VEHICLE PERCEPTION SYSTEM BASED ON HETEROGENEOUS NEURAL NETWORK SENSOR FUSION

VEHICLE PERCEPTION SYSTEM BASED ON HETEROGENEOUS NEURAL NETWORK SENSOR FUSION

By

Ash(Chang) Liu, B.Eng.

Supervisors: Dr. Martin von Mohrenschildt and Dr. Saeid Habibi

A Thesis Submitted to the School of Graduate Studies in Partial Fulfilment of the Requirements for the Degree Doctor of Philosophy

McMaster University

McMaster University DOCTOR OF PHILOSOPHY (2024) Hamilton, Ontario (Software Engineering)

TITLE: Vehicle Perception System Based on Heterogeneous Neural Network Sensor Fusion

AUTHOR: Ash Liu, B.Eng. (McMaster University)

SUPERVISOR: Dr. Martin von Mohrenschildt, Dr. Saeid Habibi

NUMBER OF PAGES: ix, 230

Abstract

Previous research conducted in the CMHT lab at McMaster University led to the successful development of a Light detection and ranging (LiDAR)-based vehicle perception system, notable for its highly accurate detection of highway objects and moderate classification capabilities. This thesis builds upon that foundation, enhancing the existing system by incorporating sensor fusion with Infrared (IR) cameras and standard cameras. The implementation of sensor fusion significantly augments the system's performance, enabling it to detect and classify objects effectively under adverse weather conditions and in poor lighting.

Key contributions of this research include:

- The development of a hardware-synchronized sensor system, blending LiDAR, IR cameras, and cameras.
- The creation of a comprehensive sensor calibration process and a novel multi-sensor dataset, the first of its kind to include annotations under various lighting and weather conditions.
- The development of an innovative ground segmentation technique using the Savitzky-Golay filter and peak detection, significantly improve the speed of ground point elimination.
- The introduction of a novel optimizer cm-reSVSF for complex-valued neural networks, demonstrating superior performance compared to traditional algorithms like ADAM and SGD.

Keywords:

Deep Learning, ADAS, Vehicle Perception system, CNN, CVNN, Hybrid Neural Network, SVSF, cm-reSVSF, Sensor Calibration, Dataset.

Acknowledgements

Firstly, I want to express my heartfelt thanks to my advisors, Dr. Martin von Mohrenschildt and Dr. Saeid Habibi, for their exceptional support and guidance throughout my doctoral studies. Their knowledge, patience, and commitment have been crucial in shaping both my research and academic development. Dr. Martin von Mohrenschildt has been incredibly supportive technically and always ready to address any challenges I faced, while Dr. Saeid Habibi has consistently provided pivotal guidance throughout my research project and offered encouragement and support whenever needed.

I also owe a special thanks to Dr. Douglas Down, a key member of my Ph.D. committee, whose insightful feedback and encouragement played a significant role in my successful defense. His expertise and thoughtful critiques have significantly enhanced my research.

I am deeply grateful to my parents for their constant support and love, which have been the cornerstone of my endurance through the rigors of Ph.D. studies. Their sacrifices and support have been a continuous source of motivation.

Additionally, I appreciate Cam Fischer, an industry expert and lab coordinator, for his essential assistance with laboratory equipment and logistics. His readiness to assist and meticulous attention to detail have greatly smoothed my experimental processes.

Finally, I acknowledge all the members and staff of the CMHT team. Their teamwork and support have fostered a nurturing and productive research atmosphere.

This journey would not have been possible without all of you. Thank you.

Contents

List of Figures

List of Tables

Acronyms

ADAS	Advanced driver-assistance systems
CCD	Charge-coupled device
CIFAR	Modified National Institute of Standards and Technology
CIFAR	Canadian Institute For Advanced Research
CMHT	Centre for Mechatronics and Hybrid Technologies
CMOS	Complementary metal oxide semiconductor
CNN	Convolutional neural network
CVCN	Complex-valued convolutional network
CVNN	Complex-valued neural network
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DNN	Deep Neural Network
EKF	Extended Kalman filter
FNN	Fourier neural networks
FOV	Field of view
FPA	Focal plane array
GLE	Ground Likelihood Estimation
GP-INSAC	Gaussian Process Incremental Sample Consensus

GPR	Gaussian Process Regression
GPS	Global positioning system
IC	Integrated circuit
IMU	Inertial measurement unit
IoU	Intersection-Over-Union
IR	Infrared
JDL	US Joint Directors of Laboratories
LiDAR	Light detection and ranging
LM	Levenberg–Marquardt algorithm
LWIR	Long-wave infrared
MEMs	Microelectronechanical systems
MRF	Markov Random Field
MWIR	Medium-wave infrared
NN	Neural network
NSC	National Safety Council
PCA	Principal Component Analysis
R-CNN	Region-Based Convolutional Neural network
Radar	Radio detection and ranging
RANSAC	Random Sample Consensus
ReLU	Rectified linear unit
ROI	Region of Interest
RVNN	Real-valued neural network
SAE	Society of automotive engineers

SGD Stochastic Gradient Descent

SOC System on Chip

SOM System on Module

SVM Support Vector Machines

SVSF Smooth variable structure filter

SWIR Short-wave infrared

WHO World Health Organization

Chapter 1

Introduction

The World Health Organization (WHO) reports that each year, road traffic crashes are responsible for 1.25 million deaths and 50 million injuries globally [128]. Of the many contributing factors, human error stands out as the predominant one, as recognized by the WHO and many other studies[132][156][147]. A statistical survey shows 94% of crashes involve driver error[123]. Interestingly, despite the increase in vehicles throughout period of 2012-2018, with a rise of about 12% in Canada (Figure 1.1) and 7.87% in the United States (Figure 1.2), a remarkable decline in traffic accidents was observed in both countries. Specifically, Canada experienced approximately a 15% reduction in fatalities and injuries (Figure 1.3, Figure 1.4), while the United States saw a reduction of about 19%. The National Safety Council (NSC) attributes this promising trend to changes in driver attitudes, behaviours, and the substantial progress made in vehicle safety technologies, especially the development and use of Advanced driver-assistance systems (ADAS) and its associated technologies. ADAS stands for Advanced Driver Assistance Systems. It is a technology that utilizes a variety of sensors to perceive the environment and the driver's condition. It then responds proactively to augment the driver's situational awareness and alleviate fatigue by automating specific monotonous vehicle controls. Such a system is very crucial for today's busy and complex traffic scenarios.

Numerous studies support the effectiveness of ADAS in enhancing driving safety and reducing human errors. For instance, Masello's research [112] highlights that implementing six common ADAS in the United Kingdom led to a remarkable 23.8% reduction in accidents. This decrease amounted to an annual total of 18,925 fewer accidents. Moreover, investigations conducted by Zoghi *et al.* [181] demonstrated that simulated experiments saw a potential reduction of up to 19% in traffic accidents through the use of ADAS. These findings collectively emphasize the significant contribution of ADAS in enhancing road safety and

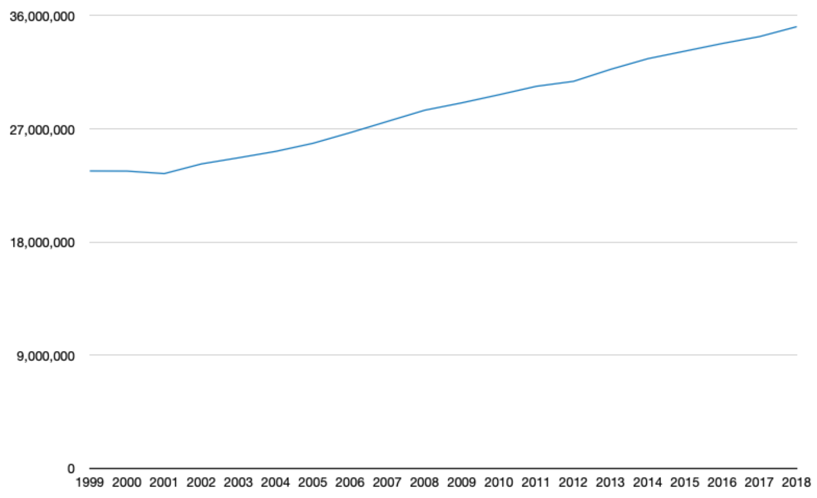


Figure 1.1: Total number of vehicle registrations in Canada[125]

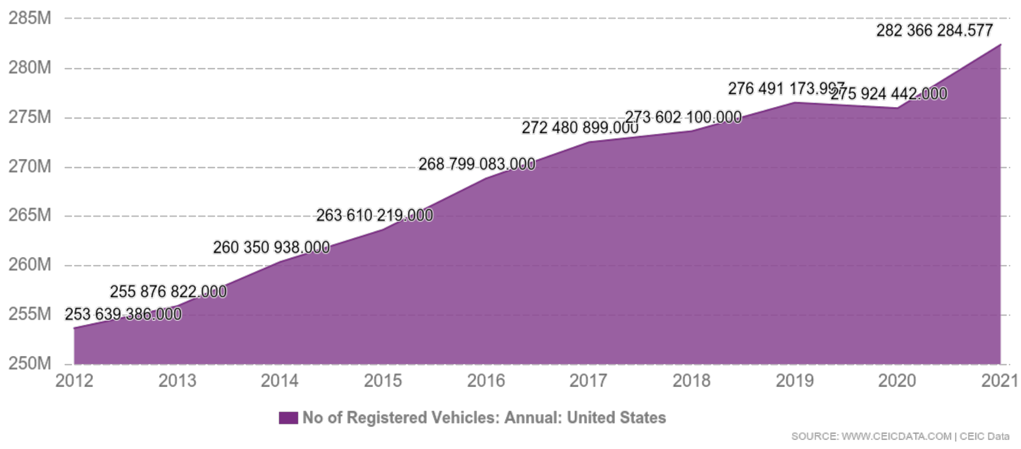


Figure 1.2: Total number of vehicle registrations in United State[31]

minimizing accidents.

1.0.1 ADAS

ADAS plays a crucial role in ensuring safe driving by providing immediate support and essential information about a vehicle's immediate surroundings. According to the National Safety Council (NSC)[114], ADAS incorporates numerous pivotal features, including Adaptive Cruise Control, Anti-Lock Braking System, Automatic Emergency Braking, Automatic Parallel Parking, Automatic Reverse Braking, Back-up Camera, Back-up Warning, Bicycle Detection, Blind Spot Warning, Brake Assist, Forward Collision Warning, Lane Departure Warning, Lane Keeping Assist, Left Turn Crash Avoidance, Obstacle Detection, Pedestrian Detection,

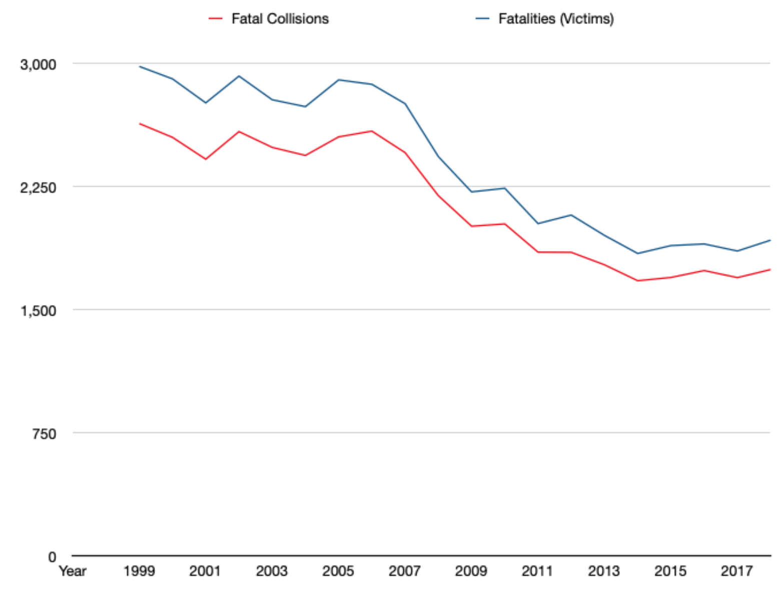


Figure 1.3: Number of fatalities from the vehicle accidents in Canada[29]

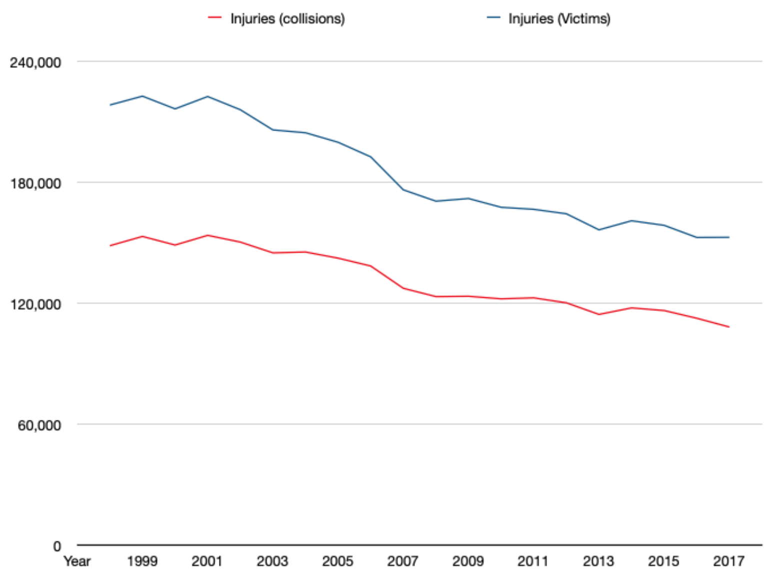


Figure 1.4: Number of injuries from the vehicle accidents in Canada[29]

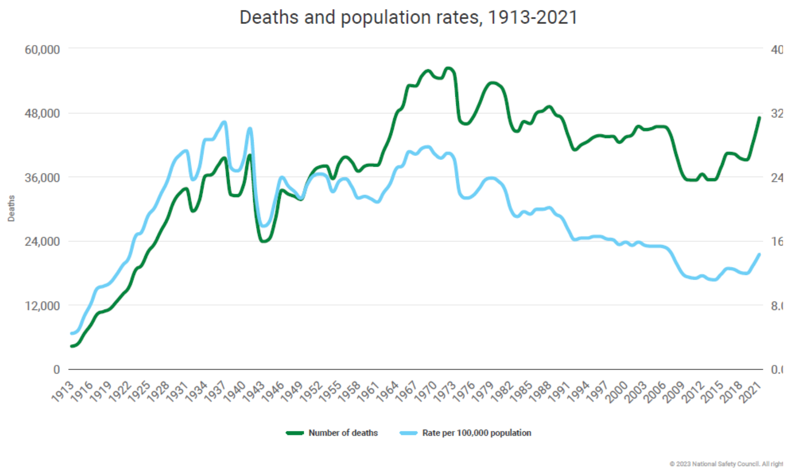
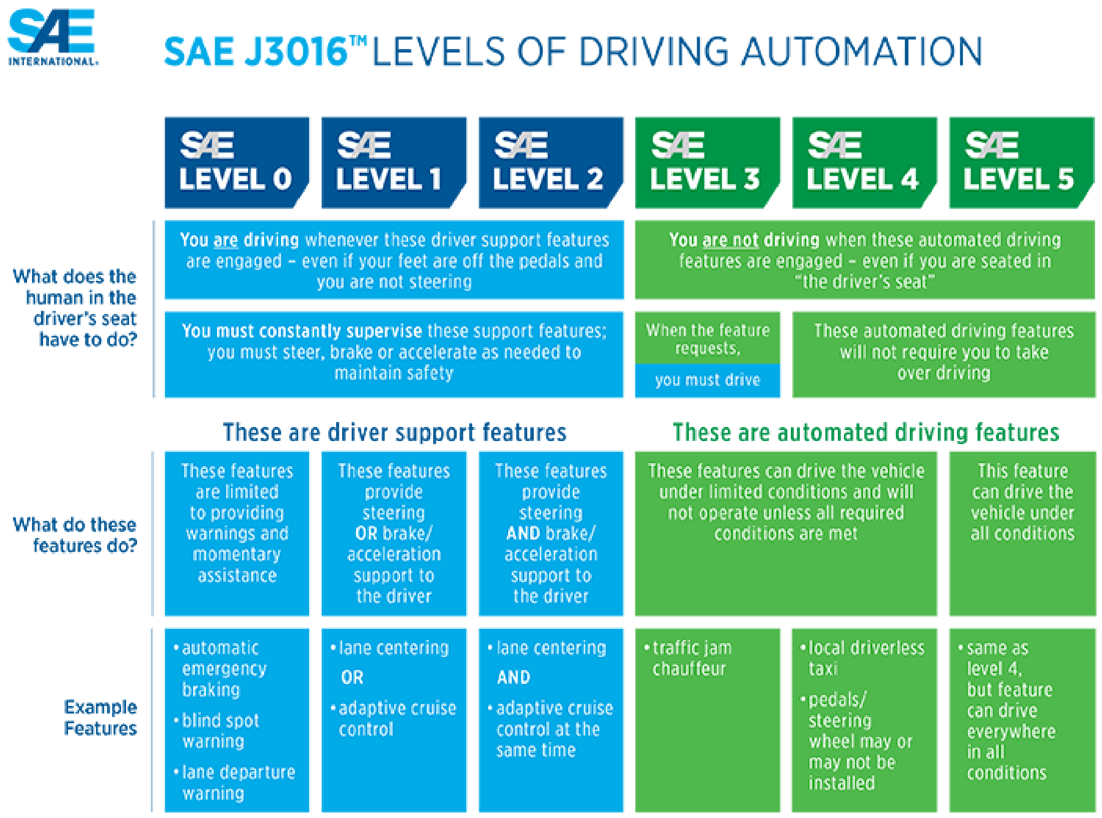


Figure 1.5: Total deaths and population rates from vehicle accidents in United State[41]

among others. These features can be generalized into two groups: vehicle perception and vehicle control, as shown in the table below.

Vehicle Perception	**Vehicle Control**
Back-up Camera	
Back-up Warning	Adaptive Cruise Control
Bicycle Detection	Anti-Lock Braking System
Blind Spot Warning	Automatic Emergency Braking
Forward Collision Warning	Parallel Parking
Lane Departure Warning	Automatic Reverse Braking
Left Turn Crash Avoidance	Brake Assist
Obstacle Detection	Lane Keeping Assist
Pedestrian Detection	

The successful commercialization of ADAS brings forth two main advantages in the path towards autonomous driving. First and foremost, it progressively instills confidence among consumers[62], fostering widespread acceptance of fully autonomous vehicles. Secondly, the functionalities of ADAS serve as fundamental building blocks for autonomous driving. As technology advances, ADAS modules can seamlessly integrate into the autonomous vehicle's ”Perception-Plan-Action” cycle. Society of automotive engineers (SAE) has released levels of driving automation first in 2016, followed by the revision in 2018(Figure 1.6).



SAE INTERNATIONAL

SAE J3016™ LEVELS OF DRIVING AUTOMATION

	SAE LEVEL 0	SAE LEVEL 1	SAE LEVEL 2	SAE LEVEL 3	SAE LEVEL 4	SAE LEVEL 5
What does the human in the driver's seat have to do?	You are driving whenever these driver support features are engaged – even if your feet are off the pedals and you are not steering			You are not driving when these automated driving features are engaged – even if you are seated in "the driver's seat"		
	You must constantly supervise these support features; you must steer, brake or accelerate as needed to maintain safety			When the feature requests, you must drive	These automated driving features will not require you to take over driving	
	These are driver support features			These are automated driving features		
What do these features do?	These features are limited to providing warnings and momentary assistance	These features provide steering OR brake/ acceleration support to the driver	These features provide steering AND brake/ acceleration support to the driver	These features can drive the vehicle under limited conditions and will not operate unless all required conditions are met		This feature can drive the vehicle under all conditions
Example Features	• automatic emergency braking • blind spot warning • lane departure warning	• lane centering OR • adaptive cruise control	• lane centering AND • adaptive cruise control at the same time	• traffic jam chauffeur	• local driverless taxi • pedals/ steering wheel may or may not be installed	• same as level 4, but feature can drive everywhere in all conditions

Figure 1.6: Level of autonomous driving[141]

As of the thesis writing, the autonomous level of most newly commercialized vehicles is primarily categorized between level 2 and level 3.

