shape and rule based method, is possible to use it on different sensor types and setups. It requires few or none of its parameters to be tuned for successful application. Moreover, being implemented completely on the CPU, it can be ported to other embedded architectures. Our evaluation shows that it achieves a high accuracy across different sensor types as indicated in Table 5.6. Due to its nature, it does not require specific measurements, such as rings numbers or specific intensity ranges, making our approach sensor independent. Figure 5.19 presents qualitative results of the ground classification on four Velodyne VLP-16 LiDAR sensors, previously calibrated with the approach presented in Section 5.4.1.

During our experiments and evaluation, we found out that to obtain better classification results it is important to set correctly the height of the LiDAR (h_{lidar}), and the angle threshold of the maximum slope (γ). The first one is usually known and when it is not, it can be estimated by averaging the first point of each of the rays. The latter can be quickly estimated using the UI provided by ROS, while analyzing the height values in the ground of the point cloud. Finally, we also carried out experiments on steep roads. On these we found out the classifier to be reliable, when

changing the slope parameter. However, we also found that keeping a high slope value on regular roads might cause misdetections.

5.6 Conclusion

We described an open-source multi-sensor fusion toolbox for autonomous vehicles. It is composed of a LiDAR-to-LiDAR extrinsic calibration algorithm, a Camera-LiDAR extrinsic calibration method, Multi-LiDAR fusion, Camera-LiDAR fusion, and a ground classification method.

Our LiDAR-to-LiDAR calibration algorithm successfully adapts a state-of-the-art matching algorithm. Through several experiments, we showed that provides calibration to the centimeter-level accuracy, without requiring a special setup. We also shared recommendations and limitations on Section 5.5.1 to ease the use of our calibration tool.

We introduced an easy-to-use Camera-to-LiDAR extrinsic calibration method. It applies the latest developments on camera pose estimation in computer vision. Instead of requiring the preparation of an specific setup, or the printing of predefined markers, a UI is provided, so the user feeds the corresponding points between the camera and LiDAR space to our algorithm. To accelerate this task, we also shared the guidelines we defined through experimentation in Section 5.3.2. Our method achieved error as low as one centimeter, comparable to other state-of-the-art developments in the field. It is important to note that our method will fail to obtain an accurate calibration if the intrinsic parameters are incorrect. The application of our method becomes challenging when using low resolution LiDARs in combination with telephoto lenses. Users can have a hard time to find shared features between images and point clouds. In these cases, targets might be inserted manually. We found that cones, or highly reflective materials are easy to identify between images and point clouds.

The third part of our framework, presented a real-time image and point cloud fusion. The tool integrates the range data to the image space. It also allows point cloud back-projecting the color information to the 3D space. To the best of our knowledge, it is the first real-time open-source available as the time of writing.

The final part of our framework is an accurate geometric and rule-based ground classifier. Thanks to the nature of the algorithm, and the low number of configurable parameters, it can be used with virtually any kind of LiDAR sensor, including data composed by several LiDARs. Moreover, our method performed on average as fast as 55 ms for the high-resolution Velodyne HDL-64. It averaged 20 and 11 ms on the HDL-32 and VLP-16 sensors respectively, achieving real-time performance on the three tested sensors.

Our multi-sensor fusion toolbox is integrated in the autonomous driving framework known as Autoware[103]. The source code can be downloaded from the following URL <<https://gitlab.com/autowarefoundation/autoware.ai>>. Future work on our toolbox will involve the full automation of the calibration methods. For instance, employing recent developments on the Structure from Motion field. We also plan to implement the fusion and classification algorithms on GPUs to further accelerate its execution.

Chapter 6

Real World Applications

6.1 Introduction

Obtaining precise perception data of the environment surrounding an autonomous robot has shown to impact significantly its performance [104]. This kind of data is especially valuable in areas such as autonomous driving, driving assistance, and self-localization systems [105, 106]. The perception data such as distance, position, the velocity of the robot and nearby objects, may be provided by different sensors, including but not limited to Global Navigation Satellite System (GNSS), Inertial Measurement Unit (IMU), wheel odometry, Light Detection and Ranging (LIDAR) or cameras.

Cameras devices sometimes are left aside in favor of the LIDAR, mainly because the latter can provide data of the whole surrounding area, depth, and position in a precise way. However, object detection through point clouds is not as developed as its image-based counterpart. As pointed by [107], object detection on images have shown a quick development and achieved high recognition rates.

The main drawback of camera based perception systems is that in most cases those provide a narrow angle of view compared to the LIDAR. To tackle this, some camera vendors provide 360 degrees cameras, but those solutions are not cost wise.

An affordable, high performance perception device for autonomous robots would also make possible the development of a wide range of simultaneous localization and mapping solutions. In this respect, since the processing requirements of the camera are well defined, Field-Programmable Gate Array (FPGA) devices can play a key role due to its low price, high dynamism and performance.

Nowadays, FPGAs are devices that more than low priced, their energy requirements have improved through time [108]. Recent series of FPGAs are passing the 16 nm barrier, leading to even lower power consumption. Moreover, these devices are adopting technologies from the Application-Specific Integrated Circuit (ASIC), becoming simultaneously faster. The size of these gadgets is compact enough to be fitted in a small casing. These features, plus the programmability inherent to them, enables the use of these circuits in mobile or portable applications, such as cameras and network devices.

In recent years, camera sensors have quickly become ubiquitous. We can see them in our everyday life: attached to our smart phones, integrated in mobile computers and used in large quantities as part of security systems in all sorts of facilities. Because of this, price of image sensors are constantly dropping. Besides, the size and quality are also quickly improving due to the high demand for these devices.

The work presented in this chapter aims to show the prototype of a 360 degree cost-effective, high performance camera based Cyber Physical System (CPS) for perception. This development provides a full view of the surroundings, using six camera sensors attached to a custom tailored FPGA board. The prototype board is equipped with standardized Camera Interface Specification (CSI) and Gigabit Ethernet ports. This provides future compatibility with newer versions of sensors and transceivers. In addition, being a programmable device, updates to the firmware can be quickly tested.

On the FPGA board, the images are captured by each sensor in RAW format. Afterwards, those are converted into full color ones and transferred to the host computer through two Gigabit Ethernet ports. Finally, the network stream is converted into a bitmap image and processed using a Histogram of Oriented Gradients (HOG) + Support Vector Machine (SVM) based object detector, so it can provide the perception information in a real time fashion.

6.2 Related Work

A few vendors in the market develop devices that can capture images from the complete surrounding of the camera. Unfortunately, most of them are targeted for the

end-user, with small resolutions and they do not provide an Application Programming Interface (API) to communicate with the device. Even if we try to use some of these cameras, those provide no access to the image stream, making its use for real time processing pointless.

The only device with access to the stream and API control we found was the Ladybug camera, developed by PointGrey. This device outputs a spherical view of the surrounding with the help of six camera sensors attached to the body. This device is capable of generating a spherically stitched 30 Mega pixel (MP) image at a rate of 10 frames per second (fps) through a Universal Serial Bus (USB) 3.0 port [109]. Sadly, the cost of the mentioned device is still high for general purpose applications. Furthermore, the camera requires a high-performance host and specific USB hub controllers to reach the mentioned performance, making it difficult for all the configuration environments.

6.3 Design and Implementation of Hexacam

This section presents a prototype system for real time image acquisition and processing of pedestrian detection that may be used in a perception system. The network design of the cyber physical system is illustrated in Figure 6.1, it can be described as follows:

1. Six cameras are connected to a custom made board for the Spartan-6 FPGA, those capture the image data in RAW format from each sensor through a Mobile Industry Processor Interface (MIPI)/CSI port
2. The FPGA packs and sends each image through a UDP port using the Gigabit Ethernet ports on the board.
3. The PC host receives, unpacks, reconstructs and stitches the images into a bitmap.
4. Finally, the algorithm known as histogram of gradients (HOG), used in conjunction with a support vector machine (SVM), obtains the pedestrian localization in the panoramic image.

The last stage is executed in a general purpose graphics processing unit (GPGPU), in this way the complete process can be performed in a real time fashion.

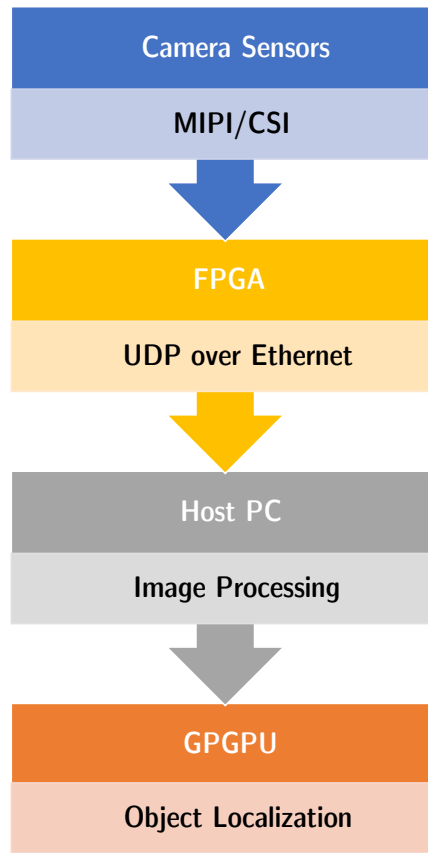


FIGURE 6.1: An overview of our prototype system.

6.3.1 FPGA board

For the prototype, a custom built board with a Xilinx Spartan-6 (XC6SLX100) with no microprocessor is used. All the computation is implemented as hardware logic. As for the camera sensors interfaces, these are connected directly to the FPGA pin out and implemented at hardware level as CSI ports. The network interfacing is done with the help of Ethernet PHY transceivers (Microchip SMSC LAN8700).

The operation unit required for each camera is about 750 slices, 1800 registers and 1800 LUTs. An overview of the logical architecture is shown in Figure 6.3.

To make sure the images are synchronized, the CSI commands to reset, setup the sensors and initialize the stream, are sent simultaneously to the six cameras. Once the data have been acquired from all the sensors, it is sent through the Ethernet port



FIGURE 6.2: Picture of the FPGA board prototype.

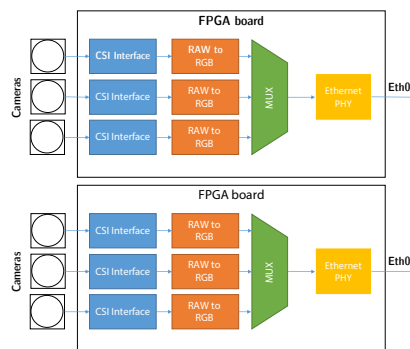


FIGURE 6.3: Board architecture.

in a serial way. To achieve this, each camera stream is connected to a multiplexer. This one selects and sends each image in round robin through the network.

6.3.2 Camera sensors

The custom tailored board includes six MIPI/CSI interfaces, to each one of which, an OmniVision 5647 camera sensor is connected. The cameras have been set in a fixed focus module. This module includes a 5 mega pixel Complementary metal–oxide–semiconductor (CMOS) image sensor which features automatic white balance, exposure, band filter and illumination detection. Moreover, the camera sensor support several image sizes, shown in Table 6.1 [110].

Since we aim to use the Ethernet bandwidth as much as possible, the sensors are configured to a predefined Full HD resolution (1920x1080). The FPGA board

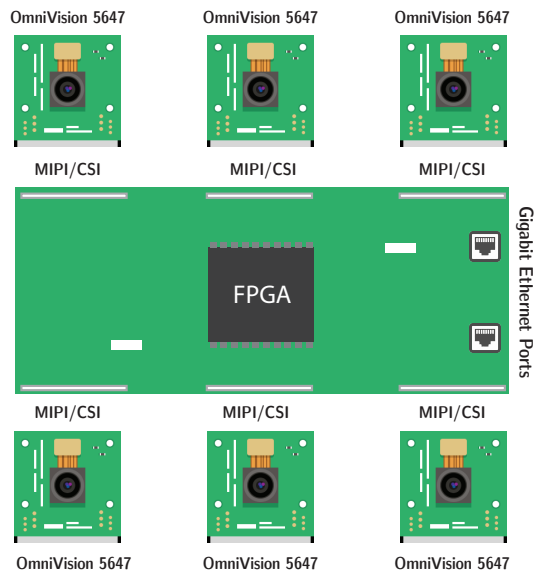


FIGURE 6.4: Custom tailored FPGA board diagram.

TABLE 6.1: Camera sensor supported resolutions and transfer rates

Mode	Resolution	Frame rate
QSXGA	2592x1944	15 fps
1080p	1920x1080	30 fps
960p	1280x960	45 fps
720p	1280x720	60 fps
VGA	640x480	90 fps
QVGA	320x240	120 fps

configures and transfers data with each camera through a CSI interface. In Figure 6.4, an organization of the board is illustrated.

6.3.3 Image processing

The image received on the FPGA is in RAW format, also known as Bayer filter mosaic [111] in its BG2R variant. As it can be appreciated cameras only capture one color per pixel. This layout for arranging colors in the filter is 50% green, 25% blue and 25% red. This is due to the fact that our eyes are more sensitive to the color green.

The process of reconstructing the full color RGB image, from the under sampled Bayer filter data acquired from the camera sensor is known as “demosaicing or demosaicking” [112]. In short, the camera sensor captures 10 bits representing a single pixel. These 10 bits are then converted to a full color RGB image (24 bits). It is worth

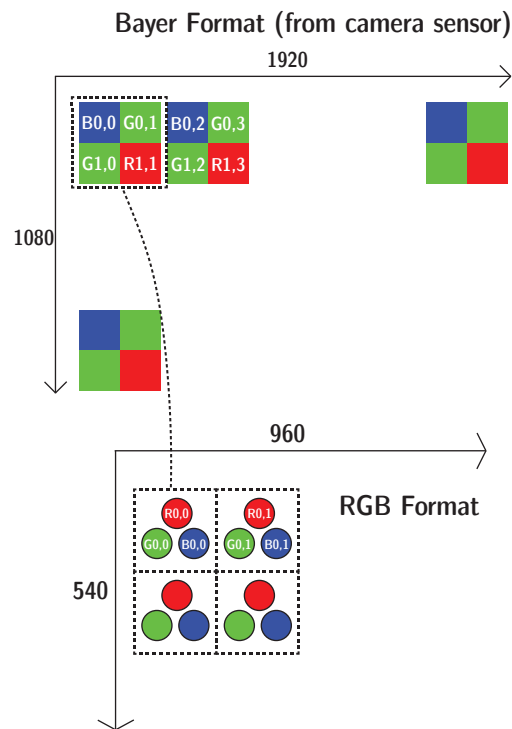


FIGURE 6.5: RGB color reconstruction from RAW image.

noting that the RGB image process reconstruction from the data obtained by each camera sensor will reduce the final resolution by four times due to the Bayer format nature, this can be easily appreciated in Figure 6.5.

There are several other algorithms to perform the demosaicing that eschew the loss of image resolution. To achieve this, those use complex interpolation methods to fill the pixel data. Unfortunately, these algorithms introduce image artifacts depending on the type of pattern found in the captured picture [113] [114]. In the Figure 6.6 is easy to appreciate one kind of the artifacts generated by the interpolation.

Due to the generation of undesired artifacts during the pixel interpolation, and since we will be down sampling anyway the image during the object detection phase. We decided not to use any algorithm to interpolate the missing pixels.



FIGURE 6.6: Left side, original image. Right side, artifacts due to pixel interpolation

If we represent the data in a matrix we get:

$$BayerArray_{1080,1920} = \begin{pmatrix} b_{0,0} & g_{0,1} & \cdots & g_{0,1919} \\ g_{1,0} & r_{1,1} & \cdots & r_{1,1919} \\ \vdots & \vdots & \ddots & \vdots \\ b_{1078,0} & g_{1078,1} & \cdots & g_{1078,1919} \\ g_{1079,0} & r_{1079,1} & \cdots & r_{1079,1919} \end{pmatrix}$$

Since we are not applying interpolation methods, the conversion is defined by the values of the Bayer array, except for the green element, which must be calculated as the average of the two elements corresponding to the Red Green Blue (RGB) pixel. These can be expressed as follows:

$$R_{0,0} = BayerArray[r_{1,1}] \quad (6.1)$$

$$G_{0,0} = \frac{BayerArray[g_{0,1}] + BayerArray[g_{0,1}]}{2} \quad (6.2)$$

$$B_{0,0} = BayerArray[b_{1,1}] \quad (6.3)$$

6.3.4 Network transmission to host

As for the actual process of acquiring images in the FPGA, it can be summarized as follows: Once the board initializes each camera, the sensors start transmitting the

image data continuously through each CSI interface. As soon as the data is on the FPGA, it is then forwarded to the host PC over two Ethernet ports. To achieve this, the image data is split into several User Datagram Protocol (UDP) packets. Two Ethernet ports are used since the network bandwidth available in one port is not enough to carry the image data of the six sensors and achieve at least 10 frames per second. A simple protocol over UDP was created to successfully transfer the image data through the network. Three types of packets were implement to achieve the image transmission: frame start, data packet and frame end.

The *FRAME_START* packet is used to indicate the initial frame corresponding to an image. Its structure is shown in the Table 6.2. An X in the value field means don't care.

TABLE 6.2: FRAME_START packet structure

Offset	Value	Description
0	XXXXXXXX	Packet ID
4	00000000	Data Offset
8	00000000	Data length
12	01000000	Frame Start

The *FRAME_DATA* contains the raw color data of the image acquired from the camera sensor. The structure of the packet is shown in the Table 6.3.

TABLE 6.3: Image reception packet queue

Offset	Value	Description
0	XXXXXXXX	Packet ID
4	XXXXXXXX	Data Offset
8	XXXXXXXX	Data length
12	00XXXXXXXX	Packet Type + Row Number
16	XXXXXXXX	Image Data
...	...	Image Data
16+Length	XXXXXXXX	Image Data

The *FRAME_END* packet is used to indicate that the transmission of data corresponding to the current image is completed. The packet structure is shown in the Table 6.4.

The Algorithm 1 describes how to receive images using the presented packet types.

To illustrate how the network transmission is done, part of an image reception packet queue is shown in the Table 6.5

TABLE 6.4: FRAME_END packet structure

Offset	Value	Description
0	XXXXXXXX	Packet ID
4	00000000	Data Offset
8	00000000	Data length
12	02000000	Frame End

Algorithm 1 UDP Image reception

```

while running do
  if Packet is FRAME_START then
    Create new Image
  else if Packet is FRAME_END then
    Close Image
    Set image as ready to be read
  else
    Store Data in specified Row
  end if
end while

```

Once all the packets are received on the host, these are processed to form the final image. The process is repeated continuously to obtain the image stream from the six cameras on different UDP ports. It can be observed in the Table 6.5, that the sum of lengths is the width of the captured image.

6.3.5 Object detector

The final step in the CPS network is object detection. This phase will extract the perception information contained in the stream captured from the six camera sensors.