1.1 Motivation

Although vehicle perception is progressing quickly, reaching high accuracy in restricted scenarios, the pursuit of research to improve the reliability and robustness of vehicle perception systems persists. Even with many level 3 autonomous vehicles on the market that focus on camera technology and are equipped with advanced image processing, many fatalities linked to false detection[20] have exposed the concerning reality that achieving a fully reliable vehicle perception system, especially when relying solely on camera-based solutions, remains a distant goal. Consequently, the search for suitable sensors for vehicle perception applications continues.

One strong candidate in this endeavour is the LiDAR sensor. From a sensor perspective, LiDAR possesses several advantages over cameras. Notably, it is not affected by environmental light sources, and its ability to generate 3D scans makes it an excellent sensor option for vehicle perception. Historically, a major drawback of LiDAR sensors was their exorbitant cost. However, with advancements in technology, the cost of LiDAR sensors has significantly decreased, making them market-ready for commercial use.

Another sensor of choice is the IR camera. Instead of detecting visible spectrums, the IR camera only detects frequencies in the IR range. As a result, the sensor is also immune to human perceptive lighting conditions and excellent at detecting anything that generates heat because of IR radiation. In vehicle perception applications, important objects often emit heat, such as a car's engine, pedestrians, or animals. Even in complete darkness, thermal imaging is still capable of detecting these objects.

While individual sensor research for vehicle perception applications has been ongoing for some time, at the start of this research, a comprehensive methodology for fusing heterogeneous sensors in a sensor fusion system for vehicle perception needed to be improved. However, with advancements in computing hardware and the adoption of neural network approaches, it became possible to leverage different sensors and effectively fuse them to enhance vehicle perception capabilities.

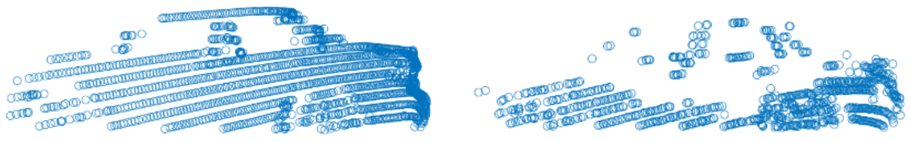
1.1.1 Past Research

McMaster University Centre for Mechatronics and Hybrid Technologies (CMHT) (Centre of Mechatronics and hybrid technology) lab has begun the vehicle perception research in the year 2012. The study was led by Dr. Luo during his time pursuing Ph.D. degree and made significant progress in the LiDAR-based vehicle perception system for highway driving[108], specifically on ground detection and object detection. This thesis is the continuation of his work with the addition of multi-sensors fusion to improving the overall system performance in different complex-driving scenarios.

1.2 Problem Description

Previous research[108] has attempted to address the vehicle perception problem using a comprehensive LiDAR-based vehicle perception system for road object detection, classification, and tracking. While the detection results have been considered good, the classification recall rate remains relatively poor.

In the previous study[108], a basic Convolutional neural network (CNN) was trained to recognize object shapes by converting LiDAR inputs from vectors to a dense range image and then performing classification based on this representation. The resolution problem gets worst when an object is further away from the sensor: The number of scans on an object rapidly decreases as it is further away from the LiDAR(Figure 1.7), causing discontinuities in the grid maps and resulting in misclassification of the object.

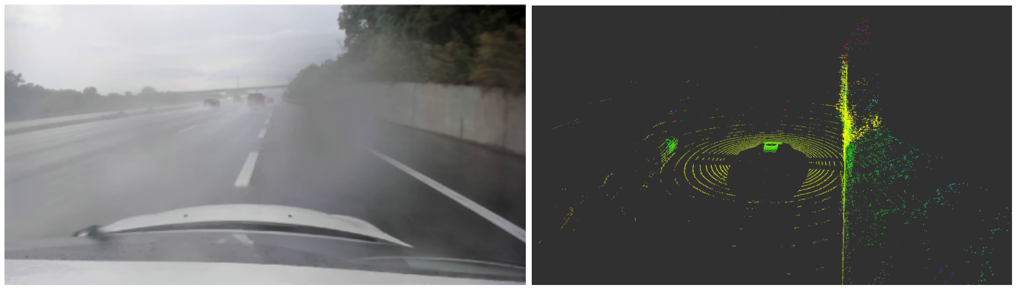


(a) Close object (b) Far object

Figure 1.7: LiDAR image appears differently from distances.

A notable concern with the preceding research is its exclusive reliance on the LiDAR sensor, a technology that is particularly susceptible to the effects of weather conditions. The effective visibility range of the LiDAR sensor appears to be compromised in instances of rainfall and its aftermath, with an observable increase in noise levels due to the divergence of the laser beam on the water accumulated on the road surface.

Given these intrinsic limitations to LiDAR-based data acquisition and its vulnerability to adverse weather, the idea that sensor fusion may offer a promising path for significant improvements in vehicular perception arises. By merging data from various sensors, it is possible to enhance the overall reliability and robustness of the perception system. This approach effectively reduces the impact of individual sensor limitations and improves the vehicle's ability to accurately perceive its surroundings across different weather conditions.



(a) Camera view (b) LiDAR view

Figure 1.8: Rain significantly diminishes the visibility of LiDAR, causing it to drop to levels as low as 5-10 meters where normally the maximum viable detection range is at 15 meters.

1.3 Novelty of the Research

In the context of this research initiative, the collaborative potential offered by LiDAR, camera, and IR camera is leveraged, seamlessly integrated within the vehicle. This effort leads to the creation of a heterogeneous sensor-fusion vehicle perception system, capable of performing real-time segmentation and object classification. The main contributions of this study can be summarized as follows:

- Implementation of a hardware-synchronized sensor system comprising LiDAR, IR camera, camera, Radio detection and ranging (Radar), and Inertial measurement unit (IMU).
- A meticulously devised comprehensive procedure for sensor calibration and spatial fusion, aimed at integrating data from LiDAR, IR camera, and camera sources.
- The creation of a novel synchronized multi-sensor labelled dataset. As of the composition of this thesis, this dataset marked a pioneering endeavor, being the first open-source repository inclusive of all these sensors, and possessing annotations under varying lighting and weather conditions.

- The development of a high-speed novel ground segmentation technique, which harnesses the Savitzky-Golay filter and peak detection mechanisms to effectively eliminate ground points.
- The introduction of an novel optimizer for complex-valued neural networks, a novel cm-reSVSF filter optimizer, exhibiting superior performance in comparison to ADAM and SGD algorithms.

1.4 Overview of the Thesis

The thesis is organized in the following manner: Chapter 1 provides an introduction. Chapter 2 introduces the sensors used and installed on the vehicles, along with an outline of the state of the art for each sensor. Chapter 3 details the sensor calibration and rectification for the sensor fusion procedure, including hand-picked algorithms for the automated spatial sensor alignment parameter estimation process, specifically for CMHT. Chapter 4 introduces the CMHT dataset, acquired by utilizing the system featured in Chapters 2 and 3. Chapter 5 and Chapter 6 describe the processing pipeline for the vehicle perception system, with ground segmentation and object detection featured in Chapter 5, and object classification using a heterogeneous neural network in Chapter 6. Chapter 7 specifically focuses on complex-valued neural network training, and presents unique findings that utilize the Smooth variable structure filter (SVSF) filter for fast training. Lastly, Chapter 8 contains the conclusion and comments on the continuation of the research.

Chapter 2

Sensors

2.1 Introduction

Sensors are vital components of autonomous vehicles and ADAS, serving as the eyes of the car and ADAS. Therefore, the appropriate selection and placement of sensor hardware on the vehicle are of paramount importance. Current technological advancements offer several options to choose from, a marked contrast to the last decade when autonomous driving was first explored. However, even today, a ”perfect” sensor does not exist, as each sensor comes with its own advantages and disadvantages. The following chapter delves into currently available sensors in detail, enumerating their pros and cons. Additionally, sensor applications are examined to understand how sensors perform their designated roles within the vehicle perception system. Finally, the CMHT vehicle sensor system setup and software implementation are introduced. At CMHT, a sensor fusion system has been developed that leverages multiple sensors to bolster the robustness of the vehicle perception system, enhancing classification performance in challenging lighting or weather conditions.

Defining a “perfect” sensor is critical to the understanding of its functionality in autonomous driving. Ideally, a “perfect” sensor has all of the following traits:

- **High resolution**: a sensor must be able to provide high fidelity data capture to detect the surrounding objects accurately. Low resolution sensors are more likely to miss capturing small objects such as fire hydrants, poles and other relatively smaller objects, and thus sensors must possess high resolution properties.

- **Low cost**: Cost is another major factor in the commercial aspect of vehicle production. Increasing the cost price to produce vehicles due to expensive sensor hardware may not make it economically feasible in the quest for the adoption of autonomous driving.
- **Passive**: a passive sensor (such as a camera, IMU, Global positioning system (GPS) and so forth) only detects and responds to the inputs from the physical environment. It does not interact with the environment when performing the measuring functions. On the other hand, active sensors like Radar, LiDAR systems often provide their own input source. Studies[87][136] have shown that multiple active sensors of the same kind can interfere with each other when working simultaneously, due to one sensor inputs coincidently feeding into another sensor, thus creating a sensor interference. For this reason, passive sensors are often preferred over active sensors.
- **High sampling rate**: The higher the frequency, the more information a sensor can acquire within a period of time. An ideal sensor sampling rate would be no less than 10Hz.
- **Weather and light condition insensitive**: Weather and daylight are natural phenomena that can often present a challenge to sensors . Rain, snow and fog will directly affect the performance of the sensors. Sensors that work in visible light frequencies are susceptible to light conditions. In some extreme cases, even shadows or light reflections can cause the sensor to fail.

The aforementioned factors play a key role in selecting the right sensors, with the task they are intended to perform taken into account. Regarding sensor selection, previous research[108] has determined that LiDAR can be very useful in detecting surroundings and accurately acquiring information in 3D space. However LiDAR lacks the ability to detect fine details and is weather-sensitive, making LiDAR unsuitable for use as the only only sensor type for any autonomous driving applications. Therefore, almost all LiDAR-based autonomous driving research vehicles are equipped with more than one kind of sensor to compensate for LiDAR drawback.

Besides LiDAR, a camera is almost a must-have sensor on autonomous driving vehicles. Cameras posses high resolution, high sampling rates, and are cost-effective, as shown in popular autonomous vehicle open data sets, where almost all the vehicles feature one or more cameras as its primary sensor[1][14][56][74][133][155][58][71][28][36]. Despite the aforementioned advantages, cameras are nowhere near perfect, due to high sensitivity to light and weather conditions and the limited capability of acquiring 3D information. Michaelis *et al.*[116] presented how camera can be easily affected by different weather conditions (figure 2.1).

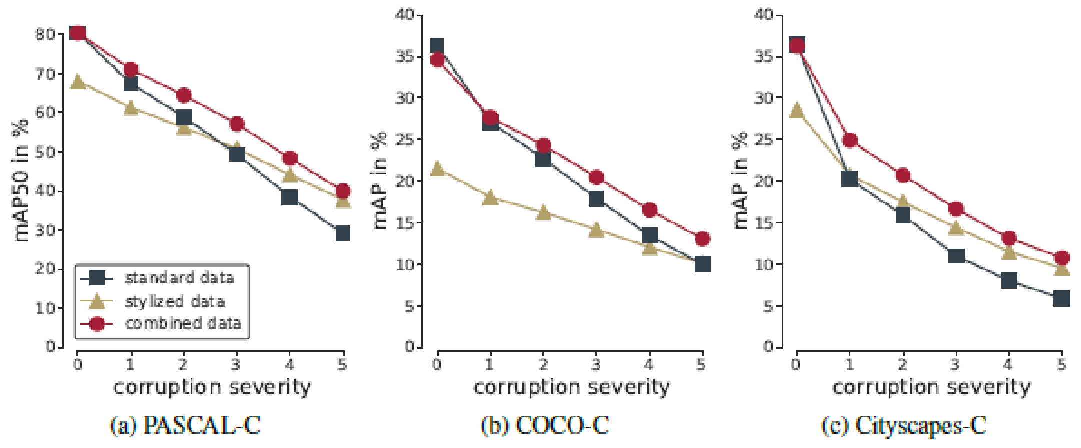


Figure 2.1: Camera performance drop under simulated weather conditions, graph source:[116]

Current research on camera variants, such as IR camera, remains insufficient[36][76]. IR cameras are currently used to detect pedestrians in a low-light environment[76]. IR camera, particularly Long-wave infrared (LWIR) cameras, have the advantage of being weather insensitive[85] and are not subject light conditions, making them suitable sensors for generating 2d images in general. However, compared to a regular camera, it is relatively low resolution (typically under 640 $\times$ 480 pixels, versus a regular camera that has a minimum of 1280 $\times$ 720 pixels), in addition to being quite expensive.

Other sensors like IMU and GPS are commonly installed autonomous driving vehicles as additional sensors. These sensors are mainly used for odometry and vehicle positioning, and some are also used in LiDAR data correction.

	Resolution	Costs	Sensor Type	Sampling Rate	Weather Sensitive	Light Sensitive
Camera	High	Low	Passive	High	Yes	Yes
Stereo Camera	High	Low	Passive	High	Yes	Yes
RADAR	Low	High	Active	Low	No	No
LiDAR	Low	High	Active	Low	Yes	No
Ultrasonic	Low	Low	Active	Low	No	No
IR Camera	High	High	Passive	High	Yes	No

Table 2.1: Table of common sensors and their properties[117]

In summary, Table 2.1 lists standard sensors installed in-vehicle applications and their pros and cons. Also, table 2.2 has a list of sensor models installed on the vehicles that were used to collect the dataset. Based on that information, LiDAR, RADAR, camera, IR camera, and IMU/GPS was included to be part of sensor system at CMHT.

	Camera	LiDAR	Radar	IR Camera	GPS/IMU
CMHT	PointGray 3 Logitech Brio	Velodyne HDL32E	Need Info	FLIR A65	Need Info
Ford Avdata[1]	Flea3 GigE Point Grey Camera 5MP Flea3 GigE Point grey Camera 1.3 MP	Velodyne HDL32E	Navtech CTS350-X	-	Applanix POS LV
Oxford [14]	Point Grey Bumblebee XB3 Point Grey Grasshopper2	Velodyne HDL32E $\times$ 2 SICK LMS-151 $\times$ 2	-	-	NovAtel SPAN-CPT ALIGN
Kitti [56]	PointGray Flea 2grayscale $\times$ 2 PointGray Flea 2 colour $\times$ 2	Velodyne HDL64E	-	-	Velodyne on board
Apollo* [74]	Velodyne VLS-128 Scala2 M16-LSR LEDDARVU Velodyne HDL64E Ulra Puck VLP32C Pandora C16 Leishen Rs-LiDAR-16 (Robosense)	MARS Wissen Technologies LI-USB30-AR023ZWDR	ARS408-21 B01HC	-	ProPak6 PwrPak 7D NV-GI120 Newton-M1
CADCD [133]	Velodyne VLP-32C	Ximea MQ013CG-E2	-	-	NovAtel OEM638 Sensonor STIM300 IMU Xsens MTi-300-AHRS MTi-30-AHRS
Waymo OD [155]	Not mentioned	Not mentioned	-	-	Not mentioned
A2D2 [58]	Velodyne VLP-16 $\times$ 5	Sekonix SF3325-100 Sekonix SF3324-100 $\times$ 5	-	-	Velodyne Onboard
Lyft Lv5 [71]	Not mentioned	Not mentioned	-	-	Not mentioned
nuScenes[28]	Not mentioned	Not mentioned	-	-	Not mentioned
KAIST[36]	Velodyne HDL32E	PointGrey Flea3	-	FLIR A655Sc	OXTS RT2002

Table 2.2: List of sensor-equipped vehicles from open-data and sensor configuration. Note: Apollo does not mention the actual sensor used in data acquisition, but rather listed all the supporting sensor models.

2.2 Sensing Technology

2.2.1 Camera

Camera research is dates as early as the 1700s. Early camera development included the use of light-sensitive materials. For centuries, the working principle behind the camera has never changed, but the media that saves the photon has evolved since then. The first digital camera was invented in 1969[79] by Willard S. Boyle and George E. Smith at Bell Labs. Boyle & Smith developed a Charge-coupled device (CCD) image sensor to replace the light-sensitive materials as the capture media for the intended image. A CCD sensor is composed of an Integrated circuit (IC) with transistorized light sensor array. When lights reflect onto the sensor matrix, it generates a signal and allows the image to transfer into a memory. Each individual transistorized light sensor on the matrix is called a pixel. A CCD's size and the number of pixels determine the image quality, the general principle being the larger the surface area of a CCD, and the higher the density of the pixels it has, the better the image quality[53].

Complementary metal oxide semiconductor (CMOS) sensor is a more recently developed light-sensitive sensor technology used in a modern digital camera. The working principle behind the CMOS is the same as CCD, both featuring a light-sensitive sensor matrix to capture the light and convert it into a digital signal. Compared to CCD cameras, CMOS cameras can be configurated to wavelengths besides the visible spectrum, and allow manufacturing in materials other than silicon such as germanium to extend its sensitivity to a different wavelength. Another advantage is that a CMOS sensor costs less than CCD sensors. Currently, however, as CMOS sensing technology is more susceptible to noise, CCD sensors are usually considered superior than CMOS sensors in terms of imaging quality[166][105], but CMOS's conversion of images to digital data is faster than CCD.

IR Camera IR cameras are special camera that use a microbolometer sensor matrix. As infrared radiation strikes the sensor, the sensor gets heated up and generates electrical resistance to create an image. Unlike cameras that work in the visible light frequency, IR camera works with a wavelength between 7.5 - 14 μm which is outside the visible spectrum.

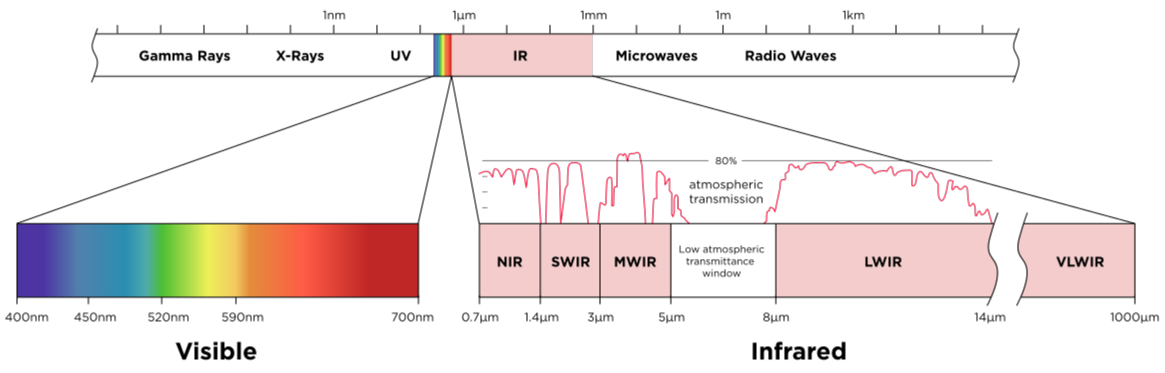


Figure 2.2: IR frequency spectrum, Image source:[47]

IR cameras can be classified into three categories based on the frequency wavelength they work on:

- **Short-wave infrared (SWIR) cameras**: SWIR sensors have the advantage of seeing through fog and haze, and SWIR sensor equipped IR cameras are the only IR cameras that are capable of penetrating clouds. Since the detecting wavelength(between 1.4 μm to 3 μm, figure 2.2) is relatively close to the visible spectrum, the benefit that SWIR sensor provides is relatively limited when compared to other IR sensor options.
- **Medium-wave infrared (MWIR) cameras**: a MWIR sensor provides similar benefits like a SWIR sensor by detecting wavelength between 3 μm to 5 μm. However, the main focus of a MWIR sensor is to provide optical imaging enhancement and gas detection rather than detecting the temperature of an object.
- **LWIR cameras**: LWIR sensors are the most common IR sensor type used in measuring an object's surface temperature. LWIR works in the spectrum wavelength between 8 μm to 14 μm, thus it can be used in a no-light environment, under bad weather conditions, and is able to see through smoke or thick fog. Moreover, a LWIR sensor can detect humans efficiently, making it very compelling in autonomous driving applications.

2.2.2 LiDAR

As discussed previously, LiDAR is an active sensor technology that can provide its own input, in addition to accurately detecting the surrounding environment in 3D. Based on the technology employed, 3D scanning LiDAR can be classified into two main categories: Rotating scan LiDAR and Solid-state LiDAR.

Rotating Scan LiDAR Rotating scan LiDAR is defined as a range finder that measures the point distance . The working principle relies on a 500*nm* to 1100*nm* wavelength laser diode (depending on the application) firing a laser beam towards the desired measuring point, which is reflected (bounces) back towards the laser's point of origin. Thereafter, by deducing the duration of time incurred from firing until the laser receiver reads the bounced laser signal, the distance from the laser diode to the desired measuring point can be computed. A rotating-scan LiDAR(figure 2.3) is designed based on this idea that the addition of a tilting mirror to control the laser firing angle, coupled with spinning the mirror to control the laser beam firing azimuth in an intricately controlled manner, allows the sensor to generate an accurate 3D point cloud scan .

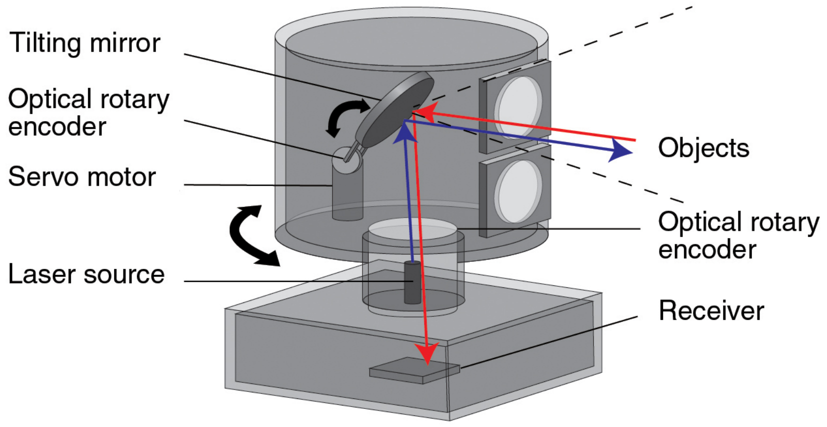


Figure 2.3: 3D rotating scan LiDAR illustration, image source:[140]

Rotating-scan LiDARs have underwent several design variations and iterations, such as in the case of the partial rotating mechanical LiDAR(figure 2.4, also called Risley prism optical steering) that does not possess a movable mirror, and instead, the mirror is replaced by a prism that spins at a high frequency to control laser azimuth. In this particular setup, the tilting mirror can stay fixed within the LiDAR's body. This design variation allows a reduction in LiDAR unit size and an increase in accuracy over conventional rotating-scan LiDARs. However, due to the inherent mechanical limitations, such a partial rotating mechanical LiDARs have a smaller Field of view (FOV) and a shorter lifespan.

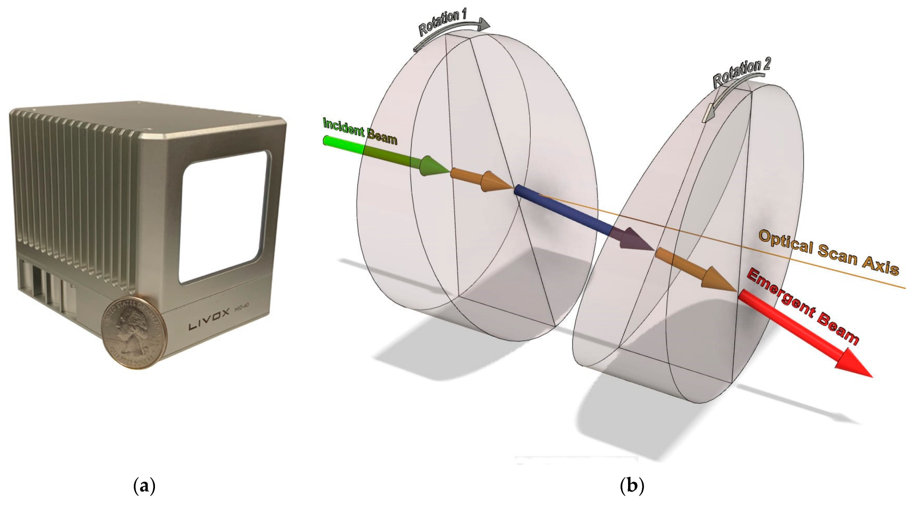


Figure 2.4: Partial rotating scan LiDAR, also known as Risley prism optical steering LiDAR, image source:[26]

Solid State LiDAR LiDAR is an advanced technological concept, featuring Microelectronechanical systems (MEMs)[168], wherein the LiDAR uses micro mirrors mounted on an IC to deflect laser beams. Compared to a rotating scan LiDAR,

A solid-state LiDAR is much smaller and size, and does not contain any complicated rotating systems. Thus, a solid-state LiDAR has the advantage of higher accuracy and significantly reduced costs. But also compared to a rotating-scan LiDAR , MEMs LiDAR technology is not product-ready for autonomous driving vehicles due to two main issues:

- **Reduced Life-span**: Due to the high-frequency vibrations sustained by the micro mirror (typically more than 100 Hz), the lifespan of these IC do not meet satisfactory standards.
- **Small FOV**: Using a single mirror results in a limited FOV. In order to increase the FOV to levels comparable with spinning-scan LiDAR , multiple IC need to be stacked together in order to capture a larger FOV. However, this process faces several technical challenges, including scan angles and projection distortions, that present further conceptual challenges for the post-processing algorithms.

It is postulated that the reasons mentioned above hindered the use of any well-known solid-state LiDAR products in autonomous driving applications up to the time of conducting this research. This would further explain why, in terms of LiDAR products, published datasets contain information on spinning-scan LiDAR only.

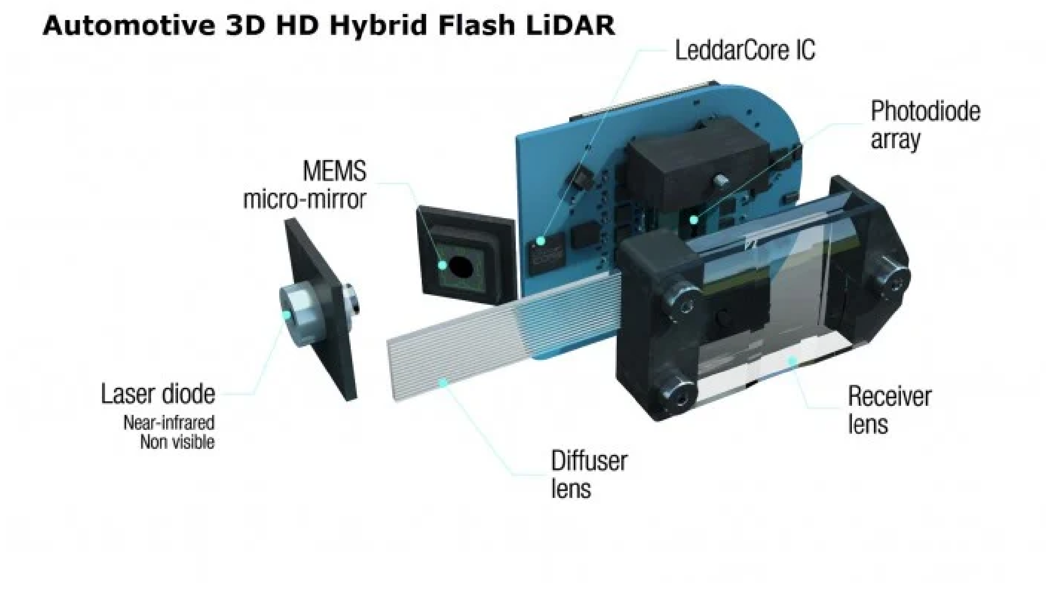


Figure 2.5: Solid state LiDAR, image source:[4]

Attributes	Solid State LiDAR	Rotating Scan LiDAR
Accuracy	+	+
Robustness	+	++
Costs	+++	+
Detection Range	+	+++
FOV	+	+++

Table 2.3: Comparison between rotating scan LiDAR and solid state LiDAR

2.3 Sensor System Setup

CMHT lab has the following sensors installed on the vehicle:

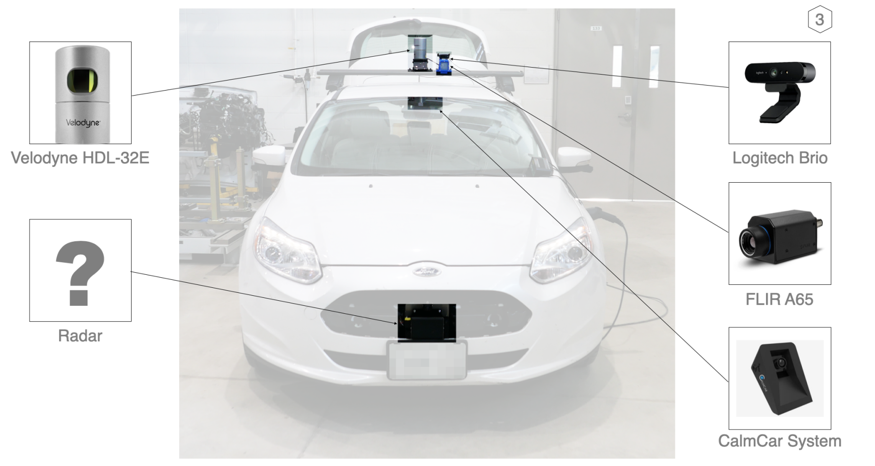


Figure 2.6: Vehicle sensor location

- A top mount LiDAR (Velodyne HDL32)
- A top mount front-facing RGB camera (Logitech Brio) (Interchangeable with Greypoint Grasshopper3)
- A top mount front-facing thermal camera (FLIR A65)
- A front mount (bumper) radar

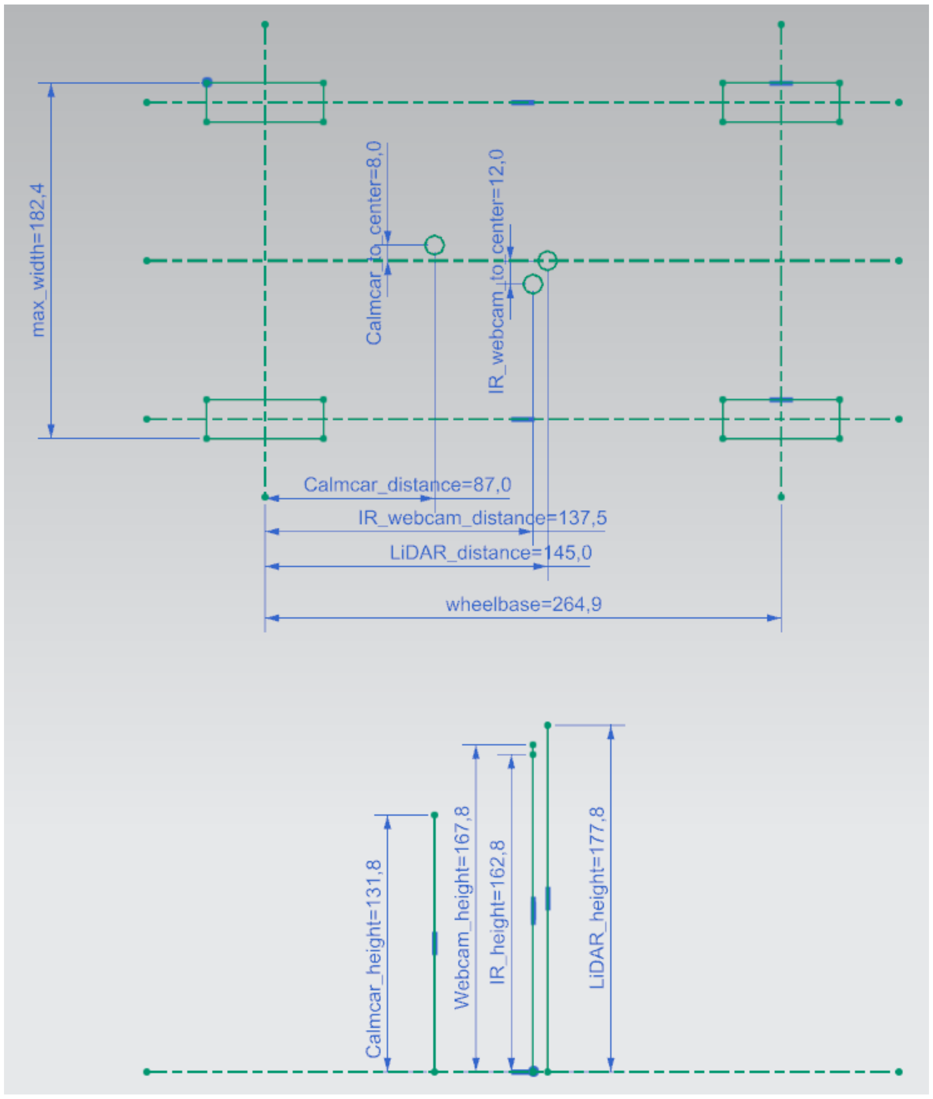
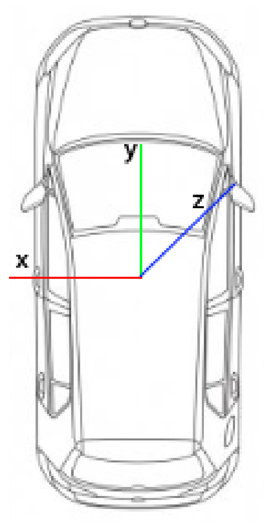
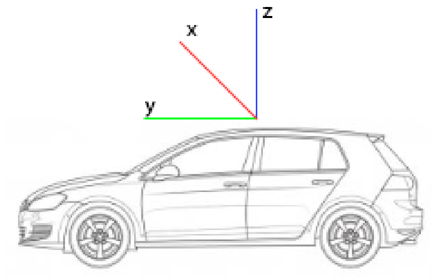


Figure 2.7: Vehicle configuration

This research emphasizes the utilization of cameras and LiDAR. Figure 2.7 illustrates the entire vehicle configuration and the locations of sensors. A local coordinate system is established with LiDAR as the origin, where the XYZ axes are defined as shown in Figure 2.8.



(a) Vehicle top view



(b) Vehicle side view

Figure 2.8: Vehicle coordinate system

2.3.1 Velodyne HDL-32E LiDAR

Velodyne HDL-32E LiDAR (Figure 2.9) was placed on the top of the vehicle. This relatively compact LiDAR uses 32 lasers aligned from +10° to -30° with an effective vertical field of view from +10.67° to -30.67°, scanning 695,000 points per second (or 1,390,000 points per second in dual return mode) to supply the point cloud data in the sensor system. Its main purposes are 3D space object detection, classification, tracking and assisting other sensors in lane tracking. In addition, as LiDAR is an accurate sensor, it also provides ground truth measuring in object positioning for benchmarking other sensors.

Figure 2.9: Velodyne HDL-32E LiDAR

2.3.2 The Camera System

Logitech Brio USB Camera In additional to a LiDAR, the vehicle was equipped with a Logitech USB camera (Figure 2.10c) on the top mount. a very high-performance and low cost imaging sensor. This camera is the main imaging sensor for acquiring 2D image readings, and its main purpose is lane detection, license plate detection, sign detection, in addition to and object detection, classification, and tracking. The camera has a large 82.1 horizontal FOV, and 52.2 degree vertical FOV, can capture at 30, 60 FPS at 720P or 1080P. It can also capture videos in 4K mode at 30 FPS.

Greypoint Grasshopper3 The Greypoint Grasshopper3 camera was selected to serve as the the stereo-vision camera with dual cameras setup, for which synchronized capture driver was developed for it. Table 2.4 shows the specs of the RGB cameras on the vehicle. This camera is a backup option for the main camera with the capability of capturing stereo vision with dual setups, and it is functions the same as the main camera.

Attributes	Logitech Brio	Greypoint Grasshopper 3
Resolution	HD, FHD, 4K	HD, FHD, 2K
Frame rate	30/60Hz	7∼60Hz
Costs	+	++
Diagonal FOV	65, 78, 90 degree	60 degree
Output mode	Mono/RGB	Mono/RGB
Sensor type	CCD	CMOS
Megapixels	13MP	2.8MP
Driver	UVC 2.0	UVC 2.0 / GigE

Table 2.4: Camera specifications

FLIR A65 Thermal Camera FLIR A65 Thermal Camera: The FLIR A65 thermal imaging temperature sensor is a high-end LWIR thermal camera that uses Focal plane array (FPA) and uncooled VOX microbolometers (Figure 2.10a). FLIR A65 has a $25^\circ \times 20^\circ$ horizontal and vertical FOV with a maximum resolution of 640 times 512 pixels @ 30Hz, in addition to a capturing spectrum range between 7.5μm to 13μm. Due to the built-in functionality of thermal drift correction to prevent temperature drifting, which was recommended to remain enabled by the manufacturer's recommendations, one to two frame losses from the rest of the sensor would occur. FLIR A65 thermal camera is a supplementary camera in the system, where its output data is utilized in object detection, classification and tracking,

(a) FLIR A65 IR camera

(b) Grasshopper3

(c) Logitech Brio

Figure 2.10: Optical sensors on the vehicle

2.4 Sensor System Implementation

A capture system was carefully designed to ensure that all sensors are synchronized. The capture system was initially implemented in C++ with the Qt framework and a custom proprietary build data transfer link system. Thereafter, the code was merged into the ROS2 system.

The trigger system was designed around the LiDAR system, based on the requirement that all sensors run at 10Hz and are synchronized. As LiDAR cannot be controlled due to the nature of its mechanical spinning, LiDAR was used as the system capture trigger; that is, whenever the LiDAR encoder reads across 0 degrees, the system will send a capture trigger to other sensor threads and create a hardware capture trigger.

2.4.1 System output

Table 2.5 shows a summarized information about the system output, and table 2.6 provides the detailed output format.

	Frequency	Synchronization	Sync delay δt	Data rate	Format	FOV
Camera	10 Hz	Yes	<8.33 ms	27 MBytes/s	1280 × 720 RGB Image	90
IR Camera	10 Hz	Yes (Software trigger)	<15 ms	6.7 MBytes/s	640 × 480 Greyscale Image	25 x 20
LiDAR	10 Hz	Trigger	0 ms	17 MBytes/s	∼58000 points scan, XYZIR Point cloud	360
Radar*	10 Hz	Yes	<8.33 ms	- MBytes/s	- points scan, XYZD Point cloud	60

Table 2.5: Table of capturing system information, *Radar outputs varies depending on number of object scanned.