For this prototype, a HOG+SVM based pedestrian detector is used to provide the perception information. We decided to use this object detection algorithm since it is robust and mature. Moreover, it has shown a high recognition rate, and due to its iterative nature it is well fitted for parallelization.

In order to obtain the highest possible frame rate from the prototype, our objective became to extract the perception data in no longer than the next frame is captured and transmitted to the host. The initial CPU implementation takes longer and creates a bottle neck. For this reason, a GPU version of the algorithm was required.

The CPU and GPU versions use the HOG descriptor provided by Dalal and Triggs, which is a vector of 64x128 elements previously trained in the INRIA person dataset [115]. The parameters of the HOG detector used during our tests are 64

TABLE 6.5: Packet Queue

Queue	Packet ID	Offset	Lenght	Type
FRAME_START	0001	0000	0000	01000000
FRAME_DATA	0002	0000	0144	00000000
FRAME_DATA	0002	0144	0144	00000000
FRAME_DATA	0002	0288	0138	00000000
...
FRAME_END	021D	0000	0000	02000000

levels in the pyramid, 9 bins for the gradient calculation and a block size of 16. The detector returns a vector of bounding boxes containing the pedestrians detected in the environmental image for each frame.

As specified by Dalal and Triggs in [115] the GPU and CPU implementation follow the workflow shown in Figure 6.7.

A short summary of the process to calculate the descriptor is:

1. The input image is divided into small sub-images known as 'cells'. These cells can be rectangular (known as R-HOG) or circular (C-HOG). In this work we use rectangular cells as describes in the original work.
2. Within each cell, accumulate a histogram of edge orientations.
3. The combined histogram entries are used as the vector descriptor.
4. A normalization between cells is performed to provide a better illumination invariance.

To conduct the object detection, a sliding window is passed through the image. The HOG descriptor is calculated for each of the subregions. Then, the obtained descriptor vector is classified with the help of a pre trained SVM model (given by Dalal and Triggs). Finally, the output score is checked against a threshold to determine if the object is found in the region.



FIGURE 6.7: HOG descriptor calculation process

6.4 Performance Evaluation

This last section will show the result of the quantitative evaluation performed on the CPS prototype for every phase of the system.

6.4.1 Camera sensors

Each of the Omnivision 5647 camera sensors are initially configured to work at a resolution of 1920 x 1080 (Full HD), 10 bit RAW, and 30 fps. In our measurements we were able to obtain 522.96 Mbps from each of the six sensors on the CSI ports. This bandwidth, delivered an effective frame rate of 26.6 fps on each sensor. This do not represent a problem since we know that the transmission stage support a maximum theoretical speed of 1 Gps on the Ethernet port.

6.4.2 Network speed

Each of the six reconstructed image requires 960x540x32 bits before being transferred. Furthermore, if we aim to achieve 10 fps on each camera, this will require a bandwidth of 121.5MiB/s. A Gigabit Ethernet port is capable of transmitting, theoretically, 122 MiB/s, but due to a limitation on the clock frequency on the FPGA, a maximum of 65 MiB/s can be obtained on each Ethernet port. For the previous reason, two Gigabit Ethernet ports should be used to accomplish the desired performance. In the 6.8 the transmission speed monitored during a sampling period with a mean of 9.41 fps can be appreciated.

As for the frame rate, the obtained result for each camera during the sampling is shown in Figure 6.9.

6.4.3 Perception System performance

The evaluation of the GPGPU implementation of the HOG with SVM detector is shown in the Figure 6.10. All the tests were done with the help of an NVIDIA GTX Titan GPU on a Linux Host running Ubuntu 14.04 with a Core i7 4770K@3.5GHz.

It is worth pointing out that the prototype is programmed in a parallel way. In other words, after the images of the six cameras have been acquired and transferred to the host, the object detection phase starts, but simultaneously the next batch of

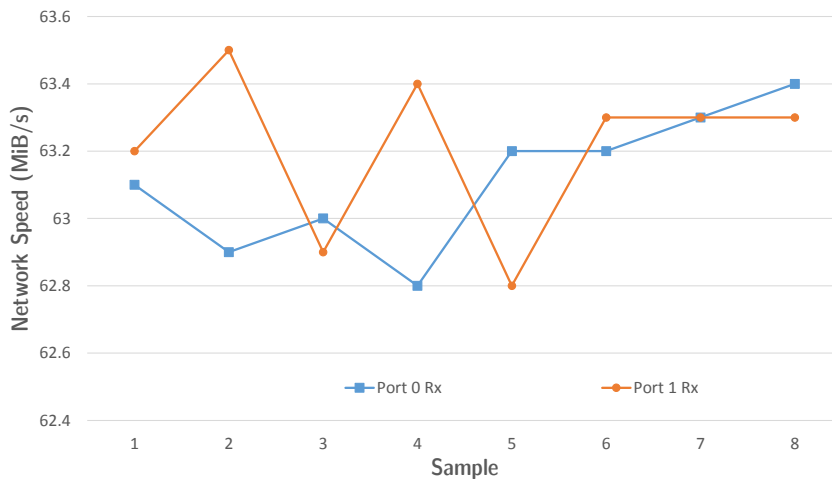


FIGURE 6.8: Network speed measured while capturing

frames is transmitted immediately. This detection phase is executed in less time than next image acquisition cycle. Thanks to this, the execution is entirely limited to the camera response time.

In Figure 6.10 can be observed that the execution time of the GPU implementation of the detector just takes 25.35 milliseconds. Put differently, the perception phase does not interfere with the image acquisition stage.

For benchmarking purposes the same detector was implemented in CPU mode, optimized using the multi threading library pthread. Unfortunately, the sequential version led to the loss of performance. The processing time required by the serial execution is 339.88 milliseconds. This would cause a serious bottle-neck in the system flow, dropping the frame rate from 9.41 to 2.94 fps. The timing of the CPU version can be appreciated in Figure 6.11.

By way of comparison, the Table 6.6 summarizes the features between our prototype and the commercially available Ladybug 5 camera.

TABLE 6.6: Comparison between HexaCam and Ladybug

Feature	Ladybug 5	HexaCam
Interface	USB 3.0	Gigabit Ethernet
Bandwidth	5 Gbps	1 Gbit
Maximum Resolution	30 MP	3 MP in current setting configurable up to 30 MP
Frame rate	10 fps	9.41 fps
Programmability	Possible	Possible
Cost	USD \$20,000~	USD \$2,000~

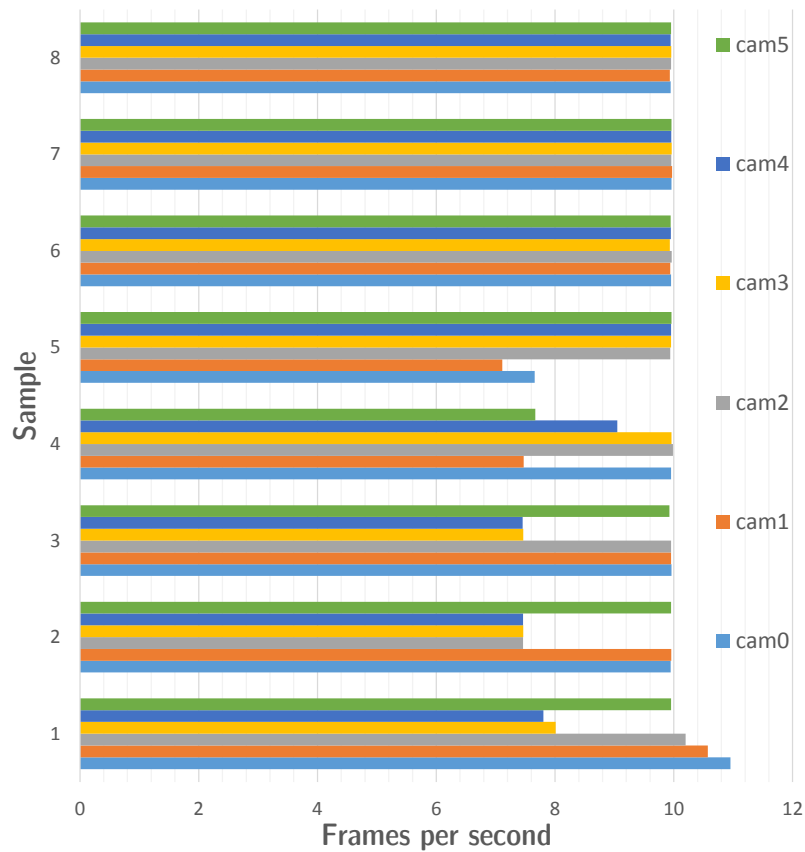


FIGURE 6.9: Framerate for each camera while capturing

6.4.4 Power consumption performance

HexaCam being an FPGA based system in the range of the 40 nm has a low power consumption requirement. Due to this and to its reduced size and weight, it can easily be used in mobile scenarios such as robotics and automotive applications [108].