Sensor Type	Data
Camera	$X_{\text{CAM}} = \{\mathbf{R} = \begin{bmatrix} r_{11} & \dots & r_{1n} \\ \dots & \dots & \dots \\ r_{m1} & \dots & r_{mn} \end{bmatrix}, \mathbf{G} = \begin{bmatrix} g_{11} & \dots & g_{1n} \\ \dots & \dots & \dots \\ g_{m1} & \dots & g_{mn} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} b_{11} & \dots & b_{1n} \\ \dots & \dots & \dots \\ b_{m1} & \dots & b_{mn} \end{bmatrix}\}$
IR Camera	$X_{\text{IRCAM}} = \begin{bmatrix} x_{11} & \dots & x_{1n} \\ \dots & \dots & \dots \\ x_{m1} & \dots & x_{mn} \end{bmatrix}$
LiDAR	$X_{\text{LiDAR}} = \{\alpha_i, \theta_i, r_i, d_i\}, 0 < i \le n$
RADAR	$X_{\text{RADAR}} = \{\theta_i, d_i, v_i\}, 0 < i \le n$

Table 2.6: Input table

2.4.1.1 LiDAR

LiDAR polar coordinate to Cartesian coordinate conversion LiDAR captures the reflectivity and distance of reflected point alone with the decoder recording azimuth and ring ID. Each ring ID associated an angle with the laser head. For convenience, polar representation inputs were converted to Cartesian representation by using equation 2.1 before the data streaming.

$$P^k_{\text{LiDAR}} = \{ \begin{bmatrix} x^k \\ y^k \\ z^k \end{bmatrix} \} = \{ \begin{bmatrix} d^k_\theta cos(\alpha^k) sin(\theta^k) \\ d^k_\theta cos(\alpha^k) cos(\theta^k) \\ d^k_\theta sin(\alpha^k) \end{bmatrix}, 1 \le j \le n; 0 \le \theta \le 2\pi\} \tag{2.1}$$

d^j is the distance measured by the jth laser scanner, θ_j is the angle of rotation for the scan, and α^j is the angle of pivot for the scan, and n is total number of scans.

Converted LiDAR output is represented in point clouds, a set p^k_M of LiDAR sequence k containing the total M number of tuples $\{x, y, z, i, r\}$: $\{x, y, z\}$ is the coordinate of the returned laser scan with respect to the LiDAR sensor position in the real world, unit in meters. $i \in \{0, 255\}$ is the corresponding laser reflection intensity, and $r \in \{0, 31\}$ is the ring number associated with the returned laser scan ID. On Velodyne HDL32E, there are 32 laser heads. Therefore the ring number ranges between 0 and 31. The firing sequence is pre-determined and arranged in a particular order, and please refer to the appendix for the Velodyne HDL32E LiDAR laser firing timing table. Each tuple $\{x, y, z, i, r\}$ representing a return scan of laser beam, also consider as a point.

Unprocessed LiDAR output data is in polar coordination from the capture system. Raw LiDAR data are packed in ROS Pointcloud2 format(table 2.7).

Data type	Name	Default Value	Note
Header	header		
uint32	seq		
time	stamp		
string	frame_id	”map”	
PointField[]	fields		
string	name	”x”	x position of a reflected point
uint32	offset	0	
uint8	datatype	7	
uint32	count		
string	name	”y”	y position of a reflected point
uint32	offset	4	
uint8	datatype	7	
uint32	count		
string	name	”z”	z position of a reflected point
uint32	offset	8	
uint8	datatype	7	
uint32	count		
string	name	”i”	Intensity of a reflected point
uint32	offset	12	
uint8	datatype	7	
uint32	count		
string	name	”r”	ring number, velodyne laser ID
uint32	offset	16	
uint8	datatype	2	
uint32	count		
bool	is_bigendian		
uint32	point_step		
uint32	row_step		
uint8[]	data		
bool	is_dense		yes

Table 2.7: LiDAR Pointcloud2 message format. Please note that at CMHT, the standard XYZI pointcloud2 msg was not used. Instead, a custom XYZIR pointcloud2 msg was created that embedded LiDAR ring ID information.

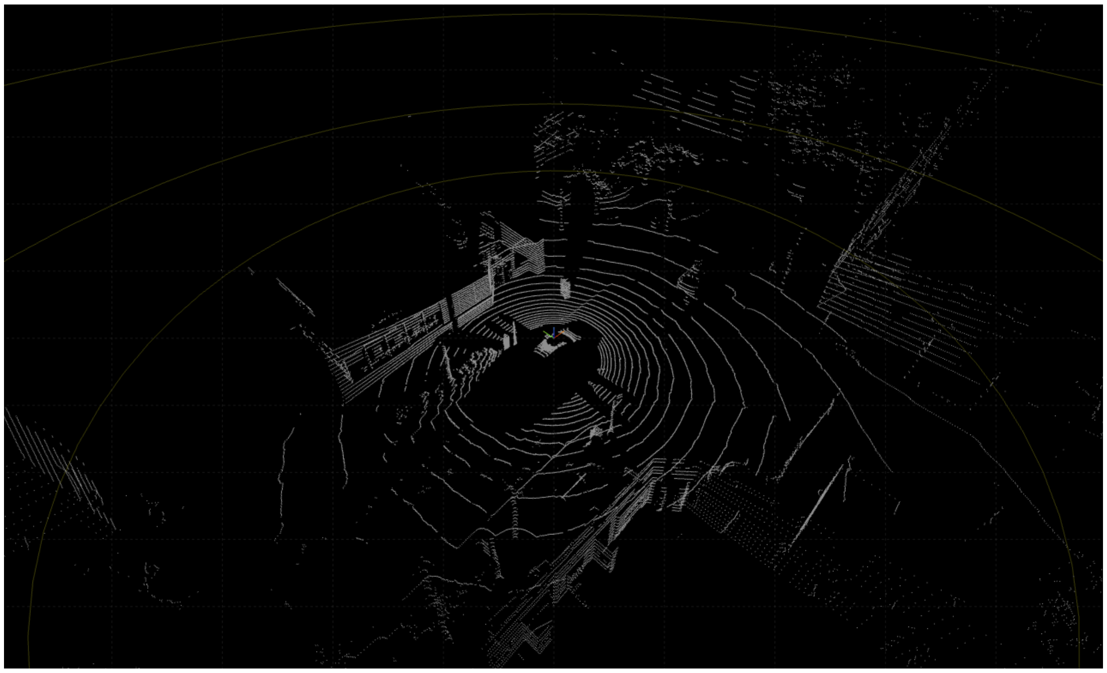


Figure 2.11: Visualization of LiDAR outputs

Convert LiDAR Cartesian Representation into Spherical Dense Representation LiDAR data are commonly stored in $\{x, y, z, i\}$ set, where x, y, z is the set of coordinate in Cartesian plane respect to the LiDAR sensor coordinate system, and i is the respect reflectivity (or intensity in come context) associated with the respect data point. This representation is normally unpacked, therefore not suitable for 2D-inputs Neural network (NN) to process. Spherical dense representation is a way to 2D visualize 3D LiDAR data, thus for this research, all LiDAR data inputs are converted to spherical dense representation.

In order to convert spherical dense representation, a *ring number* $\times$ *data count* $\times$ *number of channels* tensor was first created to store the LiDAR data. In this research, there are four channels: x, y, z, i, and the ring number is 32 because of the number of laser IDs from the Velodyne HDL32e LiDAR. When data are streaming in, the tensor starts filling from left to right, where the row position is based on ring id r, and the datum is filled according to the channel x, y, z, i(See Figure 2.12 for the data structure illustration).

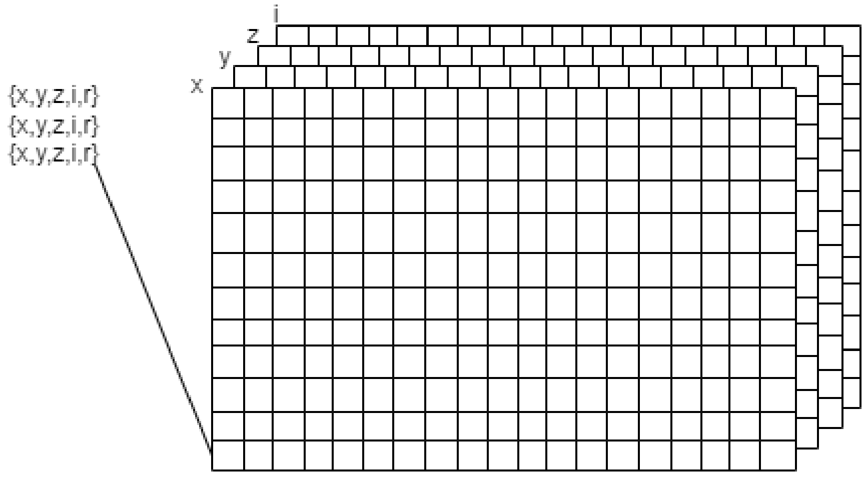
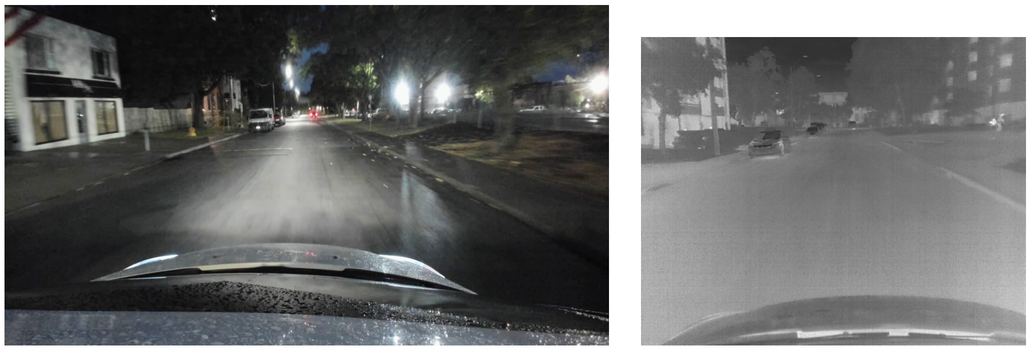


Figure 2.12: Spherical dense representation illustration

2.4.1.2 Camera and IR Camera

Both RGB camera and IR camera outputs are in pixel coordinate $\{u, v\}$. Standard ROS2 image messages was used to embed image information, and data are captured in RGB uncompressed format.



(a) Camera outputs (b) IR camera outputs

Figure 2.13: Imaging sensor outputs example.

2.5 Real-Time System Implementation

The primary capture system is developed in C++ with the QT framework. There are two distinct versions of its implementation. Initially, the system was constructed using boost TCP and UDP protocol libraries, including a TCP server that managed all connected devices and processing nodes, and P2P UDP to handle the extensive data transfer between nodes. Later, a modification was made to make it compatible with the ROS2 system. This ROS2 version maintains the original triggering system design but adds a ROS2 wrapper to facilitate data transfer. It continues to be maintained because its API is more accessible to researchers in labs. However, the ROS2 version's performance is slightly inferior to the custom-built TCP/UDP version. Moreover, future development on the boost protocol version might be challenging due to limited support and complexity in the node data transfer process. In contrast to the difficulties posed by the custom build, ROS2 has made this process more transparent and streamlined, even though its performance might be slightly worse.

Figure 2.14 provides an illustration of the triggering system design where the LiDAR sensor is used as the trigger generator. Whenever the rotary angle reaches the 0-degree mark, the LiDAR class dispatches a signal to all other sensors to synchronize their capture.

The vehicle's camera uses a GiGE interface, containing an internal hardware trigger API that precisely controls when the camera shutter captures an image. Similarly, the IR camera possesses this feature, making the entire system revolve around the LiDAR sensor. All sensor capture and interfacing classes operate on the highest-priority threads, and the data capturing process is embedded within a critical section.

The data captured by the sensors are temporarily stored in the main PC and transferred to a recorder class for processing. This recorder class runs on a thread with less priority than the main sensors, and it only processes data during the off time when sensors are not capturing. Once the sensor data is copied from the capturing buffer to a stack buffer for processing, the remaining CPU time is utilized to compress the data and send it to other nodes via either UDP or ROS2 interface.

Communication between nodes (from different machines) employs TCP protocol for reliability, while data sending uses P2P UDP with hardware compression. Then, the message (sensor data) is reconstructed in the destination nodes for further processing.

In Figure 2.14, there is an additional System on Module (SOM) (Nvidia Jetson TX2) utilized to control the IR camera. This SOM was later removed in the ROS2 version, and the controlling class was transferred to the main processing unit. This shift was made possible due to an upgrade in the main controlling PC's specifications, where an improved CPU with hardware compression support and an increased number of processing cores stabilized the system's throughput, eliminating the need for additional SOM support.

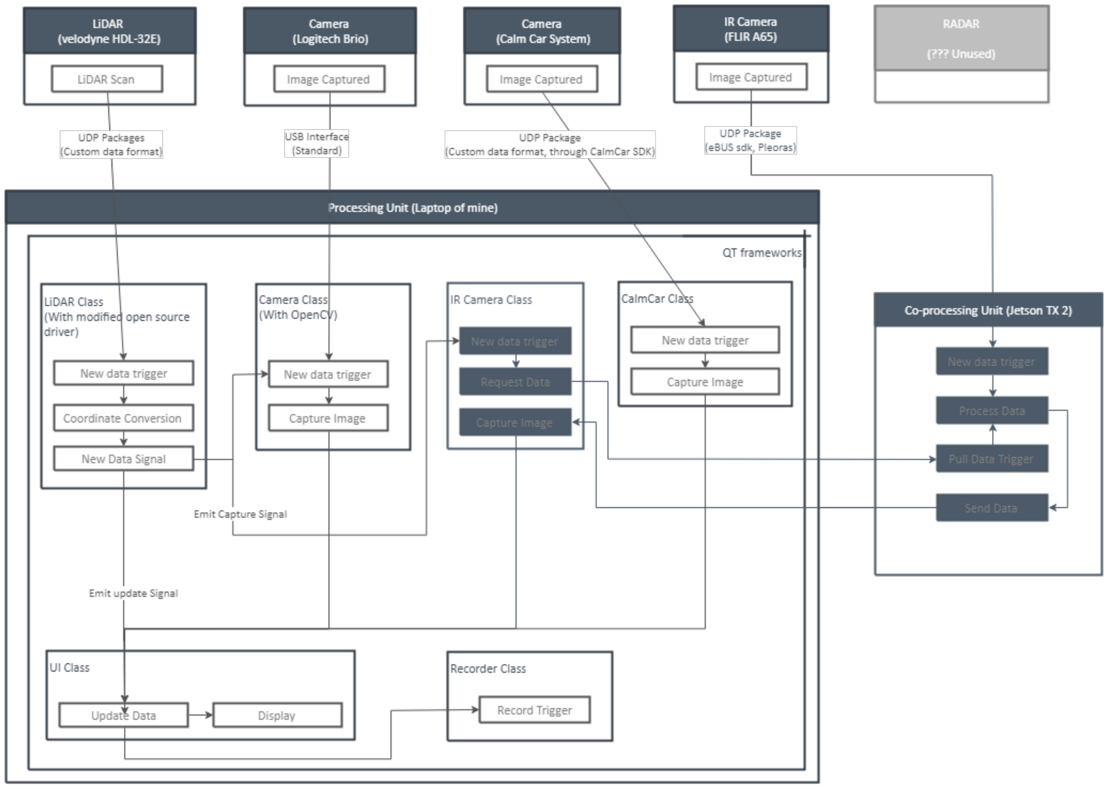


Figure 2.14: System design illustration

2.6 Results

2.6.1 Stereo Camera System Performance Test

A simple experiment was designed to estimate the two cameras' capturing time difference $\delta t_{capture}$. First, a cellphone with a screen refresh rate of 120Hz was placed, and both cameras were pointed at it (Figure 2.15). Then, a timer counter was started on the phone's screen. After the capture system was triggered, the difference between the two capture instances was measured by reading the number off both camera captures (Figure 2.16).

$\delta t_{capture}$ was estimated under the following assumptions:

- Phone screen is running at 120Hz, approximately 8.33 ms per number change.
- If the number reading is the same, then the result difference is within 8.33 ms.

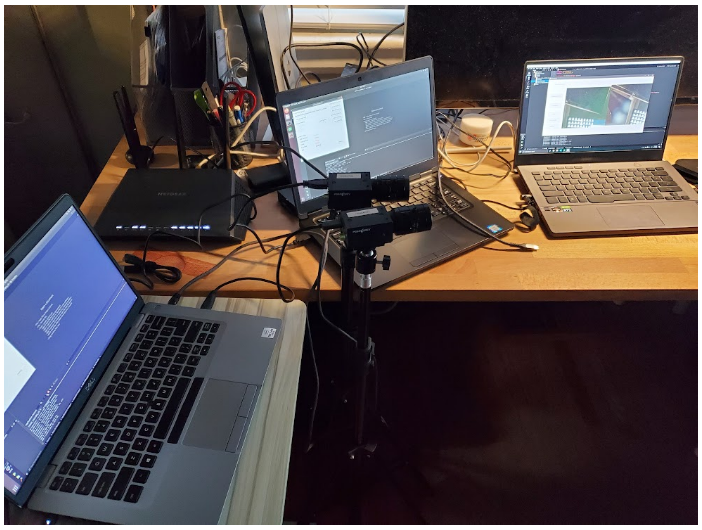


Figure 2.15: Capture synchronization test setup

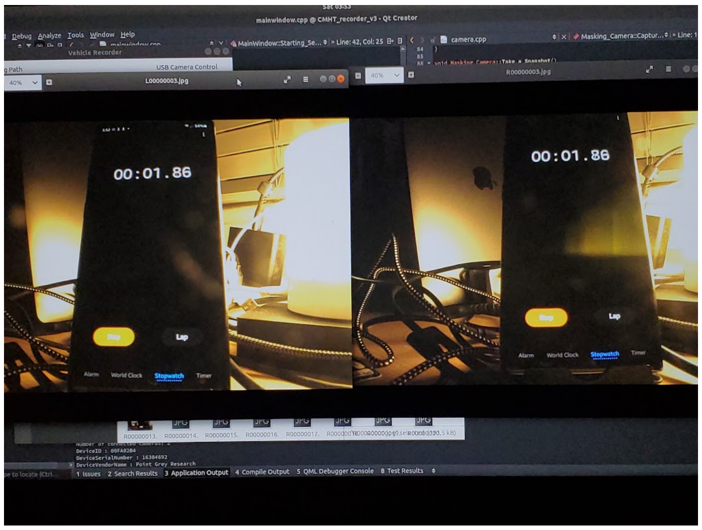


Figure 2.16: Capture synchronization test result sample

By performing this simple experiment and sampling it n=100 times, it was concluded that both cameras have a capture $\delta t_{capture} \leq 8.33ms$. Since the system is running at 10Hz, this result is considered satisfactory.

2.6.2 System Performance Stability Test

A system delay measurement test was conducted to assess the stability of the system's throughput. During the test, the timestamps of all received input packets were recorded, and the time difference(δt) between consecutive frames was analysed. The experiment was run for 350 samples, lasting 35 seconds. The implementation demonstrated an impressive average system delay of 99.83 ns, with a maximum delay of 121.21 ns and a minimum delay of 77.24 ns. The observed jitter in the system was remarkably low, measuring only about 0.000001% relative to the frame time interval, which was 100 milliseconds. This minuscule jitter can be considered negligible, indicating a highly stable system.

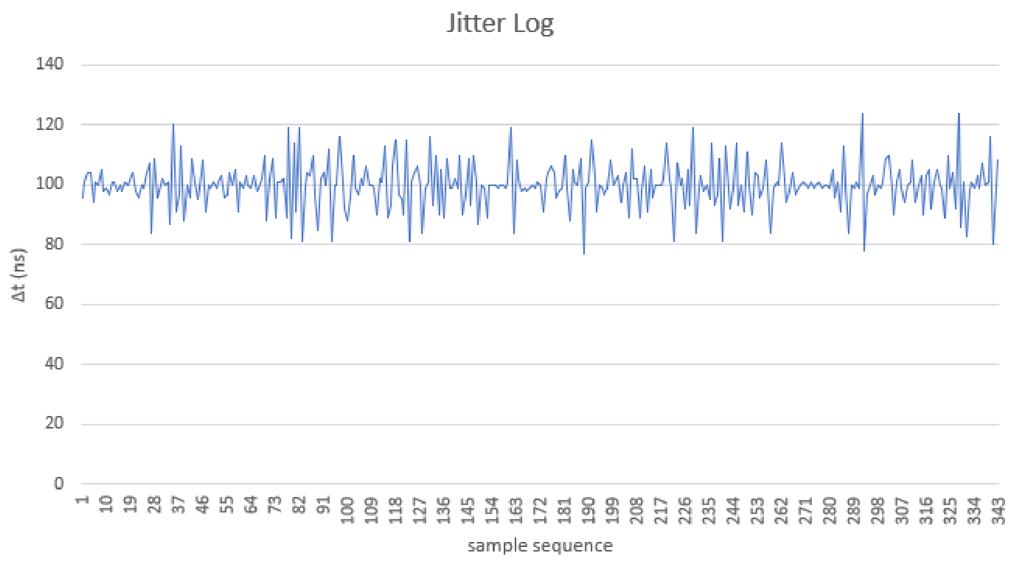


Figure 2.17: Jitter measurement

2.7 Conclusion

In conclusion, sensors are essential components of the smart vehicle, and picking only one sensor is unlikely to compose a robust, weatherproof sensing system. Therefore, it is always ideal to pick multiple sensors, taking advantage of the attributes of different sensors to cover the weaknesses of a system that depends on one type of sensor. However, simple sensor stacking would not work, which demonstrates the need for a process called “sensor fusion” to utilize the full potential of a multi-sensor approach. The next chapter will discuss the fusion of multiple sensors and the data processing entailed .