6.5 System Calibration and Fusion

Having evaluated the hardware and obtained its performance, we calibrated the cameras intrinsically and extrinsically with a Velodyne VLP-16 3D LiDAR. The device is equipped with six cameras equally distributed over a circumference, as described in Section 6.3.2; the LiDAR is installed directly on top of the camera modules. Using the guidelines discussed in Section 4.8 we obtained the camera intrinsic parameters, and later the camera-LiDAR extrinsic parameters as introduced in

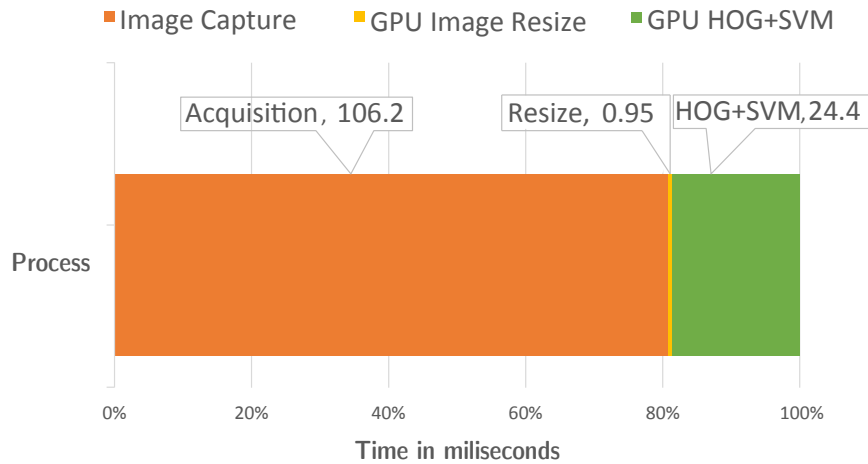


FIGURE 6.10: Processing time on GPU

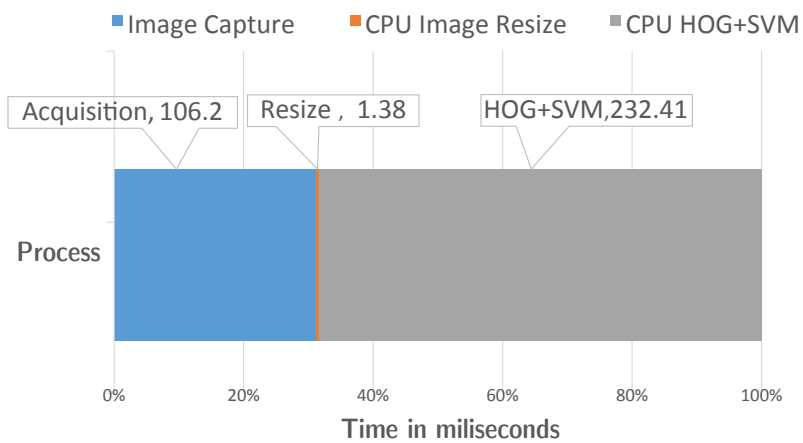


FIGURE 6.11: Processing time on CPU

Section 5.3.2. With these on hand, we proceeded to validate them qualitatively using the Image-Cloud fusion presented in Section 5.3.3. Qualitative results of this calibration and fusion are presented in Figure 6.12.

In a similar manner to the HexaCam module, we performed camera intrinsic calibration and camera-LiDAR using the LadyBug camera and a Velodyne HDL64 3D LiDAR, and fused the image from the five cameras and the point cloud from the LiDAR sensor. Qualitative results of this calibration and fusion are presented in Figure 6.13.

Both sensors provided excellent projection and backprojection results, helping to further validate the calibration methods presented in this work.

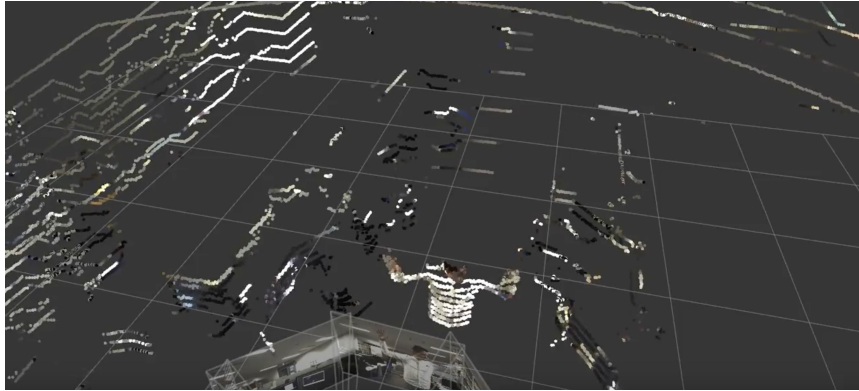


FIGURE 6.12: Qualitative results of the Image-cloud on the HexaCam device and a Velodyne VLP16.

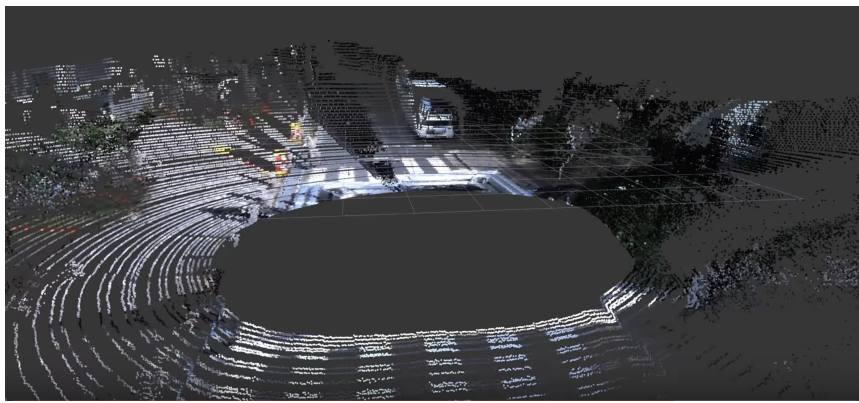


FIGURE 6.13: Qualitative results of the Image-cloud on the Ladybug camera and a Velodyne HDL64.

6.6 Conclusion

In this chapter we presented a cyber-physical system prototype to obtain the perception information contained on 360 degree images captured by a custom built FPGA board.

We found that with a tenth of the cost of the commercially available devices, we were able to provide a remarkable performance. Our prototype is able to capture images from six camera sensors and extract the perception information at a rate of 9.41 frames per second with the help of an NVIDIA GTX Titan GPU.

Additionally, we also tested and validated the methods presented in Chapters 4 and 5 using both the HexaCam presented in this chapter, and the commercially available Ladybug5 camera, which we used as a comparison point with the HexaCam device. Both sensors provided good fusion results as shown in Section 6.5. Moreover, we confirmed that our calibration methods helped us to reduce the time required to

calibrate these sensors feature large array of cameras.

For future improvements, as for the hardware, we plan to update the camera sensors configuration and increase the transmission rate of the Ethernet port, so it fully uses the available bandwidth. As for the software, we will update the firmware to transfer the RAW image instead of the full color RGB reconstruction, in this way the bandwidth will be reduced. As direct consequence, only one Ethernet port will be required, allowing the device to be used in more common hardware configurations. Moreover, one possible approach to improve the data copy between the Hexacam and the host, is to use a direct copy between the network controller and the GPU [116–118]. Finally, since the GPU implementation of HOG is quick enough, we plan to change the object detector algorithm for the well-known Deformable Part Model, using as well a GPGPU implementation [119, 120].

Chapter 7

Real World Data Circulation in Multi-Sensor Systems

7.1 Introduction

Industries worldwide are continuously looking for growth, development, and expansion of business activities. Japan is a global leader in developing and manufacturing high-quality products and services. For this reason, Japanese companies should consider the situation and events happening overseas. Additionally, companies should stay continuously open to feedback to improve and create new products and services required and desired by customers. To achieve this, companies need to create loop feedback between them and the customer base.

Real-World Data Circulation (RWDC) encompasses continuous improvement. It does it while integrating scientific fields such as Information Science, Medicine, and Economics in the loop between industry and customer, to promote the growth of companies while simultaneously developing convenient and enjoyable products for end-users that encourage social value generation. Social value quantifies the relative importance that people place on the changes they experience in their lives. In summary, RWDC aims to develop successful applications in commercial and non-profit fields, backed by data analysis and a spirit of continuous improvement. In order to achieve this, it encloses three specific phases: 1) The data acquisition step; 2) Data analysis; and 3) System implementation phase.

Autonomous Driving and Self-Navigation Robots promise the reduction of accidents, facilitate the transportation of the elderly, the automation of goods distribution, and other applications which currently suffer, or might shortly, from a shortage of personnel to satisfy transportation needs. Self-driving technologies also promise to improve the quality of life of its users and the people in charge of providing the service. For the above reason, we can consider this technology brings social value, convenience, and joy to all the persons involved, from its development and production to its end-users.

The work presented in this dissertation introduces an An End-to-End Multi-Sensor Fusion System for Autonomous Driving Applications, which as presented in Chapter 3 is part of the sensing module which enable other tasks such as perception, localization, planning and control. These modules are a niche market, targeted explicitly to the automotive, robotics, and "Mobility as a Service (MaaS)" sectors to mention some. However, sensing is essential to implement self-driving technologies, which, when implemented, implicitly improve the quality of life of users and thus bring social value.

In the following subsections, we will discuss the work presented in Chapters 4 to 6 in terms of the RWDC. All these works are tightly related to each other. Camera intrinsic calibration is essential to enable the camera-LiDAR extrinsic calibration. Consequently, accurate LiDAR-LiDAR calibration is critical to allow a multi-sensor fusion system. Finally, in Section 7.4 this work introduces the personal experience of the author as the leader of a startup backed up by the University.