Chapter 3

Sensor Fusion

3.1 Introduction

Sensor fusion enhances the overall reliability and performance of the system but introduces some inherent challenges: high cost, intensive processing, and data alignment. Modern technological advancements have enabled the mass production of sensors at relatively low costs, and powerful processors allow for the real-time computation of complex systems. However, data alignment remains an issue: each sensor operates in its own coordinate system because it originates from the sensor's installation location, and sensors are installed on different parts of the vehicle. Additionally, LiDAR captures data in 3D space, while the camera captures data in 2D space. To unify the coordinate systems, alignment techniques must be applied. This chapter discusses the fusion techniques specifically tailored and implemented for aligning the CMHT sensors. Furthermore, an introduction to the CMHT dataset is provided. This complementary dataset combines LiDAR, IR camera, camera, GPS, and Radar readings captured at different times of the day, under various weather and lighting conditions, in urban and highway driving scenarios.

3.2 Sensor Fusion

The first sensor fusion functional model was defined in the mid of 1980s by US Joint Directors of Laboratories (JDL)[23]. The very first model defined a five-level data fusion model that was designed for military and intelligence service, later the model was redefined for general purpose[152] in 1998, and expanded into a six-level data fusion model (Figure 3.1) in 2003[24][22].

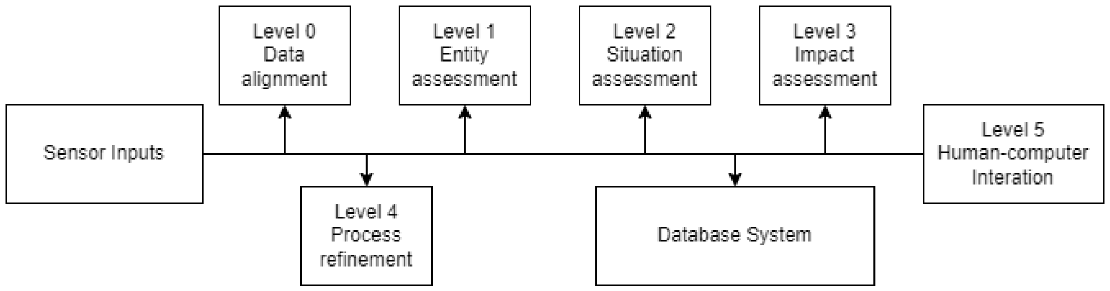


Figure 3.1: Original JDL fusion level diagram

Refined JDL six-level sensor fusion functional model:

- Data alignment (Level 0): data alignment ensures that sensor data are synchronized and aligned correctly. This alignment includes time synchronization, spatial alignment and dynamic alignment. Time synchronization synchronizes the sensor base on time; spatial alignment aligns the sensor by its relative position; and dynamic alignment aligns different sensors based on their dynamics.
- Entity assessment (Level 1): extracting and characterizing object features from the data such as shapes, size, classification and more.
- Situation assessment (Level 2): situation assessment is the process of using sensor data information to understand the relation amoung the entities[21]. For example understanding if the vehicle is on a highway.
- Impact assessment (Level 3): estimating or predicting the effects of the object on the user.
- Process refinement (Level 4): process refine is by using existing data information to improve the entire process for better performance/results.
- User refinement (Level 5): user/human fine tuning the process to improve the performance.

- Mission refinement (Level 6): mission-based fine tuning process to complete the objective better.

JDL sensor fusion model provides a very general abstraction of the sensor fusion process, and despite it originating from the military, the model is now extended into multiple domains such as autonomous driving[145][115][106] and robotics[86][146][153] for tasks like localization and perception.

Sensor fusion in autonomous driving has a specific goal: autonomous driving vehicle takes the homogeneous or heterogeneous sensor inputs, combine and refine the inputs for a better understanding of the surrounding (i.e. object classification, object tracking, vehicle localization), and improve the sensor data quality. There are a lot of sensor fusion autonomous driving research, and based on the architecture of system, those sensor fusion techniques are commonly categorized into three categories[48][163]:

- Data level sensor fusion(figure 3.2): the fusion process happens at the sensor data level. At this level, the sensor data is first associated, then perform the information extraction. Data level sensor fusion is the most straight forward sensor fusion, also simple to implement and relatively less computational complexity. Some data-level fusion models[83][16][113] are focusing on solving vehicle position and localization problems by IMU with GPS. Additionally, research has been conducted on implementing data level sensor fusion in low computing power devices, for example Suhr *et al.*[154] implemented a low-cost vehicle localization system fusing GPS, IMU, wheel encoder, map and pre-processed front camera information at data level on the Intel®Edison System on Chip (SOC), a very low-power atom chip. Data-level sensor fusion models can also apply to solve complex problems like object tracking, for example Zhen Jia*et al.*[80] proposed two camera data-level sensor fusion model to track moving targets. However data level sensor fusion also comes with a few disadvantages: sensor fusion model needs to be precise to have the result accurate; and data level sensor fusion model usually does not deal with sensor failures, where failed sensor inputs can affect the fusion outcomes;

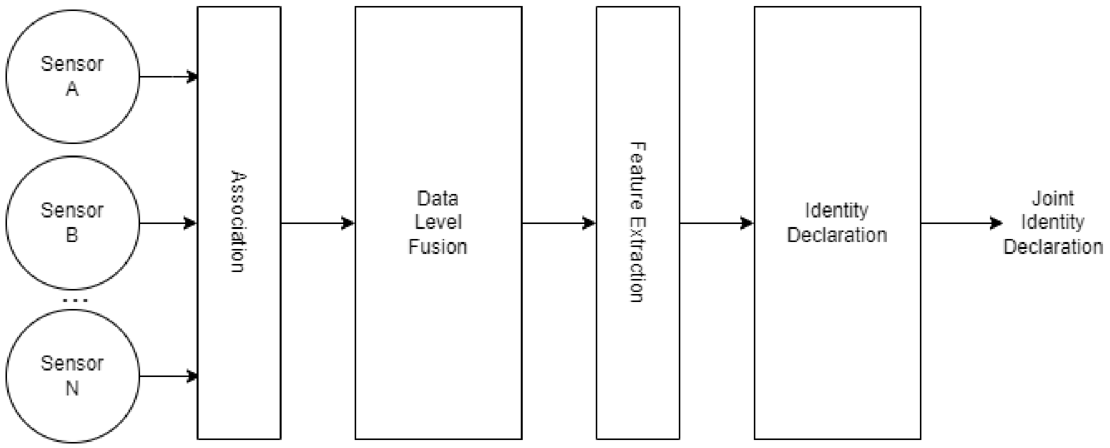


Figure 3.2: Data level sensor fusion model

- Feature level sensor fusion(figure 3.4): multiple sensor data are feature extracted first, then associated and processed. The advantage of feature level sensor fusion is the model correlates different features from the sensor and generate a set of salient features for the final sensor outputs, also because the feature extraction happened after the data collection, therefore it uses less bandwidth to transfer the data when compared to data level sensor fusion. The model is very popular in autonomous driving research and especially common within works that fuses high-resolution sensors such as LiDAR and cameras[35]; for example Qingquan Li *et al.*[102] developed a feature level heterogeneous sensor fusion model where features such as RGB information are extracted from the camera, and road information with depth from the LiDAR, then combine the features together to detect lanes; Thao Dang *et al.*[42] work developed a feature level homogeneous sensor fusion, where the research extract features from multiple stereo cameras to create a stereo image to retrieve depth and stereopsis information, then by using Kalman-filter on the features to track objects on the road; Other heterogeneous sensor fusion network works like Radar to camera [84][124][98] also well researched. Another expanding field within feature level sensor fusion for autonomous driving focuses on neural network-based research[84],[124],[98]. This specific area is rapidly advancing due to two primary factors: the availability of open-source dataset (Ford drive, KITTI and others) becomes more accessible for the neural network to train on; and succession of image-based segmentation and classification neural network architecture such as Region-Based Convolutional Neural network (R-CNN), fast R-CNN, and YOLO. Those autonomous driving NN based decision level sensor fusion follows a general pipeline (shown in figure 3.3, and the pipeline can be modelled by the feature level sensor fusion model.

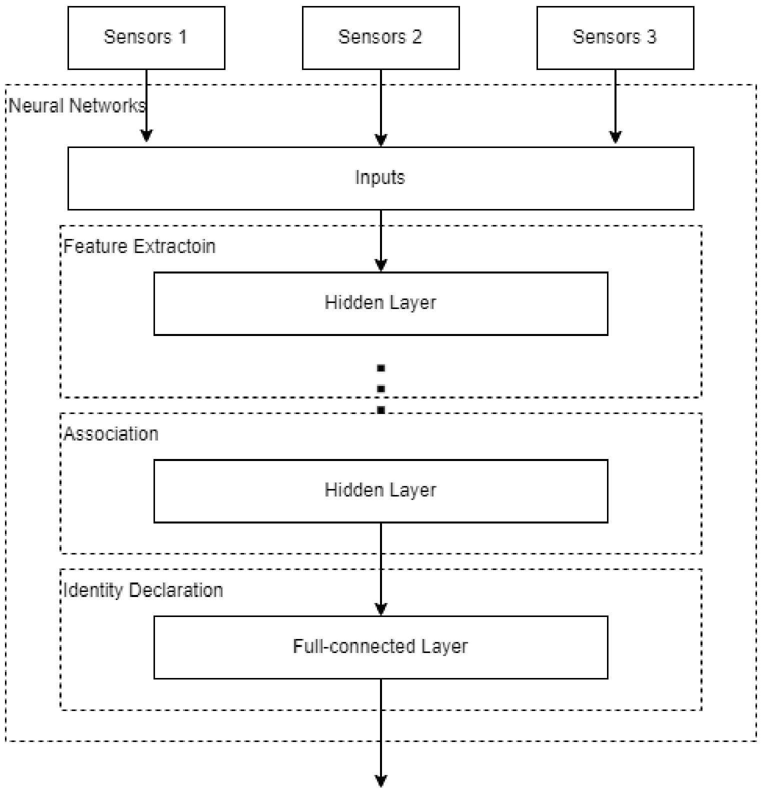


Figure 3.3: Neural network based sensor fusion approach general process pipeline (Feature level sensor fusion)

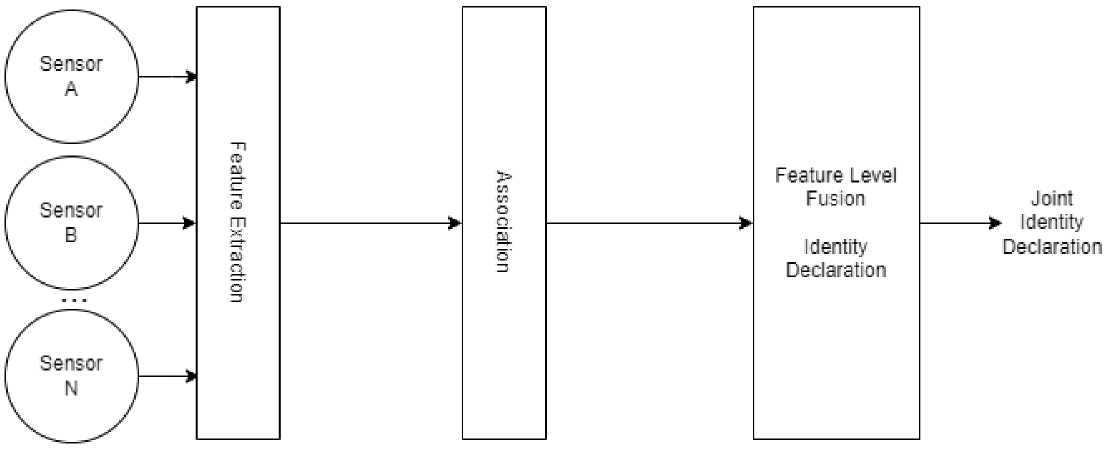


Figure 3.4: Feature level sensor fusion model

- Decision level sensor fusion(figure 3.5): each sensor information is feature extracted and processed individually,and the results are associated and produce the final output. The process is similar to human biological making decisions, thus named decision level sensor fusion. Comparing to other sensor fusion models, decision level sensor fusion for autonomous vehicle is mostly focusing on NN based solutions, for example Sang-Il Oh *et al.*[127] proposed a LiDAR with camera sensor fusion model: LiDAR and camera are regional proposed separately, then feed the Region of Interest (ROI) into two different CNN, and lastly fused the results together; Alireza Asvadi *et al.*[8] proposed a similar but slightly more advance heterogeneous decision level sensor fusion model where LiDAR features dense and reflection are fed into two separated YOLO networks for vehicle detection, and camera RGB information is feed into another YOLO network, then results from three YOLO networks are combined and going through a detection fusion to complete the sensor fusion process. Michael Manz*et al.*[110] work is taking a traditional approach, where he fuses LiDAR with a camera to determine the drivability of the road for the vehicle.

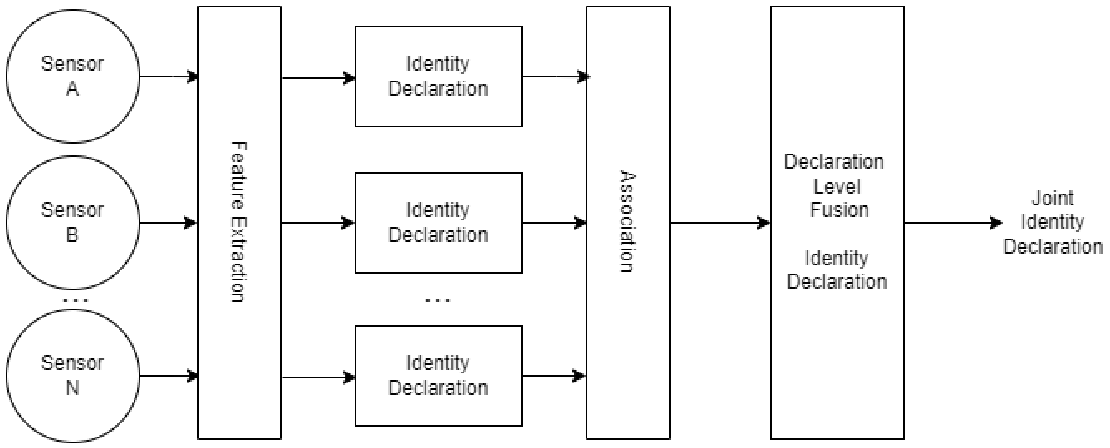
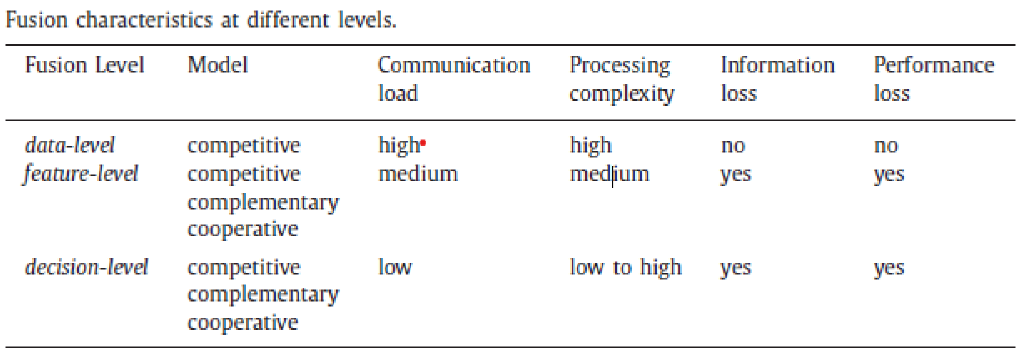


Figure 3.5: Decision-based sensor fusion model

There is no definitive answer to which sensor fusion model is superior, there are still a lot more to explore in the research area. Works done by Gravina *et al.*[61] has a table summary of the characterises of each sensor fusion level (figure 3.6).



Fusion characteristics at different levels.

Fusion Level	Model	Communication load	Processing complexity	Information loss	Performance loss
data-level	competitive	high	high	no	no
feature-level	competitive complementary cooperative	medium	medium	yes	yes
decision-level	competitive complementary cooperative	low	low to high	yes	yes

Figure 3.6: Fusion characteristics at different levels, table source:[61]

3.3 Sensor Calibration

Sensor calibration is required to optimize the data capture and minimize measurement error. The work focuses on calibrating the optical sensors, mainly meditating the distortion effect from the different types of lenses. The accuracy of the object coordinate in the image affects the overlay performance in the following data-level sensor fusion process.

On the other hands,LiDAR calibration is not as common as camera calibration due to it usually happens during manufacturing through empirical and proprietary procedures. Also, the calibration models are highly manufacturing-dependent and difficult to create a generalized model. There are some works[63], [120], [9] done on this topic, though most of them are for aviation, and the methodology is difficult to transfer to the autonomous driving vehicle LiDAR calibration.

3.4 Overview

At CMHT, a feature-level sensor fusion pipeline was designed to extract features from LiDAR, camera, and IR camera, and then correlate these sensor features using spatial data alignment. This chapter focuses on discussing the data alignment, while the next chapter will concentrate on feature extraction. The LiDAR to camera alignment process pipeline is shown in Figure 3.7 with three steps after taking the sensor inputs: After acquiring sensor data, camera calibration was performed on the camera inputs to correct any image distortion; at the same time, the LiDAR to camera projection was calculated; finally, both results were combined to compute the LiDAR to camera spatial alignment. The process of aligning LiDAR to IR camera is exactly the same as for an RGB camera, thus they are discussed in the same manner.

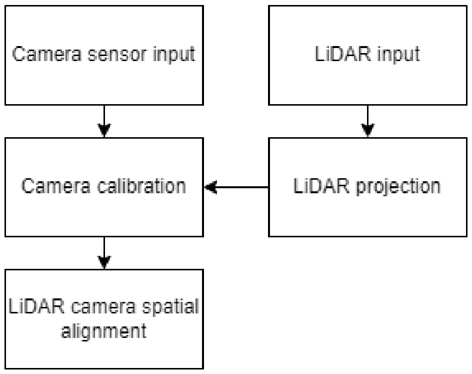


Figure 3.7: LiDAR to camera spatial alignment procedure

3.5 LiDAR to Camera Spatial Alignment

Expanding the camera onto a bird-eyes view or any other similar 2D to 3D prediction requires a strong assumption. The result is often less accurate because the depth is approximated/citemiethig2019leveraging.

Thus for this research, LiDAR was projected onto the camera coordinate for better position estimation. Figure 3.8 shows the overview of the projection process. The LiDAR to camera projection requires the follow steps:

- Translate the LiDAR coordinate into world coordinate.
- Translate the world coordinate into camera 3D space coordinate
- Project the camera 3D space coordinate onto pixel coordinate

Now the LiDAR point clouds in $[x, y, z]$ coordinate is mapped onto camera pixel coordinate $[u, v]$. For the camera part, it only needs one step which is mapping the camera distorted coordinate to the camera pixel coordinate(Detail in section 3.5.1).