7.2 Automatic Single-Shot Camera Calibration

The research related to sensing focuses on the continuous improvement of perception systems for self-driving vehicles, as shown in the loop in Figure 7.1. In this cycle, the safe deployment of sensing systems provides wellness and improved quality of life to the users. In a study conducted by the Highway Loss Data Institute (HLDI), in the last 20 years, ADAS technologies installed in vehicles have been continuously enhancing user safety [121]. Figure 7.2 shows the accident reduction achieved thanks to each of the sensing technologies introduced in Chapter 3. This figure shows how

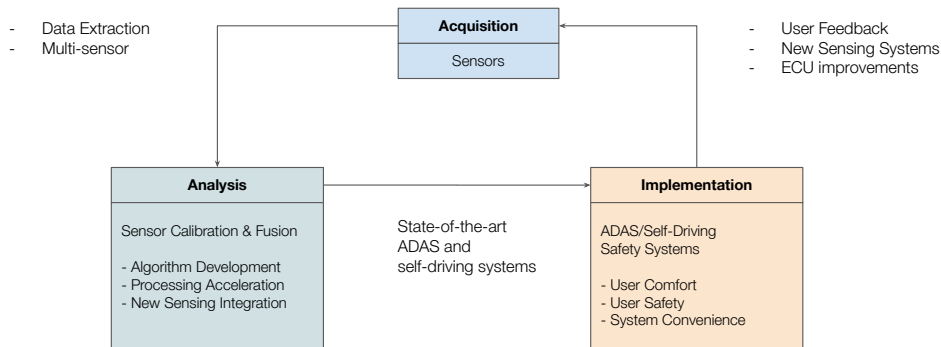


FIGURE 7.1: Continuous improvement RWDC loop applied to sensing systems.

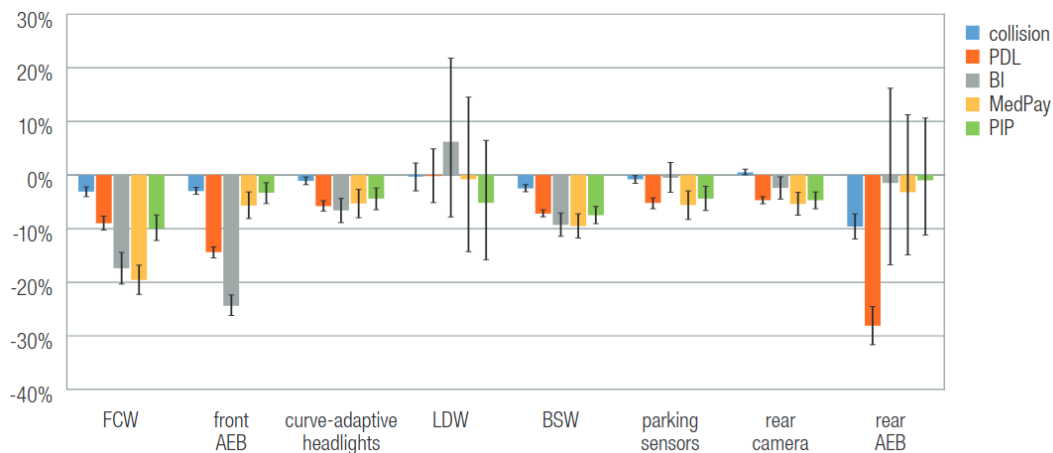


FIGURE 7.2: Percent change in claim frequency associated with collision avoidance technologies by coverage type since 2009. Please refer to Table 1 and section 3.2 for an explanation of each of the technologies.

sensing technologies are constantly improving and bringing social value in the shape of safety, reducing accidents, and, more importantly, protecting the users' lives.

There are two critical points to be considered when developing sensing systems; the first is the system's usability, and the second is the reliability. First, we developed a completely automatic system that can obtain the camera intrinsic parameters using a single shot. A fully automatic system helps accelerate development and deployment, removing the need for specialized personnel. To address the second point, we generated thousands of multiple checkerboard poses and evaluated them to obtain position and rotation intervals that maximize the probability of estimating accurate camera intrinsic parameters. These results gave us enough information

to generate checkerboard pose guidelines. Using these guidelines, we developed sets of multiple checkerboard poses and evaluated them using synthetic data obtained from a simulator and three different cameras in the real world, as explained in Sections 4.6.5 and 4.7.

Each of the above developments can be associated with Figure 7.1 and summarized in Table 7.1. **Acquisition.** Data Acquisition might take multiple forms in the sense of the Data Circulation. For instance, camera sensors provide raw data, millions of bits forming an image, these images in our case contain checkerboards used for camera calibration. However, these images by themselves do not expose immediate useful information. For this reason, we performed an **analysis**, on which we detect checkerboards, as explained in Section 4.3 backed by previous research [31–33]. Once understanding the methods used for calibration we proceeded to simulate and score checkerboard positionings to find the optimal poses as described in Section 4.4. With these results, we generated guidelines to position checkerboards to obtain accurate calibration parameters. Finally, we **implemented** a fully automated calibration system as illustrated in Section 4.8, and also released an image overlay tool to help the checkerboard positioning on the real-world as shown in Figure 4.7. In order to close the loop, we additionally pointed some of the improvements that our method in Section 4.8. Case in point, extend our work for other non-perspective cameras such as fish-eye lenses.

TABLE 7.1: Single-shot calibration related to the Data Circulation

Circulation Phase	Method	Data
Acquisition	Single Sensor	Images
Analysis	Simulation + Statistical Analysis	Optimized Score
Implementation	Positioning helper + Single-Shot calibration toolbox	Calibration Parameters

7.3 Multi-Sensor Fusion Toolbox for Autonomous Driving

As we introduced in Section 7.2 accurate sensing systems are required to ensure user safety and enable the prompt triggering of ADAS systems and self-driving applications. An autonomous vehicle requires several sensors to understand its surroundings and act accordingly in different scenarios [84]. Images from camera devices, range data from LiDARs, speed information from radars, and other sensor data are fused to achieve single-digit centimeter-level accuracy of the object detection. Thanks to the fast development of autonomous driving technologies, the cost of these sensors is rapidly reducing. The simultaneous integration of data from multiple sensors is known as fusion, and it is used to overcome weaknesses in each sensor. Multiple LiDARs and cameras allow the system to benefit from redundant, complementary, and timely information. Moreover, multiple sensors distributed around the vehicle improve the field of view, hence the safety of the system and its users. Nevertheless, increasing the number of sensors requires better synchronization, fast and reliable fusion techniques, and optimized processing methods due to the increased bandwidth.

The multi-sensor fusion toolbox we presented in Chapter 5 is integrated in the autonomous driving framework known as Autoware [103], and it is currently being used all over the world. Thank to this, our sensing framework brings impact and accelerate the development of systems that consequently bring social value by improving the safety of users. For example, the Autoware platform could be adapted in multiple vehicles to transport individuals who cannot drive by themselves or do not have someone to assist them and are located in the countryside, where other transportation services are unavailable.

Additionally, Autoware, being an open-source project, is an evolving software with constantly changing requirements, thanks to opinions from users riding self-driving vehicles and feedback from other developers. The development of Autoware is constant and continuously improving, integrating new algorithms and, more importantly, bringing value to the riders' life.

In a similar manner to Section 7.2, we can identify the data circulation phases

as: **Acquisition.** Data obtain from sensors, in this case not only cameras which generate images as explained in Section 3.5, and LiDARs, which generate point cloud as explained in Section 3.4 and Section 3.4.1. **Analysis.** When working with images and point clouds, in order to obtain their relationship, we pointed that an additional step for its relation is required (Section 5.3). Once established the relationship, the analysis consisted of the direct projection of the 3D points to the image and the optimization of the projection as explained in Section 5.3.3. **Implementation.** It implied the integration of multi-LiDAR calibration using the features obtained in the analysis and a score reduction. We implemented this in the form of an online optimization toolbox and a semi-supervised calibration method inside the Autoware [103] framework.

The data circulation phases related to our work in Multi-Sensor Fusion Toolbox for Autonomous Driving is summarized in Table 7.2.

TABLE 7.2: Multi-Sensor Fusion Toolbox related to the Data Circulation

Circulation Phase	Method	Data
Acquisition	Multiple Sensors	Images + Point Cloud
Analysis	Feature Extraction+ Statistical Analysis	Optimized Score
Implementation	Semi-supervised calibration toolbox	Calibration Parameters

In summary, an accurately calibrated multi-sensor system, being an integral part of an ADAS system (Section 3.2), helps to build technologies that save lives in short-term while being used in current generation level 2 systems enabling the prompt and accurate triggering of collision warning (Section 3.2.1) and collision intervention systems (Section 3.2.2). Additionally, automated calibration systems, such as the one presented in this dissertation, helps develop and deploy next-generation self-driving systems at a large scale in the long term, providing the sensing required for different dynamic driving tasks such as parking, lane changing, planning, among others (Section 3.1.2).

7.4 Leading a Start-Up Company

As a member of the Real-World Data Circulation Leaders program, the author of this work had the chance to start a company, name it Perception Engine, and perform the role of chief representative while being a foreign national. Perception Engine, a startup focused on sensing and perception systems, has given the author the chance to practice the RWDC with the products and services it currently offers.

Perception Engine has been a bridge between the research results presented in Chapters 4 to 6 and the industry. Case in point, as explained in Section 7.1, accurate calibration parameters are of the utmost importance to ADAS and self-driving systems. For this reason, calibration systems that accelerate the calibration process are essential to expedite deployment and production. Figure 7.3 shows some of the real-world systems in the results of calibration have been successfully applied. Figure 7.3a illustrates the Olympic Vehicle Platform debuted on the Tokyo Olympics, which used the calibration method presented in Chapter 5, a system composed only by multiple LiDAR sensors.