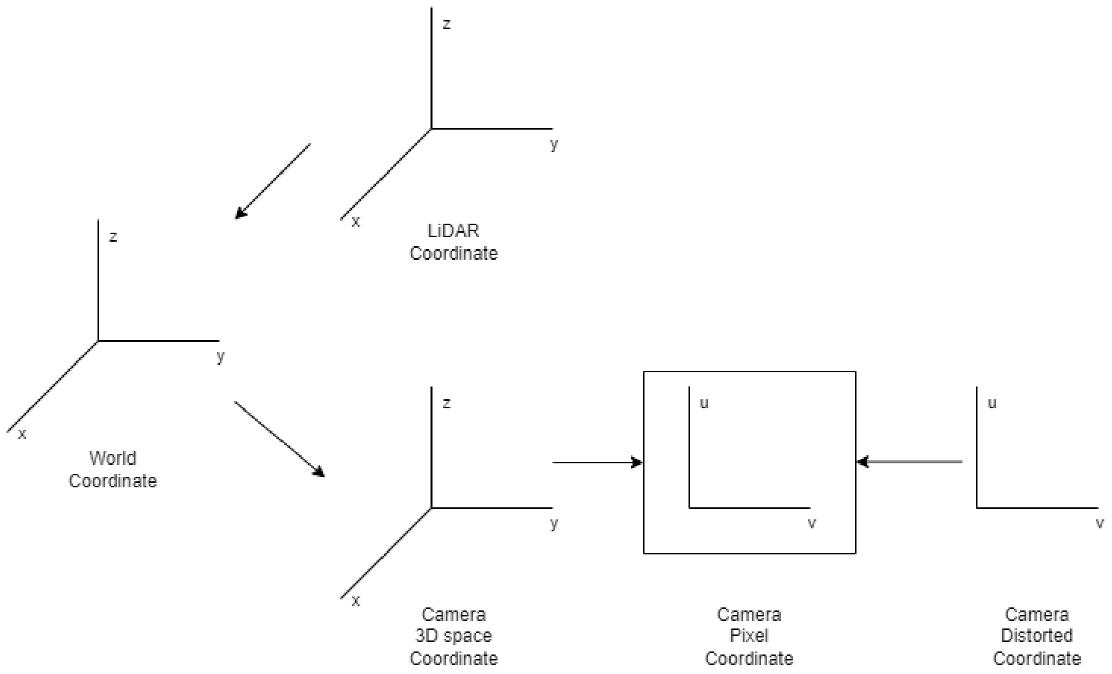


Figure 3.8: Coordinate system illustration

Since the world coordinate was defined to be the same as the LiDAR coordinate in this work, the first step can be omitted. Figure 3.9 shows the actual process of LiDAR to camera in this work.

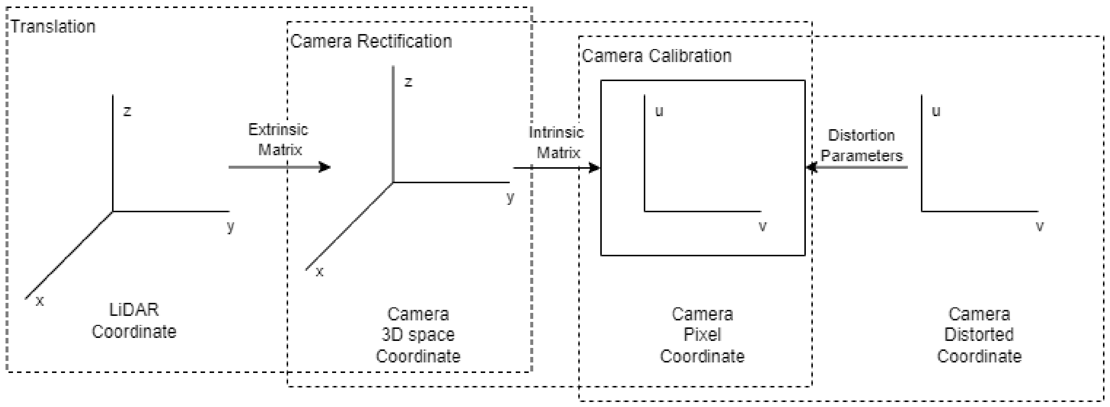


Figure 3.9: Coordinate system illustration

Transformation equation 3.1 is used to convert the LiDAR coordinate (X,Y,Z) into the camera coordinate (U,V). In this equation, the rotation matrix is represented by **R** and the translation matrix by **t**. (f_u, f_v, u_0, v_0) is the camera intrinsic parameter.

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} f_u & 0 & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{3.1}$$

3.5.1 Camera Calibration

Modern cameras use lenses to improve the imaging capturing quality and provide various utilities. For example, most robotic and autonomous vehicle applications use fish-eye lenses simply because it allows the camera to capture a larger field of view with limited light-sensitive sensor surface area.

However, the fish eye also came with a significant drawback where the convex-shape lenses distort the captured images (Figure 3.10).

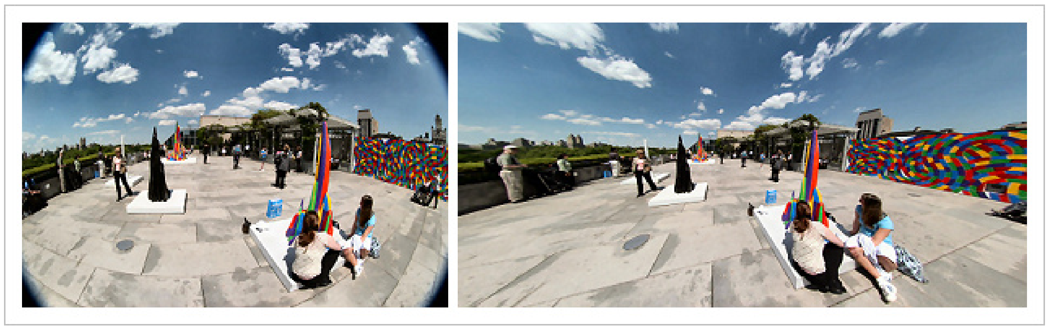


Figure 3.10: Fisheye view example. [10]

The distortion introduced by the concavity of the lens is called radial distortion. Barrel, pincushion, and moustache distortion are three kinds of radial distortions.

- **Barrel distortion**: the lens has a concave shape. The affected image's center is stretched, and the edge of the image is compressed in the outer direction from the centre. The image appears to be mapped on a spherical surface.

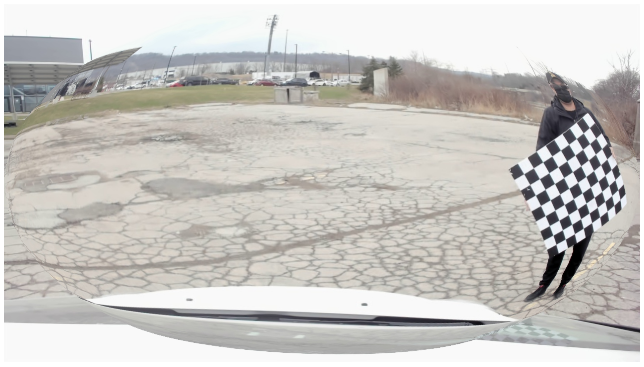


Figure 3.11: Example of a barrel distortion, image is artificially distorted for demonstration purpose.

- **Pincushion distortion**: the lens has a convex shape, and resulting image compressed around the centre, and stretching out on the edge. Fisheye lens is a common application for this kind of distortion.

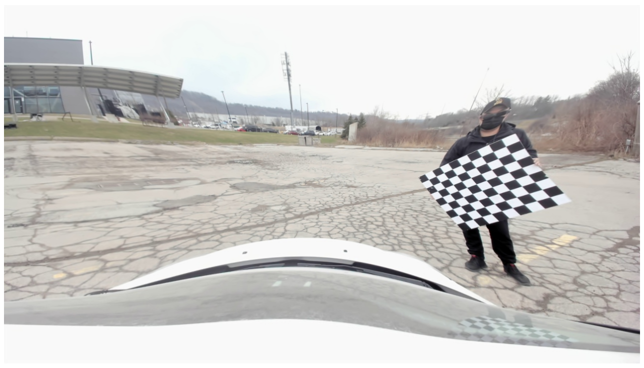


Figure 3.12: Example of a pincushion distortion, image is artificially distorted for demonstration purpose.

- **Moustache distortion**: the lens is deformed, and image result is a mix of both distortions together.

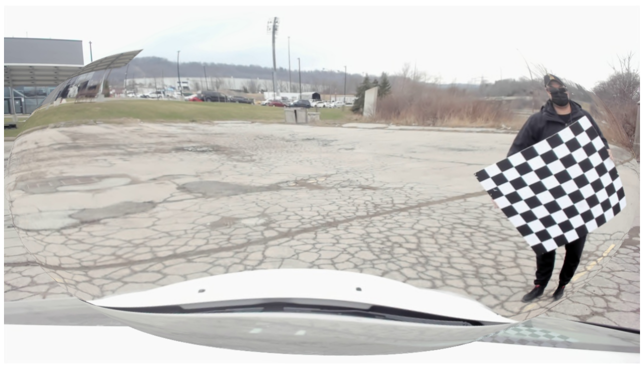


Figure 3.13: Example of a moustache distortion, image is artificially distorted for demonstration purpose.

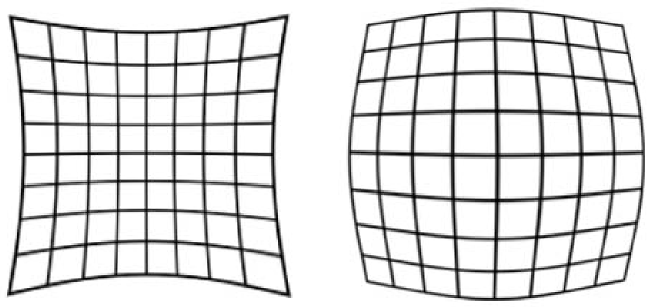
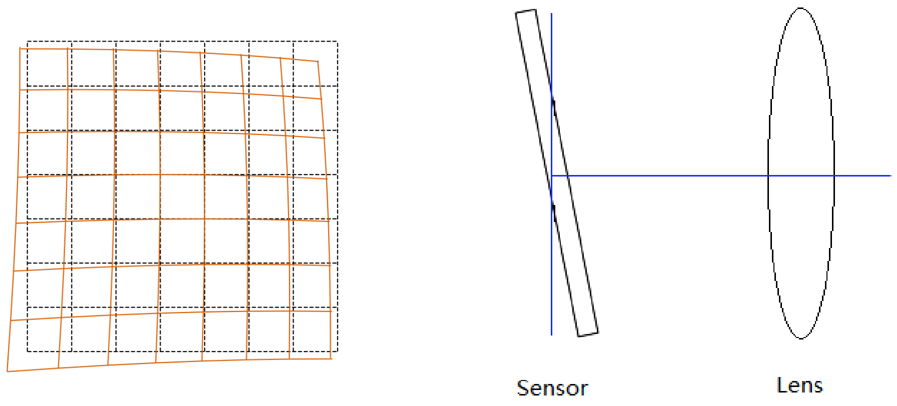


Figure 3.14: Distortion Illustration. Left: pincushion distortion, Right: barrel distortion.

Tangential distortion(also called de-centred distortion) is introduced from the camera sensor is not being perfectly parallel to the lens (figure 3.15), thus skewing the image.



(a) Image distortion (b) Camera section view

Figure 3.15: Tangential distortion illustration.

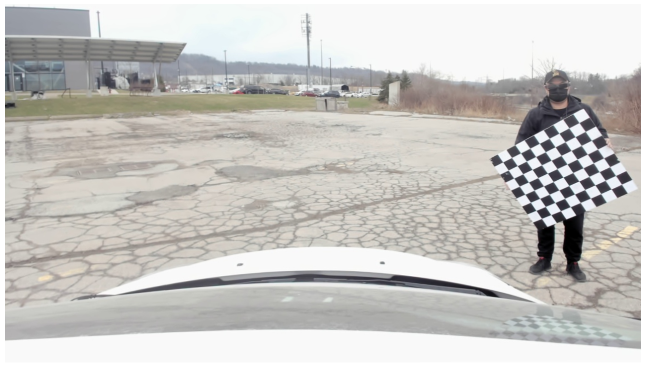


Figure 3.16: Example of a tangential distortion, image is artificially distorted for demonstration purpose.

A developed model can un-distort this distortion effect by carefully tuning the parameters, and the parameter tuning process is called camera calibration. Early camera calibration method was developed by Duane C. Brown *et al.*[27] in the early 1970s. The calibration method used plumb-lines to compute the parameters in a non-linear collinearity equation model. But the first adopted method was developed by Roger Y. Tsai *et al.*[160], his work introduced a two-stage process by combining linear and non-linear techniques, and the method is refined by the later works[100], [169], and [66]. Zhang *et al.*[179] improved the calibration method by allowing the camera and planar pattern to move around freely while still being able to accurately model the distortion, and due to its simplicity and autonomy, this is one of the most popular calibration methods used by this day.

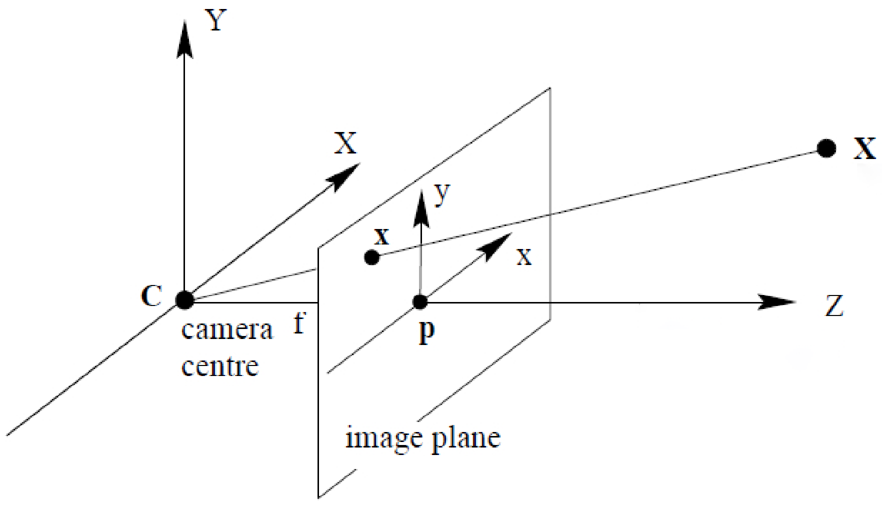


Figure 3.17: Pin hole camera illustration

3.5.1.1 Pinhole Camera Model

Before discussing Zhang's calibration technique, it is essential to understand the relation between the pixel position of an image captured by a camera and its real-world spatial information. An $m \times n$ resolution image contains many points, and denote each point position $\begin{bmatrix} u & v \end{bmatrix}^T$. u and v are the pixel indices in the computer system, and $\begin{bmatrix} x & y & z \end{bmatrix}^T$ denotes the point in the real-world 3d coordinate. Based on the figure 3.17, equation 3.2 and equation 3.3 can be formulated by using trigonometry.

$$u = -\frac{fx}{z} \tag{3.2}$$

$$v = -\frac{fy}{z} \tag{3.3}$$

Equation 3.4 can be derived by combining both equations 3.2 and 3.3, and equation 3.4 shows the relation between 3D real world coordinate $\begin{bmatrix} x & y & z \end{bmatrix}^T$ and pixel coordinate $\begin{bmatrix} u & v \end{bmatrix}^T$ in the image, where f is the focal length of the camera.

$$\begin{pmatrix} u \\ v \end{pmatrix} = -\frac{f}{z} \begin{pmatrix} x \\ y \end{pmatrix} \tag{3.4}$$

3.5.1.2 Brown-Conrady Radial and Tangential Un-disortion Model

Brown-Conrady model[39] was used to reverse the camera lens distortion effect.

$$\begin{aligned} x_u = x_d + (x_d - x_c)(K_1 r^2 + K_2 r^4 + \dots) + [p_1(r^2 + 2(x_d - x_c)^2) \\ +2p_2(x_d - x_c)(y_d - y_c)](1 + p_3 r^2 + p_4 r^4 \dots) \end{aligned} \tag{3.5}$$

$$\begin{aligned} y_u = y_d + (y_d - y_c)(K_1 r^2 + K_2 r^4 + \dots) + [2p_1(x_d - x_c)(y_d - y_c) \\ +p_2(r^2 + 2(y_d - y_c)^2)](1 + p_3 r^2 + p_4 r^4 \dots) \end{aligned} \tag{3.6}$$

$$r = \sqrt{(x_d - x_c)^2 + (y_d - y_c)^2} \tag{3.7}$$

- x_u and y_u is the undistorted pixel coordinate of the point.
- x_d and y_d is the distorted pixel coordinate of the point.
- x_c and y_c is the centre of the distortion.
- K_n is the *n*th radial distortion coefficients.
- p_n is the coefficient of the *n*th tangential distortion.

The model contains two power series: first power series $(x_d - x_c)(K_1 r^2 + K_2 r^4 + \dots)$ is the radial distortion, and second power series $[p_1(r^2 + 2(x_d - x_c)^2) + 2p_2(x_d - x_c)(y_d - y_c)](1 + p_3 r^2 + p_4 r^4 \dots)$ is the tangential distortion.

3.5.1.3 Zhang's Calibration Technique

Zhang's calibration method can compute both distortion variables, extrinsic variable for the setup by utilizing checker board inputs (Figure 3.18, 3.19).

Zhang's calibration method uses the following procedure[178]:

- A special checkerboard was created and used for both camera and IR camera calibration. Details on the checkerboard can be found in the next section.
- Capture several images of the model plane using different angles by either moving the camera or the plane.
- Detect the feature points in the images. This step can be done manually or using feature extraction techniques. In this case, Canny edge extraction method was used.

- Using the closed-form solution as described, determine both five intrinsic parameters and extrinsic parameters.
- Calculate the coefficients of the radial distortion by solving the linear least-squares.
- Minimizing all the parameters.

To estimate the distortion, Zhang proposed a simple differential estimation: Zhang assumes only the first two distortion factors K_1 and K_2 are having major impact to the image, and can ignore the rest of K terms. By expanding the equation 3.5 and 3.6 with equation 3.4 to get an approximation relation equation 3.8 and 3.9.

$$\hat{u} = u + (u - u_0)[K_1(x^2 + y^2) + K_2(x^2 + y^2)^2] \tag{3.8}$$

$$\hat{v} = v + (v - v_0)[K_1(x^2 + y^2) + K_2(x^2 + y^2)^2] \tag{3.9}$$

Then the equation can be rewritten into linear form (equation 3.10) and finally solve the linear system to get **K** distortion coefficients.

$$\begin{bmatrix} (u-u_0)(x^2+y^2) & (u-u_0)(x^2+y^2)^2 \\ (v-v_0)(x^2+y^2) & (v-v_0)(x^2+y^2)^2 \end{bmatrix} \begin{bmatrix} K_1 \\ K_2 \end{bmatrix} = \begin{bmatrix} \hat{u}-u \\ \hat{v}-v \end{bmatrix} \tag{3.10}$$

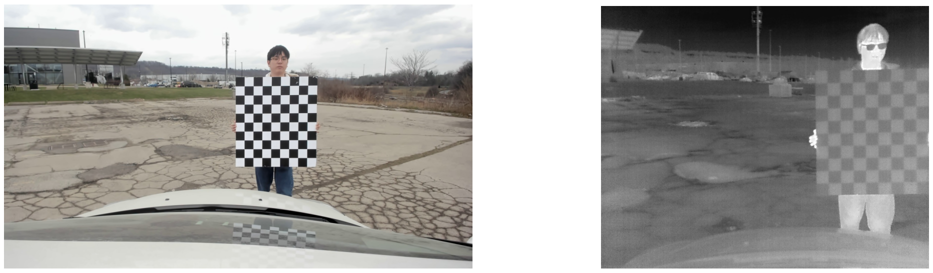


Figure 3.18: A calibration example with the checkerboard close to the centre of the sensor, image to the left is the RGB camera, and image to the right is the IR camera.