Vehicle To Everything (V2X) systems are a new generation of information and communication technologies that connect vehicles to everything (vehicle-to-vehicle, vehicle-to-infrastructure, vehicle-to-network, vehicle-to-device, vehicle-to-grid). V2x systems create a more comfortable and safer transportation environment and have much significance for improving traffic efficiency and reducing pollution and accident rates. Figure 7.3b shows a system requiring multiple sensor calibration in a V2X application. The autonomous vehicle shown below and the sensing system installed in the utility pole seen on the top right required the calibration of multiple LiDARs and cameras.

Similarly, the vehicle shown in Figure 7.3c required not only multiple LiDAR calibrations but also multiple camera intrinsic and extrinsic calibration. Finally, the experimental vehicle developed in Nagoya University vehicle illustrated in Figure 7.3d is equipped with state-of-the-art LiDARs, radars, and camera sensors. This vehicle requires constant calibration and maintenance since its constantly capturing data all around Japan, and due to mechanical vibration, sensors are slowly changing their physical position, modifying the calibration parameters.



(A) Olympic Vehicle Platform Calibration.



(B) Vehicle To Everything Calibration



(C) Robot Taxi.



(D) Nagoya University's Experimental Vehicle

FIGURE 7.3: Example Vehicles using real-world multiple sensor calibration technologies developed by Perception Engine.

7.5 Summary

This chapter discussed each of the works in this dissertation to the RWDC. The development of accurate and sensing and automatic camera and multi-sensor calibration systems aimed to improve vehicle users' safety and constant enhancement in the current generation of ADAS systems and the future development of fully self-driving systems.

Finally, the work executed in Perception Engine helped to continuously close the data circulation loop as presented in Figure 7.1. Numerous projects required constant changes and adaptations for continuous implementation cycles, bringing customer feedback. Each of these presented new challenges, demanding additional data acquisition, processing, and implementation; Leading to continuous improvement in terms of the whole system while simultaneously bringing important information for further enhancements.

Chapter 8

Conclusions

This thesis has proposed an end-to-end multi-sensor fusion system for autonomous driving applications, a first-of-its-kind method to generate clear guidelines for single shot camera intrinsic calibration using multiple checkerboards, suitable for use in 3D applications. Additionally, we also introduced an open-source multi-sensor fusion toolbox for autonomous vehicles. It is composed of a LiDAR-to-LiDAR extrinsic calibration algorithm, a Camera-LiDAR extrinsic calibration method, Multi-LiDAR fusion, Camera-LiDAR fusion, and a ground classification method. Our LiDAR-to-LiDAR calibration algorithm successfully adapts a state-of-the-art matching algorithm. Through several experiments, we showed that provides calibration to the centimeter-level accuracy, without requiring a special setup.

The overall results show that with the help of camera simulations we accelerated the camera modeling process, its evaluation, and ultimately the creation of guidelines to obtain accurate intrinsic parameters. We can also infer that even if the simulations create ideal image conditions, i.e., images without chromatic aberration, vignetting, and so on, we can still transfer the lessons learned to the real world. It would have been challenging and costly to replicate the simulated experiments in the real world since they require specialized equipment to position and rotate the checkerboards. Moreover, to obtain the ground truth corner coordinates, a team of labelers would be necessary to identify each corner at the pixel level, extending the time required to complete this work.

We introduced an easy-to-use Camera-to-LiDAR extrinsic calibration method. It applies the latest developments on camera pose estimation in computer vision. Instead of requiring the preparation of an specific setup, or the printing of predefined

markers, a UI is provided, so the user feeds the corresponding points between the camera and LiDAR space to our algorithm. Our method achieved error as low as one centimeter, comparable to other state-of-the-art developments in the field. It is important to note that our method will fail to obtain an accurate calibration if the intrinsic parameters are incorrect. The application of our method becomes challenging when using low resolution LiDARs in combination with telephoto lenses. Users can have a hard time to find shared features between images and point clouds. In these cases, targets might be inserted manually. We found that cones, or highly reflective materials are easy to identify between images and point clouds.

The final part of our framework is an accurate geometric and rule-based ground classifier. Thanks to the nature of the algorithm, and the low number of configurable parameters, it can be used with virtually any kind of LiDAR sensor, including data composed by several LiDARs. Moreover, our method performed on average as fast as 55 ms for the high-resolution Velodyne HDL-64. It averaged 20 and 11 ms on the HDL-32 and VLP-16 sensors respectively, achieving real-time performance on the three tested sensors.

Regarding future extension of our single-shot camera intrinsic calibration module, we still need to explore cameras other than perspective ones. For instance, integrate the fish-eye lens model by Scaramuzza [122] which would help to extend the applications of our framework, and ensure a one-shot accurate calibration for this type of ultra wide angle lenses.

Additionally, we also aim to convert our semi-supervised camera-LiDAR to follow a fully unsupervised approach. This would allow the acceleration and deployment of this module that could be built into a calibration garage that would allow to perform full automatic calibration of the cameras and LiDARs installed in the vehicle.

Finally, we believe the theory and algorithms developed in this work can be further extended to integrate other sensors such as radars, stereo cameras and ultrasonic sensors. Integrating all these in our framework, would extend and facilitate the calibration and processing of data for different ODDs and ADAS tasks in automotive and robotics applications. This would additionally ease the sensor selection to fit production costs, while also ensuring accuracy on the sensor calibration.

Bibliography

- [1] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. 2nd ed. New York, NY, USA: Cambridge University Press, 2003. ISBN: 0521540518.
- [2] Radu Bogdan Rusu and Steve Cousins. “3D is here: Point Cloud Library (PCL)”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China: IEEE, 2011.
- [3] Weimin Wang, Ken Sakurada, and Nobuo Kawaguchi. “Reflectance Intensity Assisted Automatic and Accurate Extrinsic Calibration of 3D LiDAR and Panoramic Camera Using a Printed Chessboard”. In: *Remote Sensing* 9.8 (2017). ISSN: 2072-4292. DOI: 10.3390/rs9080851. URL: <https://www.mdpi.com/2072-4292/9/8/851>.
- [4] Filippo Basso, Emanuele Menegatti, and Alberto Pretto. “Robust Intrinsic and Extrinsic Calibration of RGB-D Cameras”. In: *IEEE Transactions on Robotics* 34.5 (2018), pp. 1315–1332. DOI: 10.1109/TR0.2018.2853742.
- [5] Andreas Geiger et al. “Automatic camera and range sensor calibration using a single shot”. In: *2012 IEEE International Conference on Robotics and Automation*. 2012, pp. 3936–3943. DOI: 10.1109/ICRA.2012.6224570.
- [6] Francisco Vasconcelos, João P. Barreto, and Urbano Nunes. “A Minimal Solution for the Extrinsic Calibration of a Camera and a Laser-Rangefinder”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34.11 (2012), pp. 2097–2107. DOI: 10.1109/TPAMI.2012.18.
- [7] Lipu Zhou, Zimo Li, and Michael Kaess. “Automatic Extrinsic Calibration of a Camera and a 3D LiDAR Using Line and Plane Correspondences”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2018, pp. 5562–5569. DOI: 10.1109/IROS.2018.8593660.

-
- [8] Zoltan Pustai and Levente Hajder. "Accurate Calibration of LiDAR-Camera Systems Using Ordinary Boxes". In: *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*. 2017, pp. 394–402. DOI: 10.1109/ICCVW.2017.53.
- [9] Yole Développement. *LIDAR FOR AUTOMOTIVE AND INDUSTRIAL APPLICATIONS 2021*. 2021. URL: <https://www.yole.fr> (visited on 01/07/2022).
- [10] World Health Organization. *Global Status Report on Road Safety*. Tech. rep. Management of Noncommunicable Diseases, Disability, Violence and Injury Prevention, 2018.
- [11] Eleni Petridou and Maria Moustaki. "Human Factors in the Causation of Road Traffic Crashes". In: *European Journal of Epidemiology* 16.9 (2000), pp. 819–826.
- [12] R. O'toole. "Gridlock: Why We're Stuck in Traffic and What to Do About It". In: 2010.
- [13] S. Widodo, T. Hasegawa, and S. Tsugawa. "Vehicle fuel consumption and emission estimation in environment-adaptive driving with or without inter-vehicle communications". In: *Proceedings of the IEEE Intelligent Vehicles Symposium 2000 (Cat. No.00TH8511)*. 2000, pp. 382–386. DOI: 10.1109/IVS.2000.898373.
- [14] Bing Yu, Weigong Zhang, and Yingfeng Cai. "A Lane Departure Warning System Based on Machine Vision". In: *2008 IEEE Pacific-Asia Workshop on Computational Intelligence and Industrial Application*. Vol. 1. 2008, pp. 197–201.
- [15] *Handbook of Driver Assistance Systems*. Springer, 2016.
- [16] Nicolas Carion, Massa, and others. "End-to-End Object Detection with Transformers". In: *Computer Vision – ECCV 2020*. 2020.
- [17] Kaiming He et al. "Mask R-CNN". In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 2980–2988.
- [18] Joseph Redmon and Ali Farhadi. "YOLO9000: Better, Faster, Stronger". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.

- [19] Markus Maurer et al. *Autonomous driving: technical, legal and social aspects*. Springer Nature, 2016.
- [20] Chuck Thorpe et al. "Vision and navigation for the Carnegie-Mellon Navlab". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 10.3 (1988), pp. 362–373.
- [21] Takeo Kanade, Chuck Thorpe, and William (Red) L. Whittaker. "Autonomous Land Vehicle Project at CMU". In: *Proceedings of ACM 14th Annual Conference on Computer Science (CSC '86)*. 1986, pp. 71–80.
- [22] Martin Buehler, Karl Iagnemma, and Sanjiv Singh. "The 2005 DARPA grand challenge: the great robot race". In: 36 (2007).
- [23] Michael Montemerlo, Becker, et al. "Junior: The Stanford entry in the Urban Challenge". In: *Journal of Field Robotics* 25.9 (2008), pp. 569–597.
- [24] Martin Buehler, Karl Iagnemma, and Sanjiv Singh. *The DARPA urban challenge: autonomous vehicles in city traffic*. Vol. 56. springer, 2009.
- [25] Waymo. *Waymo Story*. 2021. URL: <https://waymo.com/company/> (visited on 12/07/2021).
- [26] Waymo. *Introducing the 5th-generation Waymo Driver*. 2020. URL: <https://blog.waymo.com/2020/03/introducing-5th-generation-waymo-driver.html> (visited on 12/07/2021).
- [27] Morgan Quigley et al. "ROS: an Open-Source Robot Operating System". In: *ICRA workshop on open source software*. Vol. 3. 3.2. Kobe, Japan. 2009, p. 5.
- [28] Shinpei Kato et al. "Autoware on Board: Enabling Autonomous Vehicles with Embedded Systems". In: *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*. 2018, pp. 287–296.
- [29] The Autoware Foundation. *Autoware Story*. 2019. URL: <https://www.autoware.org/visionandmission> (visited on 12/07/2021).
- [30] Greeley Daily Tribune. *Phantom Auto Parade Greeley Streets Saturday*. 1935. URL: <https://www.newspapers.com/newspage/24976380/> (visited on 12/07/2021).

-
- [31] Z. Zhang. "A flexible new technique for camera calibration". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.11 (2000), pp. 1330–1334. DOI: 10.1109/34.888718.
- [32] J. Kannala and S.S. Brandt. "A generic camera model and calibration method for conventional, wide-angle, and fish-eye lenses". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28.8 (2006), pp. 1335–1340. DOI: 10.1109/TPAMI.2006.153.
- [33] J. Heikkila and O. Silven. "A four-step camera calibration procedure with implicit image correction". In: *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 1997, pp. 1106–1112. DOI: 10.1109/CVPR.1997.609468.
- [34] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).
- [35] *MATLAB version (R2021a)*. The Mathworks, Inc. Natick, Massachusetts, 2021.
- [36] Oleksandr Bogdan et al. "DeepCalib: a deep learning approach for automatic intrinsic calibration of wide field-of-view cameras". In: *Proceedings of the 15th ACM SIGGRAPH European Conference on Visual Media Production*. 2018.
- [37] Yannick Hold-Geoffroy et al. "A Perceptual Measure for Deep Single Image Camera Calibration". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, pp. 2354–2363. DOI: 10.1109/CVPR.2018.00250.
- [38] Joris Domhof, Julian F.P. Kooij, and Darius M. Gavrilă. "An Extrinsic Calibration Tool for Radar, Camera and Lidar". In: *2019 International Conference on Robotics and Automation (ICRA)*. 2019, pp. 8107–8113. DOI: 10.1109/ICRA.2019.8794186.
- [39] Martin Simony et al. "Complex-YOLO: An Euler-Region-Proposal for Real-time 3D Object Detection on Point Clouds". In: *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*. 2018.
- [40] Jason Ku et al. "Joint 3D Proposal Generation and Object Detection from View Aggregation". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2018, pp. 1–8. DOI: 10.1109/IROS.2018.8594049.

- [41] Charles R. Qi et al. "Frustum PointNets for 3D Object Detection from RGB-D Data". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, pp. 918–927. DOI: 10.1109/CVPR.2018.00102.
- [42] Xiaozhi Chen et al. "Multi-view 3D Object Detection Network for Autonomous Driving". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 6526–6534. DOI: 10.1109/CVPR.2017.691.
- [43] A Alempijevic et al. "Mutual Information Based Sensor Registration and Calibration". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2006.
- [44] Marcelo Pereira et al. "Self Calibration of Multiple LiDARs and Cameras on Autonomous Vehicles". In: *Robotics and Autonomous Systems* 83 (2016), pp. 326–337.
- [45] P. J. Besl and N. D. McKay. "A method for registration of 3-D shapes". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14.2 (1992), pp. 239–256. ISSN: 0162-8828. DOI: 10.1109/34.121791.
- [46] Oleg Naroditsky, Alexander Patterson, and Kostas Daniilidis. "Automatic Alignment of a Camera with a Line Scan LiDAR System". In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE. 2011, pp. 3429–3434.
- [47] Martin Vel'as et al. "Calibration of RGB Camera with Velodyne LiDAR". In: (2014).
- [48] Kiho Kwak et al. "Extrinsic Calibration of a Single Line Scanning LiDAR and a Camera". In: *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE. 2011, pp. 3283–3289.
- [49] Gaurav Pandey et al. "Automatic Targetless Extrinsic Calibration of a 3D LiDAR and Camera by Maximizing Mutual Information." In: *AAAI*. 2012.
- [50] Jesse Levinson and Sebastian Thrun. "Unsupervised Calibration for Multi-Beam Lasers". In: *Experimental Robotics*. Springer. 2014, pp. 179–193.

- [51] Andreas Geiger, Philip Lenz, and Raquel Urtasun. "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012.
- [52] Sagar Behere and Martin Torngren. "A functional architecture for autonomous driving". In: *2015 First International Workshop on Automotive Software Architecture (WASA)*. 2015, pp. 3–10. DOI: 10.1145/2752489.2752491.
- [53] Society of Automotive Engineers. "Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles". In: (2021).
- [54] Gröchenig S Rehl K. "Evaluating Localization Accuracy of Automated Driving Systems". In: *Sensors (Basel)*. MDPI. 2021.
- [55] Japanese Automotive Standards Organization. "Taxonomy and definitions for terms related to driving automation systems for On-Road Motor Vehicles". In: (2018).
- [56] Society of Automotive Engineers. "Taxonomy & Definitions for Operational Design Domain". In: (2021).
- [57] National Highway Traffic Safety Administration. "A framework for automated driving system testable cases and scenarios". In: (2018).
- [58] Marco Galvani. "History and future of driver assistance". In: *IEEE Instrumentation Measurement Magazine* 22.1 (2019), pp. 11–16. DOI: 10.1109/MIM.2019.8633345.
- [59] Barry L Stann et al. "MEMS-scanned ladar sensor for small ground robots". In: *Laser Radar Technology and Applications XV*. Vol. 7684. International Society for Optics and Photonics. 2010, 76841E.
- [60] Jingyun Liu et al. "TOF Lidar Development in Autonomous Vehicle". In: *2018 IEEE 3rd Optoelectronics Global Conference (OGC)*. 2018, pp. 185–190. DOI: 10.1109/OGC.2018.8529992.
- [61] Asher Gelbart et al. "Flash lidar based on multiple-slit streak tube imaging lidar". In: *Laser Radar Technology and Applications VII*. Vol. 4723. International Society for Optics and Photonics. 2002, pp. 9–18.

- [62] Paul F McManamon et al. "Optical phased array technology". In: *Proceedings of the IEEE* 84.2 (1996), pp. 268–298.
- [63] Youmin Wang and Ming C. Wu. "Micromirror based optical phased array for wide-angle beamsteering". In: *2017 IEEE 30th International Conference on Micro Electro Mechanical Systems (MEMS)*. 2017, pp. 897–900. DOI: 10.1109/MEMSYS.2017.7863553.
- [64] Point Cloud Library. *The Point Cloud Data file format*. 2021. URL: https://pointclouds.org/documentation/tutorials/pcd_file_format.html (visited on 12/20/2021).
- [65] Alexander Lehmann. *LibLZF*. 2008. URL: <http://oldhome.schmorp.de/marc/liblzf.html> (visited on 12/20/2021).
- [66] C Brown Duane. "Close-range camera calibration". In: *Photogramm. Eng* 37.8 (1971), pp. 855–866.
- [67] Fanlu Wu, Hong Wei, and Xiangjun Wang. "Correction of image radial distortion based on division model". In: *Optical Engineering* 56.1 (2017), pp. 1 – 10.
- [68] "Mobile Robot Localization and Mapping". In: (2014). Ed. by Spyros G. Tzafestas, pp. 479–531.
- [69] Tim Caselitz et al. "Monocular camera localization in 3D LiDAR maps". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2016, pp. 1926–1931. DOI: 10.1109/IROS.2016.7759304.
- [70] Ryan W. Wolcott and Ryan M. Eustice. "Visual localization within LIDAR maps for automated urban driving". In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2014, pp. 176–183. DOI: 10.1109/IROS.2014.6942558.
- [71] Jesus Muñoz-Bulnes et al. "Deep fully convolutional networks with random data augmentation for enhanced generalization in road detection". In: *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*. 2017, pp. 366–371. DOI: 10.1109/ITSC.2017.8317901.