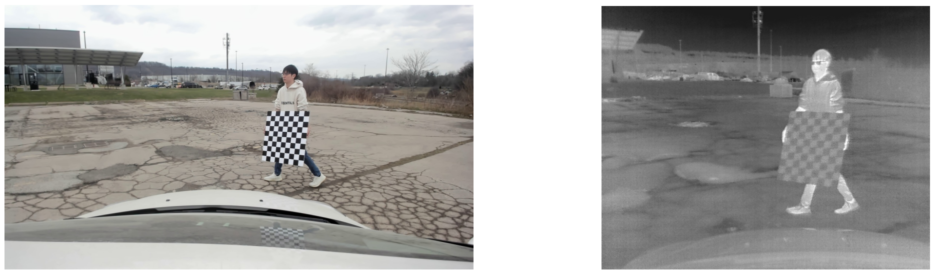


Figure 3.19: Another calibration example with an off-center checkerboard position. It's important to note that LiDAR sensor data is also collected during this time, resulting in the utilization of the same set of image data for computing both intrinsic and extrinsic parameters, as well as distortion.

3.5.2 Camera and LiDAR Spatial Coordinates Estimation

Aligning the spatial coordinates of LiDAR and camera sensors involves capturing data and images of the same objects and surroundings with both sensors, detecting features in the images, and correlating them to calculate intrinsic and extrinsic parameters. Current research on automatic spatial alignment of LiDAR and camera is mainly focused on identifying the most appropriate objects for feature matching and developing the optimal feature extraction algorithm.

Early works[12][143] work on feature matching on common daily objects, then using various minimizing techniques to find the extrinsic parameters. Qilong Zhang *et al.*[178] used checkerboards as calibration objects to estimate the camera's extrinsic parameters in relation to the ranger finder sensor. After capturing the sensor data, they used the checkerboard to identify feature points and performed Levenberg-Marquardt method to minimize the error function and finally estimate the extrinsic parameters. Notably, the calibration process is very close reassembles to popular LiDAR to camera calibration methods today[180].

3.5.2.1 Overview

At CMHT, a customized automated calibration pipeline was employed for the equipment used to determine the extrinsic and transform parameters for aligning the vehicle LiDAR and other sensors. In the preparation process, calibration data sets were captured by placing the checkerboard at various locations.

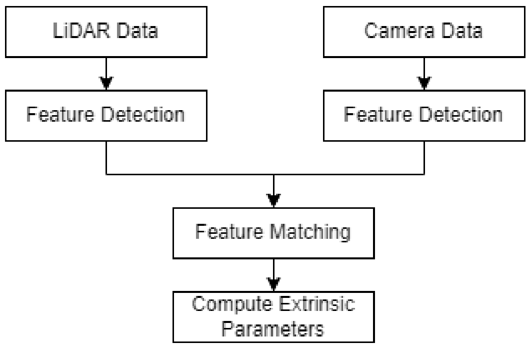


Figure 3.20: Overview of camera and LiDAR spatial alignment pipeline.

3.5.2.2 Feature Detection

In this work, a checkerboard is employed for calibrating both the LiDAR and the camera. The LiDAR is responsible for detecting the surface plane of the checkerboard and identifying its corners. On the other hand, the camera detects the corners of the checkerboard pattern as features. Through matching these detected features in the camera and LiDAR, the extrinsic parameters can be computed to achieve spatial alignment of the two sensors. This approach guarantees precise calibration and alignment of the two sensors, a critical requirement for dependable data fusion and perception tasks. The utilization of the checkerboard simplifies the calibration process and enhances the accuracy and robustness of the entire system.

LiDAR Featured Extraction The process (Figure 3.21) of detecting a featured calibration plane involves several steps. Firstly, LiDAR data is captured and clustered using Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm[50] . Once the clusters have been identified, each one is subjected to filtering to remove any noise or outliers, which is done using Random Sample Consensus (RANSAC) algorithm[52]. The resulting clusters are then checked to determine if their surface area is close to the computed checkerboard surface area within certain tolerances. If a cluster passes this test, it is deemed to be the correct calibration plane and is output as such. By following these steps, it is possible to accurately identify and calibrate a featured plane using LiDAR data.

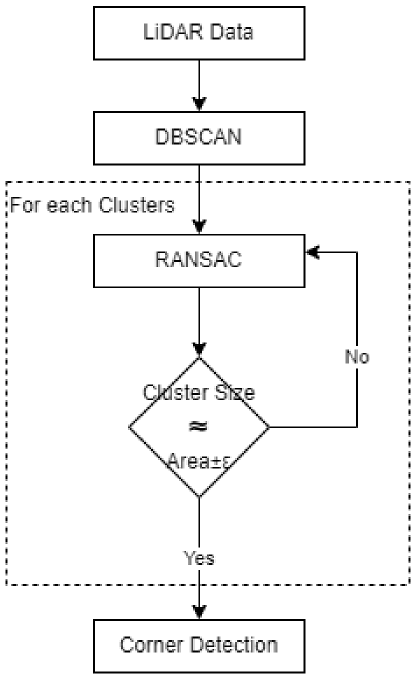


Figure 3.21: LiDAR feature detection pipeline.

DBSCAN To account for the possibility of background objects being mistakenly identified as checkerboard planes during the calibration process, the research utilized the DBSCAN algorithm (Algorithm 1) to determine the number of clusters present in the scene. This clustering algorithm does not require prior knowledge of the number of clusters and thus allows for more accurate identification of relevant objects. Following this process, the algorithm produced a set of N $\{x,y,z\}$ coordinates, where N represents the number of objects detected.

RANSAC Employing a modified RANSAC algorithm, the process outlined in Algorithm 2 is utilized for removing outliers from a given cluster. Assumption is made that there are no duplicate LiDAR points within the cluster. The approach involves random selection of three points to establish a plane by calculating the normal vector, denoted as $\vec{\mathbf{N}} =< A,B,C >$. Subsequent step involves computation of the distance offset, d_k, from the input LiDAR point $< x_k,y_k,z_k >$ for all other points, utilizing Equation 3.11. Points for which d_k exceeds the designated distance threshold, Th_{distance}, are filtered out. Lastly, the error function expressed in Equation 3.12 is employed to determine the optimality of the current plane set.

$$Ax_k + By_k + Cz_k + d_k = 0 \tag{3.11}$$

Algorithm 1 DBSCAN

```
Require: points list $P$, minimum distance $\varepsilon$, minimum number of LiDAR points $\tau$
  ID = 0
  for $i = 1, 2, ..., P$.length do
    if $P[i]$.label $\neq$ undefine then
      Find list of neighbour points $N$ in $P$ with $\varepsilon$
    end if
    if $N$.length $< \tau$ then
      $P[i]$.label = Noise
      continue
    end if
    ID = ID +1
    $P[i]$.label = ID
    S = N
    for $j = 1, 2, ..., S$.length do
      if $S[j]$.label = Noise then
        $S[j]$.label = ID
      end if
      if $S[j]$.label $\neq$ undefined then
        continue
      end if
      $S[j]$.label = ID
      Find list of neighbour points $N$ in $S$ with $\varepsilon$
      if $N$.length $\geq \tau$ then
        $S$.add($N$)
      end if
    end for
  end for
```

Corner	Condition 1	Condition 2
Bottom Left	Min(x)	Min(z)
Upper Left	Max(z)	Min(x)
Upper Right	Max(x)	Max(z)
Bottom Right	Min(z)	Max(z)

Table 3.1: Corner matching table

$$\varepsilon = \frac{\sum |d_k|}{n}, \text{ where } k = 1, 2, ..., n \tag{3.12}$$

Vertex Detection Once the points belonging to the checkerboard plane are identified, the process can proceed to locate the vertices (corners) of the checkerboard in the LiDAR scans.

One significant assumption is that the checkerboard always faces towards the observer and is closely aligned with the LiDAR's x and z planes. To detect the vertices, the following strategy is employed:

- To locate the bottom-left corner of the board, the plane point $< x, y, z >$ with the minimum x value is sought. If there are multiple points within a threshold value tr with close ranges, the one with the minimum z value is selected.
- For locating the remaining corners, please refer to Table 3.1. Despite the previous filtering step, certain scan points may still lie on the plane of the checkerboard but not necessarily belong to the checkerboard scan. In the subsequent step, these points must be eliminated.

Algorithm 2 RANSAC Outlier Filtering

```
Require: points list $P$, distance threshold $Th_{\text{distance}}$, iteration threshold $k$,
  minimum number of inlier $n$, point list Inlier, list $D$, point list $\text{Inlier}_{\text{best}}$,
  best error $\varepsilon_{\text{best}}$
  while iteration $< k$ do
    Randomly pick 3 points from $P$ to form a 3-tuple $T$
    if points in $T$ is collinear then
      continue
    end if
    Compute the norm $\vec{\mathbf{N}} = < A, B, C >$ from $T$
    for all point $p$ in $P$ do
      Substitute $p = < x, y, z >$ with $\vec{\mathbf{N}}$ into Equation 3.11 to compute $d$
      if $d \leq Th_{\text{distance}}$ then
        Inlier.add($p$)
      end if
    end for
    if Number of points in Inlier $\geq n$ then
      for all points $m$ in Inlier do
        Substitute $m = < x, y, z >$ with $\vec{\mathbf{N}}$ into Equation 3.11 to compute $d$
        $D$.add($d$)
      end for
      Substitute $D$ into Equation 3.12 to compute $\varepsilon$
      if $\varepsilon_{\text{best}} \geq \varepsilon$ then
        $\varepsilon_{\text{best}} = \varepsilon$
        $\text{Inlier}_{\text{best}} = \text{Inlier}$
      end if
    end if
    iteration = iteration + 1
  end while
```

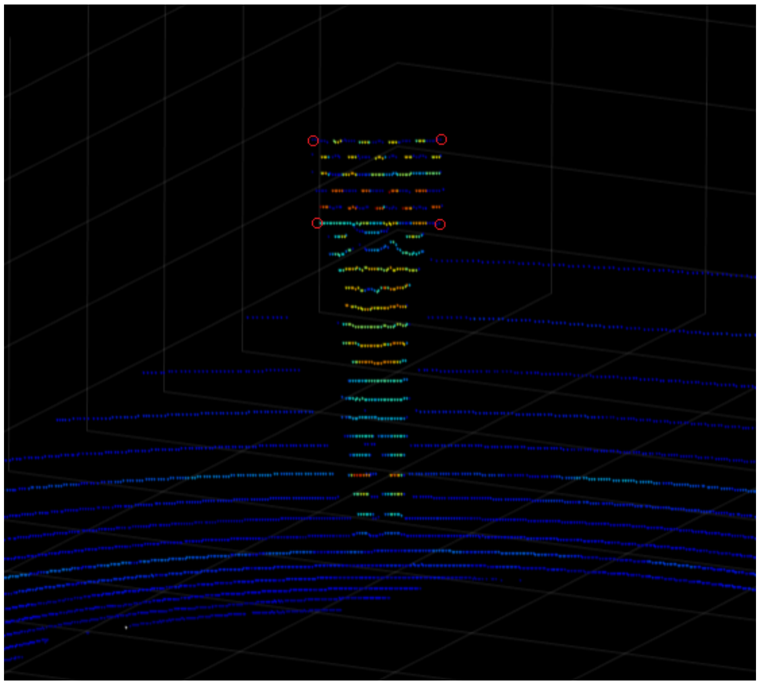


Figure 3.22: Checkerboard vertices marked by red circles

Given equation the line in 3D space is:

$$[x,y,z] = [x_0,y_0,z_0] + t[a,b,c] \tag{3.13}$$

Which represents all the points on the line passing through the point $[x_0,y_0,z_0]$ and in the direction of the slope vector $\vec{\mathbf{r}} = [a,b,c]$. t is an arbitrary real value.

With knowledge of the approximate starting vertex's location, it becomes possible to define four lines representing the four sides of the LiDAR-scanned checkerboard plane. For the four corners C_1,C_2,C_3,C_4, the four sides are constructed by substituting values into Equation 3.13, resulting in equations for the four sides $\overline{\mathbf{S}12},\overline{\mathbf{S}23},\overline{\mathbf{S}34},\overline{\mathbf{S}41}$.

$$\overline{\mathbf{S}_{12}} = \begin{bmatrix} C_{1x}+(C_{2x}-C_{1x})t \\ C_{1y}+(C_{2y}-C_{1y})t \\ C_{1z}+(C_{2z}-C_{1z})t \end{bmatrix}, \overline{\mathbf{S}_{23}} = \begin{bmatrix} C_{2x}+(C_{3x}-C_{2x})t \\ C_{2y}+(C_{3y}-C_{2y})t \\ C_{2z}+(C_{3z}-C_{2z})t \end{bmatrix}$$

$$\overline{\mathbf{S}_{34}} = \begin{bmatrix} C_{3x}+(C_{4x}-C_{3x})t \\ C_{3y}+(C_{4y}-C_{3y})t \\ C_{3z}+(C_{4z}-C_{3z})t \end{bmatrix}, \overline{\mathbf{S}_{41}} = \begin{bmatrix} C_{4x}+(C_{1x}-C_{4x})t \\ C_{4y}+(C_{1y}-C_{4y})t \\ C_{4z}+(C_{1z}-C_{4z})t \end{bmatrix}$$

For a given point $p_k =< x_k, y_k, z_k >$, the shortest distance $d_{k,ij}$ from the point p_k to the side $\overline{\mathbf{S}_{ij}}$ can be computed using Equation 3.14.

$$d^2_{k,ij} = [(C_{ix} - x_k) + (C_{jx} - C_{ix})t]^2 + [(C_{iy} - y_k) + (C_{jy} - C_{iy})t]^2 + [(C_{iz} - x_z) + (C_{jz} - C_{iz})t]^2 \tag{3.14}$$

where t can be computed using the equation 3.15.

$$t = -\frac{(C_{ix} - x_k) \cdot (C_{jx} - C_{ix})}{|C_{jx} - C_{ix}|^2} \tag{3.15}$$

Substituting equation 3.15 back into equation 3.14, the final equation 3.16 is obtained. The distance $d_{k,ij}$ is kept in squared form because this number will be used to determine if the corners $\{C_1, C_2, C_3, C_4\}$ actually align with the checkerboard.

$$d^2_{k,ij} = \frac{|(C_{jx} - C_{ix}) \times (C_{ix} - x_k)|^2}{|C_{jx} - C_{ix}|^2} \tag{3.16}$$

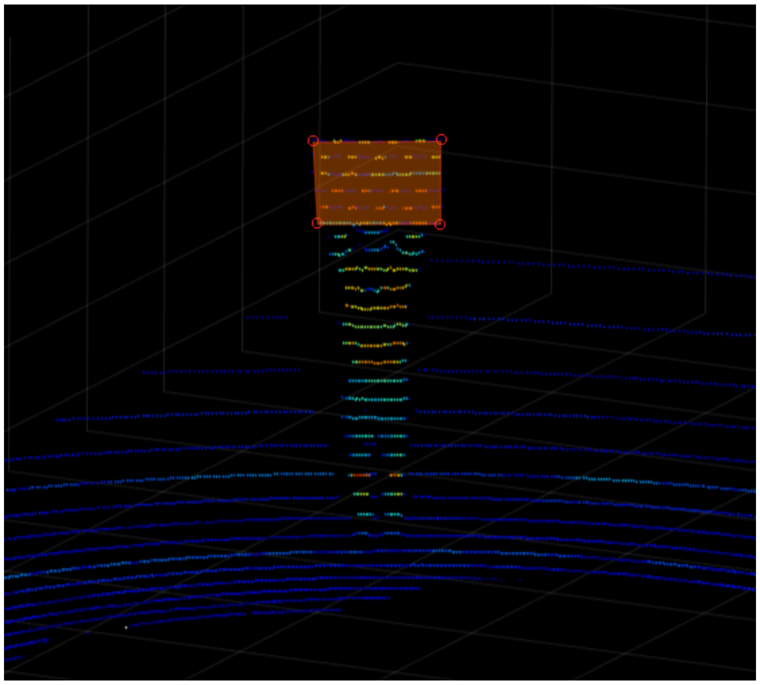


Figure 3.23: Plane formed by enclose vertices

To determine the scan points that enclose the checkerboard shape, the Quickhull algorithm[13] is employed to construct a convex hull of the checkerboard. To simplify the process, only the x and z coordinates of the LiDAR data are considered, assuming that the depth y has a negligible impact on the result and avoiding the need to deal with hyperplanes.

$$ax + by + c = 0 \tag{3.17}$$

Given a standard form of linear equation (Equation 3.17), and two points C_i, C_j. Coefficient a, b and c can be computed with the following equation.

$$\begin{aligned} a &= (C_{iy} - C_{jy}) \\ b &= (C_{jx} - C_{ix}) \\ c &= (C_{ix}C_{jy} - C_{jx}C_{iy}) \end{aligned}$$

$$(C_{iy} - C_{jy})x + (C_{jx} - C_{ix})y + (C_{ix}C_{jy} - C_{jx}C_{iy}) = 0 \tag{3.18}$$

To determine the half-plane in which a point $t =< x, z >$ lies with respect to the line $l_{C_iC_j}$, Equation 3.18 can be utilized. If the equation equals 0, then t is collinear with the points C_i and C_j. Otherwise, t resides in one of the two half-planes. To ascertain which half-plane t is positioned on, it is examined whether the left side of the equation is greater or less than 0. If it is greater than 0, then t lies on the half-plane to the left of the line; if it is less than 0, then t lies on the half-plane to the right of the line. Importantly, it should be noted that in this computation, the y value has been swapped with the z value, effectively treating all 3D points $< x, y, z >$ as 2D points $< x, z >$.

Barycentric coordinates were employed to establish whether a point $t =< x, y >$ lies within a triangle constituted by the points C_i, C_j, C_k. If λ_1, λ_2, and λ_3 are all positive, then the point resides inside the triangle; conversely, if any of them are not positive, the point lies outside the triangle.

$$\lambda_1 = \frac{((C_{jy} - C_{ky}) * (x - C_{kx}) + (C_{kx} - C_{jx}) * (y - C_{ky}))}{((C_{jy} - C_{ky}) * (C_{ix} - C_{kx}) + (C_{kx} - C_{jx}) * (C_{iy} - C_{ky}))} \tag{3.19}$$

$$\lambda_2 = \frac{(C_{ky} - C_{iy}) * (x - C_{kx}) + (C_{ix} - C_{kx}) * (y - C_{ky}))}{((C_{jy} - C_{ky}) * (C_{ix} - C_{kx}) + (C_{kx} - C_{jx}) * (C_{iy} - C_{ky}))} \tag{3.20}$$

$$\lambda_3 = 1 - \lambda_1 - \lambda_2 \tag{3.21}$$

With the foundational calculations in place, the application of the Quickhull algorithm to the LiDAR scan points can be undertaken, following the steps delineated in Algorithm 3.

After the convex hull points have been extracted, the previously identified corners can be validated. In particular, Equation 3.16 is employed to calculate the distance of each point from its corresponding line segment. This is followed by the computation of the error using Equation 3.22.