- [72] Agnieszka Mikołajczyk and Michał Grochowski. "Data augmentation for improving deep learning in image classification problem". In: *2018 International Interdisciplinary PhD Workshop (IIPHDW)*. 2018, pp. 117–122. DOI: 10.1109/IIPHDW.2018.8388338.
- [73] Zhengyou Zhang. "Camera calibration with one-dimensional objects". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 26.7 (2004), pp. 892–899. DOI: 10.1109/TPAMI.2004.21.
- [74] Fabio Remondino and Clive Fraser. "Digital camera calibration methods: Considerations and comparisons". In: *Ine. Arch. Photogramm. Remote Sens. Spat. Inf. Sci.* 36 (Nov. 2005).
- [75] Jiunn-Kai Huang and Jessy W. Grizzle. "Improvements to Target-Based 3D LiDAR to Camera Calibration". In: *IEEE Access* 8 (2020), pp. 134101–134110. DOI: 10.1109/ACCESS.2020.3010734.
- [76] Shi Guiming and Suo Jidong. "Multi-Scale Harris Corner Detection Algorithm Based on Canny Edge-Detection". In: *2018 IEEE International Conference on Computer and Communication Engineering Technology (CCET)*. 2018, pp. 305–309. DOI: 10.1109/CCET.2018.8542206.
- [77] Jong-Eun Ha. "Automatic detection of chessboard and its applications". In: *Optical Engineering* 48.6 (2009), pp. 1–8. DOI: 10.1117/1.3156053. URL: <https://doi.org/10.1117/1.3156053>.
- [78] Chris Harris and Mike Stephens. "A combined corner and edge detector". In: *In Proc. of Fourth Alvey Vision Conference*. 1988, pp. 147–151.
- [79] Jianbo Shi and Tomasi. "Good features to track". In: *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*. 1994, pp. 593–600. DOI: 10.1109/CVPR.1994.323794.
- [80] J. Y. Bouguet. *A Release of a Camera Calibration Toolbox for Matlab*, 2008. URL: http://www.vision.caltech.edu/bouguetj/calib_doc/.
- [81] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation. Stichting Blender Foundation, Amsterdam, 2018. URL: <http://www.blender.org>.

-
- [82] Guodong Rong et al. "LGSVL Simulator: A High Fidelity Simulator for Autonomous Driving". In: *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. 2020, pp. 1–6. DOI: 10.1109/ITSC45102.2020.9294422.
- [83] Daniel J Fagnant and Kara Kockelman. "Preparing a Nation for Autonomous Vehicles: Opportunities, Barriers and Policy Recommendations". In: *Transportation Research Part A: Policy and Practice* 77 (2015), pp. 167–181.
- [84] Umit Ozguner, Christoph Stiller, and Keith Redmill. "Systems for Safety and Autonomous Behavior in Cars: The DARPA Grand Challenge Experience". In: *Proceedings of the IEEE* 95.2 (2007), pp. 397–412.
- [85] Anna Petrovskaya and Sebastian Thrun. "Model Based Vehicle Tracking for Autonomous Driving in Urban Environments". In: *Proceedings of Robotics: Science and Systems IV, Zurich, Switzerland* 34 (2008).
- [86] Jason Ku et al. "Joint 3D Proposal Generation and Object Detection from View Aggregation". In: *arXiv preprint arXiv:1712.02294* (2017).
- [87] Martin Simon et al. "Complex-YOLO: Real-time 3D Object Detection on Point Clouds". In: *arXiv preprint arXiv:1803.06199* (2018).
- [88] Charles R Qi et al. "Frustum PointNets for 3D Object Detection from RGB-D Data". In: *arXiv preprint arXiv:1711.08488* (2017).
- [89] Radu Bogdan Rusu and Steve Cousins. "3D is Here: Point Cloud Library (PCL)". In: *Robotics and automation (ICRA), 2011 IEEE International Conference on*. IEEE. 2011, pp. 1–4.
- [90] Andreas Geiger et al. "Automatic Camera and Range Sensor Calibration Using a Single Shot". In: *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE. 2012, pp. 3936–3943.
- [91] Weimin Wang, Ken Sakurada, and Nobuo Kawaguchi. "Reflectance Intensity Assisted Automatic and Accurate Extrinsic Calibration of 3D LiDAR and Panoramic Camera Using a Printed Chessboard". In: *Remote Sensing* 9.8 (2017), p. 851.

- [92] Ahmed Abdelhafiz, Björn Riedel, and Wolfgang Niemeier. "Towards a 3D True Colored Space by the Fusion of Laser Scanner Point Cloud and Digital Photos". In: *Proceedings of the ISPRS Working Group V/4 Workshop (3D-ARCH)*. Citeseer. 2005.
- [93] Keqi Zhang et al. "A Progressive Morphological Filter for Removing Non-ground Measurements from Airborne LiDAR Data". In: *IEEE transactions on geoscience and remote sensing* 41.4 (2003), pp. 872–882.
- [94] Xuelian Meng et al. "A Multi-Directional Ground Filtering Algorithm for Airborne LiDAR". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 64.1 (2009), pp. 117–124.
- [95] Francesco Pirotti, Alberto Guarnieri, and Antonio Vettore. "Ground Filtering and Vegetation Mapping using Multi-Return Terrestrial Laser Scanning". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 76 (2013), pp. 56–63.
- [96] Daniel Maturana and Sebastian Scherer. "3D Convolutional Neural Networks for Landing Zone Detection from LiDAR". In: *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE. 2015, pp. 3471–3478.
- [97] Zhe Chen and Zijing Chen. "RBNet: A Deep Neural Network for Unified Road and Road Boundary Detection". In: *International Conference on Neural Information Processing*. Springer. 2017, pp. 677–687.
- [98] Xiaofeng Han et al. "Semisupervised and Weakly Supervised Road Detection Based on Generative Adversarial Networks". In: *IEEE Signal Processing Letters* 25.4 (2018), pp. 551–555.
- [99] Martin Magnusson, Achim Lilienthal, and Tom Duckett. "Scan Registration for Autonomous Mining Vehicles Using 3D-NDT". In: *Journal of Field Robotics* 24.10 (2007), pp. 803–827.
- [100] Eijiro Takeuchi and Takashi Tsubouchi. "A 3-D Scan Matching Using Improved 3-D Normal Distributions Transform for Mobile Robotic Mapping". In: *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*. IEEE. 2006, pp. 3068–3073.

-
- [101] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).
- [102] Vincent Lepetit, Francesc Moreno-Noguer, and P Fua. "EPnP: Efficient Perspective-n-Point Camera Pose Estimation". In: *International Journal of Computer Vision* 81.2 (2009), pp. 155–166.
- [103] Shinpei Kato et al. "An Open Approach to Autonomous Vehicles". In: *IEEE Micro* 35.6 (2015), pp. 60–68.
- [104] C. Urmson et al. "High speed navigation of unrehearsed terrain: Red Team technology for the Grand Challenge 2004". In: *TR CMU-RI-TR-04-37*. 2004, pp. 39–40.
- [105] Levinson et al. "Map-Based Precision Vehicle Localization in Urban Environments". In: *In: Proceedings of Robotics: Science and Systems, Atlanta, GA, USA*. 2007.
- [106] McNaughton et al. "Motion planning for autonomous driving with a conformal spatiotemporal lattice." In: *In: ICRA*. 2011, 4889—4895.
- [107] R. Girshick et al. "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation". In: 2014.
- [108] M. Smerdon. *High-Volume Spartan-6 FPGAs: Performance and Power Leadership by Design*. 2011. URL: http://www.xilinx.com/support/documentation/white_papers/wp396_S6_HV_Perf_Power.pdf (visited on 04/01/2015).
- [109] PointGrey. *Ladybug 5*. <http://www.ptgrey.com/support/downloads/10150>. Accessed: 2015-04-01.
- [110] OmniVision. *Omnivision 5647 Product Brief*. <http://www.ovt.com/uploads/parts/OV5647.pdf>. Accessed: 2015-04-01.
- [111] Bryce E. Bayer. "Color imaging array". Pat. 1976.
- [112] R. Kimmel. "Demosaiicing: Image reconstruction from color CCD samples." In: *IEEE Trans. on Image Processing*. Vol. 1. 1999.
- [113] Ronald W Schafer and Russel M Mersereau. "Demosaiicing: color filter array interpolation". In: *Signal Processing Magazine, IEEE* 22.1 (2005).

- [114] Zhan Yu et al. "An analysis of color demosaicing in plenoptic cameras". In: *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE. 2012, pp. 901–908.
- [115] N. Dalal and B. Triggs. "Histograms of Oriented Gradients for Human Detection". In: 2005, pp. 886–893.
- [116] Shinpei Kato, Jason Aumiller, and Scott Brandt. "Zero-copy I/O Processing for Low-latency GPU Computing". In: *Proceedings of the ACM/IEEE 4th International Conference on Cyber-Physical Systems*. ICCPS '13. Philadelphia, Pennsylvania: ACM, 2013, pp. 170–178. ISBN: 978-1-4503-1996-6. DOI: 10.1145/2502524.2502548.
- [117] Yusuke Fujii et al. "Data transfer matters for GPU computing". In: *Parallel and Distributed Systems (ICPADS), 2013 International Conference on*. IEEE. 2013, pp. 275–282.
- [118] Anh Nguyen et al. "Reducing Data Copies between GPUs and NICs". In: *Cyber-Physical Systems, Networks, and Applications (CPSNA), 2014 IEEE International Conference on*. IEEE. 2014, pp. 37–42.
- [119] Pedro Felzenszwalb, David McAllester, and Deva Ramanan. "A discriminatively trained, multiscale, deformable part model". In: *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*. IEEE. 2008, pp. 1–8.
- [120] Manato Hirabayashi et al. "GPU implementations of object detection using HOG features and deformable models". In: *Cyber-Physical Systems, Networks, and Applications (CPSNA), 2013 IEEE 1st International Conference on*. IEEE. 2013, pp. 106–111.
- [121] Highway Loss Data Institute. *Compendium of HLDI collision avoidance research*. 2020. URL: <https://www.iihs.org> (visited on 12/29/2021).
- [122] Davide Scaramuzza, Agostino Martinelli, and Roland Siegwart. "A flexible technique for accurate omnidirectional camera calibration and structure from motion". In: *Fourth IEEE International Conference on Computer Vision Systems (ICVS'06)*. IEEE. 2006, pp. 45–45.