Algorithm 3 Quick Convex Hull

```
Require: line start point $C_s$, line end point $C_d$, point list $P$
Require: left points list $U$, right points list $D$, hull points list $H = \{C_s, C_d\}$
  for all points $t$ in $P$ do
    if $t$ is on the left side of line $l_{C_s,C_d}$ then
      $U$.add($t$)
    else
      $D$.add($t$)
    end if
  end for
  Findhull ($U$,$C_s$,$C_d$)
  Findhull ($D$,$C_d$,$C_s$)
  Return $H$
  Function Findhull(point list $p_l$, starting point $l_s$, end point $l_e$)
  if $p_l = \emptyset$ then
    Return
  end if
  Find the furthest point $f$ using equation 3.14
  $H$.add($f$)
  Remove all the points lies inside the triangle $< l_s, l_e, f >$ from $p_l$
  In $p_l$, put points from left side of the line $l_{l_s,f}$ into set $s_1$
  In $p_l$, put points from right side of the line $l_{l_e,f}$ into set $s_2$
  Findhull ($s_1$,$l_s$,$f$)
  Findhull ($s_2$,$f$,$l_d$)
  End
```

Determining the corresponding convex hull points for a given side is a relatively straightforward task. Specifically, a search is conducted to identify points lying within the value range of each corner, guided by the condition $\overline{\mathbf{S}ij} = ((p_k|s_{ix} \leq p_{kx} \leq s_{jx}) \wedge (s_{iy} \leq p_{ky} \leq s_{jy}))$. Subsequently, a fitting test is performed to assess the capacity of the initially selected corner points to encompass all the points identified along the side. This test involves calculating the distance of point k from line S_{ij} using Equation 3.16, and then aggregating the error using Equation 3.22. If the resulting error ε surpasses the threshold value th_{side}, the set is considered inadequate, prompting a restart of the process.

$$\varepsilon = \sum d_{k,ij}^2 \tag{3.22}$$

Finally, leveraging the physical dimensions of the checkerboard enables us to promptly assess the suitability of the detected plane. This is achieved by computing the percentage error, as described in Equation 3.23, between the actual side length $\overline{\mathbf{S}_k}$real and the estimated side length $\overline{\mathbf{S}_k}$, following the approach outlined in the work by Yoonsu Park *et al.*[131]. Subsequently, the computed error ε_k is compared against a chosen threshold value T_k. If the error is below the threshold, it is inferred that the set of LiDAR scan points corresponds to the checkerboard intended for calibration. Conversely, if the error surpasses the threshold, the set is discarded, initiating a new search for an alternative set of image and LiDAR data for calibration purposes.

$$\varepsilon_k = \frac{|\overline{\mathbf{S}_k}|_{\text{real}} - |\overline{\mathbf{S}_k}|}{|\overline{\mathbf{S}_k}|_{\text{real}}} \tag{3.23}$$

Camera Feature Extraction Identifying the black and white corners on a checkerboard is a pivotal task in camera calibration, given their utilization in both Zhang's calibration, as explained in the earlier section, and the extrinsic estimation process. In this study, the Harris corner detection method [65] has been adopted for this purpose. The camera's input image is first transformed into grayscale, and subsequently, the Harris filter is employed on the processed image to discern the corners of significance.

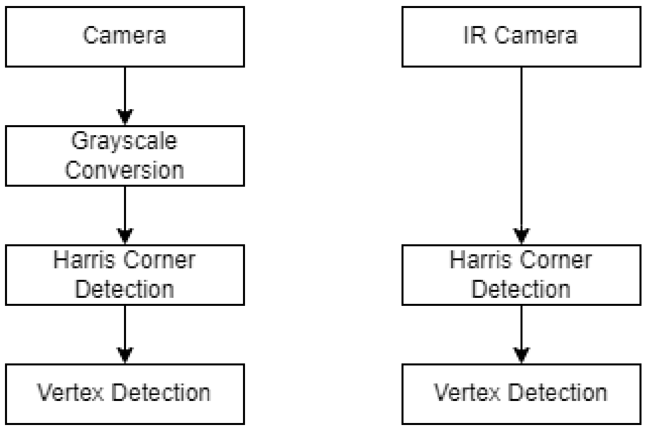


Figure 3.24: Camera feature detection pipeline.

Gray Scale Conversion In this research, weighted grayscale conversion (as specified in Equation 3.24) was employed to transform the RGB camera from its native RGB colour space $< R, G, B >$ to grayscale colour space, thereby reducing the number of channels from 3 to 1. It should be noted that this conversion process was deemed superfluous for the IR camera, as the latter was already generating an image that resembled grayscale.

$$\text{gray} = 0.2989 \times R + 0.5870 \times G + 0.1140 \times B \tag{3.24}$$

Harris Corner Detection After obtaining the filtered grayscale image, the next step involves determining its partial derivatives with respect to the horizontal and vertical axes, denoted as $I_x(x,y)$ and $I_y(x,y)$, respectively. For this purpose, the Sobel operator is employed, a widely recognized technique in digital image processing for edge detection. By applying the Sobel operator to the filtered grayscale image, accurate computation of the partial derivatives in both directions is achieved. Importantly, it should be noted that the Gaussian smoothing inherent in the Sobel operator obviates the need for extra smoothing using a Gaussian filter to suppress image noise.

In the research, a 3×3 Sobel vertical filter (Equation 3.25) and a 3×3 Sobel horizontal filter (Equation 3.26) are employed to convolve with the image $\mathbf{I}$ (Equation 3.27 and Equation 3.28). The outcome of this operation is the creation of a vertical partial derivative matrix I_y and a horizontal partial derivative matrix I_x.

$$V = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{3.25}$$

$$H = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{3.26}$$

$$I_x = \mathbf{I} * H \tag{3.27}$$

$$I_y = \mathbf{I} * V \tag{3.28}$$

In the subsequent stage, Harris corner detection method was employed by using Equation 3.29. In this equation, the matrix M represents the structure tensor, while $w(x,y)$ is the window function of size $a \times b$, with a rectangular shape selected for this study (as indicated in Equation 3.31).

$$E(u,v) \approx [u,v]M \begin{bmatrix} u \\ v \end{bmatrix} \tag{3.29}$$

$$M = \sum_{(x,y)\in w} w(x,y) \begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix} \tag{3.30}$$

$$w(x,y) = \begin{cases} 1, & \text{if } |x| < \frac{a}{2} \text{ and } |y| < \frac{b}{2} \\ 0, & \text{otherwise} \end{cases} \tag{3.31}$$

The Harris corner detection method utilizes the response function R, as defined in Equation 3.32, to compute the score of the window. The free parameter k is empirically determined to lie within the range of [0.04, 0.06] for the Harris detector.

$$R = \det(M) - k\operatorname{trace}(M)^2 \tag{3.32}$$

$$\det(M) = \lambda_1 \lambda_2 \tag{3.33}$$

$$\operatorname{trace}(M) = \lambda_1 + \lambda_2 \tag{3.34}$$

The response function is simplified by substituting Equation 3.33 and Equation 3.34 into Equation 3.32. The resulting expression is shown in Equation 3.35.

$$R = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2 \tag{3.35}$$

Subsequently, the corner pixels are determined by comparing the response value R with a predetermined threshold value T_{corner}, considering the point to be a corner if the response value exceeds this threshold. Furthermore, the type of region to which the point belongs can be identified based on the following conditions:

- if $|R|$ is small, λ_1 and λ_2 are small, then the region is flat.
- if $R < 0$, $\lambda_1 >> \lambda_2$ or $\lambda_2 >> \lambda_1$, then the region is an edge.
- if R is large, $\lambda_1 \approx \lambda_2$ and both large, then the region is a corner.

The above conditions are summarized in Figure 3.25.

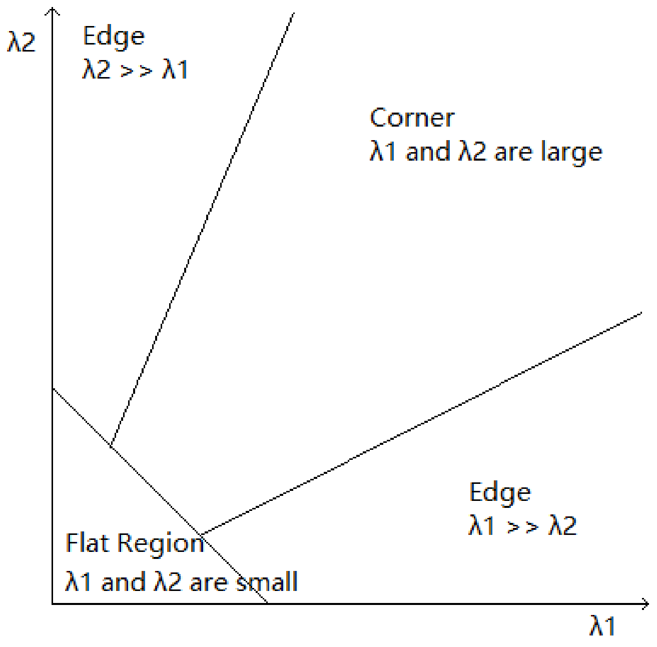


Figure 3.25: Harris corner detection

A list of corners has been obtained using the aforementioned method, which can be utilized for both Zhang's camera calibration for intrinsic calibration, and for the subsequent step of estimating the vertex of the checkerboard. This process involves feature matching with LiDAR data to extract extrinsic parameters.

Vertex Detection A similar methodology, when applied to LiDAR data, can also be used for camera vertex detection. However, since the projection from the camera to real-world coordinates is currently unknown, the application is restricted to using DBSCAN, RANSAC, and a fitting process only. It's noteworthy that suitability tests based on real dimensions are not applicable to the camera vertex detection process.

3.5.2.3 IR Camera Extrinsic Parameter

The IR camera employs a feature detection method similar to that of a regular camera. However, the capture of a different light frequency requires certain precautions to be taken during the calibration process. Specifically, the IR camera is not sensitive to the visual light spectrum, necessitating that the surrounding temperature difference be relatively greater than that of the checkerboard during calibration. Therefore, during summertime calibration, the checkerboard is exposed to direct sunlight for approximately 30 minutes to allow it to heat up. Conversely, during wintertime calibration, a heater is used to warm the board (Figure 3.26). The checkerboard's square size is 81mm.

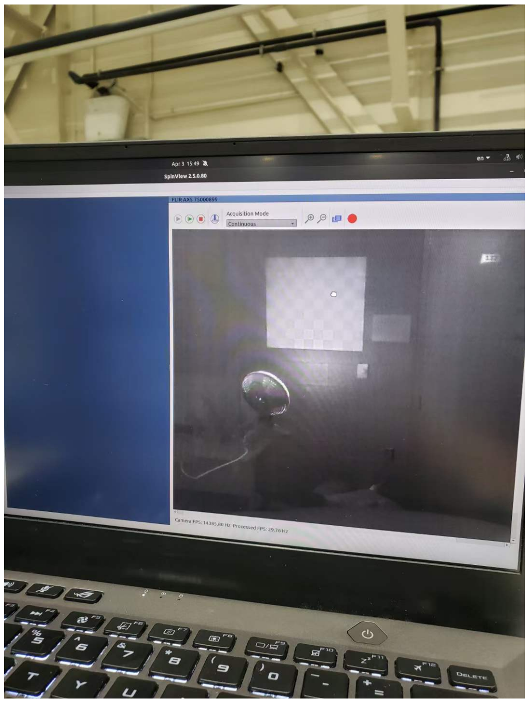


Figure 3.26: Baking the checkerboard sheet

The checkerboard pattern used in this study is composed of three layers (Figure 3.27). The first layer is a black pattern, followed by a white pattern, and the final layer is a hard wooden board. This design reduces surface bending and flexing when the checkerboard is moved around. The black and white pattern layers are mounted on the hard wooden board to minimize heat exchange and interference during the calibration process. This way the checkerboard is specially designed to absorb heat differently on black and white patterns, generating the checkerboard pattern that the IR camera captures. Even when both surfaces are uniformly heated, the white pattern always remains at a lower temperature than the black pattern, enabling us to proceed with the calibration process similar to that of a regular camera.

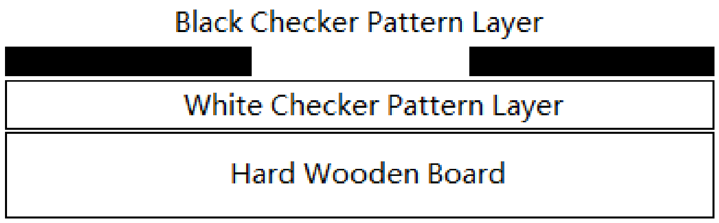


Figure 3.27: A cross section illustration of the checkerboard

3.5.2.4 Estimating the Extrinsic Parameter

After extracting features from both the camera and LiDAR, the first step to compute the extrinsic parameter is to match the corners obtained from the two sources. This is accomplished by spatially matching the co-ordinates based on the (u,v) image coordinates and (x,z) LiDAR coordinates. Given a set of image corner coordinates $(< u_1, v_1 >, < u_2, v_2 >, < u_3, v_3 >, < u_4, v_4 >)$ and LiDAR corner coordinates $(< x_1, y_1, z_1 >, < x_2, y_2, z_2 >, < x_3, y_3, z_3 >, < x_4, y_4, z_4 >)$, the pairs are matched as follows:

- MIN(x) matches MIN(u)
- MIN(z) matches MIN(v)
- MAX(x) matches MAX(u)
- MAX(z) matches MAX(v)

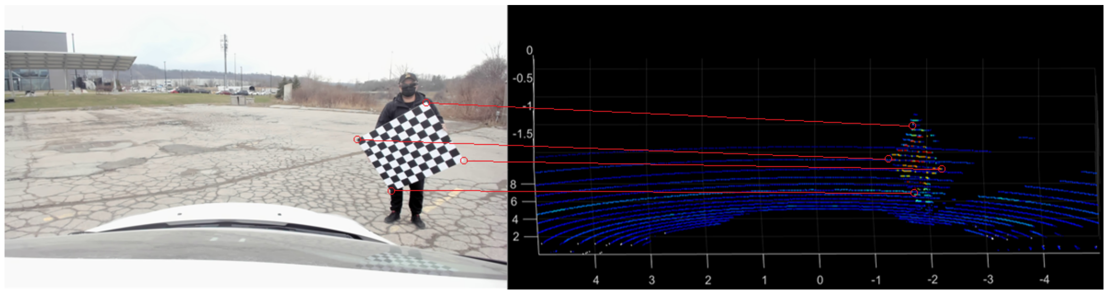


Figure 3.28: Matching corner features

The result of is a calibration dataset **S** contains n samples of calibration data $s_i = < x_i, y_i, z_i, u_i, v_i >, i = 1,2,3,4...n$.

Projection Matrix Rewrite the equation 3.1, projection matrix **M** (Equation 3.37) was obtained.

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} f_u & 0 & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = M \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{3.36}$$

$$M = \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{pmatrix} \tag{3.37}$$

By rearranging and multiplying the equation, two distinct equations (Equation 3.38 and Equation 3.39) can be derived to calculate the dot product of u and v. Using these equations in conjunction with a sufficient number of input pairs, the problem can be effectively transformed into a non-linear least squares problem. This transformation facilitates the estimation of projection matrix parameters.

$$u = \frac{m_{11}x + m_{12}y + m_{13}z + m_{14}}{m_{31}x + m_{32}y + m_{33}z + m_{34}} \tag{3.38}$$

$$v = \frac{m_{21}x + m_{22}y + m_{23}z + m_{24}}{m_{31}x + m_{32}y + m_{33}z + m_{34}} \tag{3.39}$$

A variety of algorithms are available for resolving the non-linear equations under consideration. In this study, the Levenberg–Marquardt algorithm (LM) technique was chosen over other frequently employed optimizers such as Newton's method. This selection is motivated by the fact that the initial estimation approximation tends to be relatively inaccurate on the first attempt. Moreover, the calibration dataset possessed is relatively limited, and the optimization procedure is conducted only once, with performance not considered the foremost priority.

Levenberg-Marquardt Algorithm The multivariable function $f(s_i, \mathbf{M})$ (Equation 3.40) can be derived by transforming Equation 3.38 and Equation 3.39 into matrix form. Here, $s_i \in \mathbf{S}$ represents the calibration sample, and **M** represents the projection matrix. This transformation is based on the work of Park *et al.*[131]. By vertically stacking the matrices in Equation 3.40, a non-linear system can be constructed between the dataset $\mathbf{S} = \{s_i | i = 1, 2, 3, 4...n\}$ and the function $f(s_i, \mathbf{M})$.

$$\begin{bmatrix} f_1(s_i, \mathbf{M}) \\ f_2(s_i, \mathbf{M}) \end{bmatrix} = \begin{bmatrix} x_i & y_i & z_i & 1 & 0 & 0 & 0 & 0 & -u_i x_i & -u_i y_i & -u_i z_i & -u_i \\ 0 & 0 & 0 & 0 & x_i & y_i & z_i & 1 & -v_i x_i & -v_i y_i & -v_i z_i & -v_i \end{bmatrix} \mathbf{M} \tag{3.40}$$

$$\mathbf{M}^T = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} & m_{21} & m_{22} & m_{23} & m_{24} & m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \tag{3.41}$$

In order to find an optimal projection matrix **M**, minimize the sum of squares by obtaining a set of values $\mathbf{M}_{\text{optimal}}$ is required. Ideally, if sum of squares of all $f(s_i, \mathbf{M})$ approximates a zero matrix (Equation 3.42), optimal projection matrix **M** was found.

$$\mathbf{M}_{\text{optimal}} \in \text{argmin}_{\mathbf{M}} \sum_{i=1}^{n} [f(s_i, \mathbf{M})]^2 \tag{3.42}$$

$$f(s_i, \mathbf{M}_{k+1} + \delta_k) \approx f(s_i, \mathbf{M}_k) + \mathbf{J}_k, \delta_k \tag{3.43}$$

The LM algorithm (Algorithm 4) can be applied to Equation 3.40 to estimate the projection matrix parameters $m_{11}, m_{12}, ...m_{34}$. LM approximates the linearization of the function $f(s_i, \mathbf{M})$ using Equation 3.43. To use the algorithm, several parameters must be predefined:

- An initial value M_0, which in this research was set to $\mathbf{M}_0 = 1, 1, 1..., 1$ using a uniform distribution to initialize **M**.
- A damping parameter l
- A max iteration it_{max}
- A stopping threshold ε

Once these parameters have been defined, the algorithm procedure shown in Figure 3.29 can be used to approximate the optimal projection matrix value **M**. The variables required for the algorithm are computed using Equation 3.44, 3.45, and 3.46, while Equation 3.47 is used to update the variables during the optimization process.

$$\mathbf{J}_k = \begin{bmatrix} \frac{\partial f_1(s_1,\mathbf{M}_k)}{\partial m_{11}} & \frac{\partial f_i(s_1,\mathbf{M}_k)}{\partial m_{12}} & \cdots & \frac{\partial f_1(s_1,\mathbf{M}_k)}{\partial m_{34}} \\ \frac{\partial f_2(s_1,\mathbf{M}_k)}{\partial m_{11}} & \frac{\partial f_2(s_1,\mathbf{M}_k)}{\partial m_{12}} & \cdots & \frac{\partial f_2(s_1,\mathbf{M}_k)}{\partial m_{34}} \\ \frac{\partial f_1(s_2,\mathbf{M}_k)}{\partial m_{11}} & \frac{\partial f_i(s_2,\mathbf{M}_k)}{\partial m_{12}} & \cdots & \frac{\partial f_1(s_2,\mathbf{M}_k)}{\partial m_{34}} \\ \frac{\partial f_2(s_2,\mathbf{M}_k)}{\partial m_{11}} & \frac{\partial f_2(s_2,\mathbf{M}_k)}{\partial m_{12}} & \cdots & \frac{\partial f_2(s_2,\mathbf{M}_k)}{\partial m_{34}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_1(s_n,\mathbf{M}_k)}{\partial m_{11}} & \frac{\partial f_i(s_n,\mathbf{M}_k)}{\partial m_{12}} & \cdots & \frac{\partial f_1(s_n,\mathbf{M}_k)}{\partial m_{34}} \\ \frac{\partial f_2(s_n,\mathbf{M}_k)}{\partial m_{11}} & \frac{\partial f_2(s_n,\mathbf{M}_k)}{\partial m_{12}} & \cdots & \frac{\partial f_2(s_n,\mathbf{M}_k)}{\partial m_{34}} \end{bmatrix} \tag{3.44}$$

$$(\mathbf{J}_k^T \mathbf{J}_k + \lambda_k \mathbf{I})\delta_k = \mathbf{J}_k^T [f(s_i, \mathbf{M_k})] \tag{3.45}$$

$$\sigma_k = \sum_{i=1}^{n} [f(s_i, \mathbf{M}_k)]^2 \tag{3.46}$$

$$\mathbf{M_{k+1}} = \mathbf{M_k} + \delta_k \tag{3.47}$$

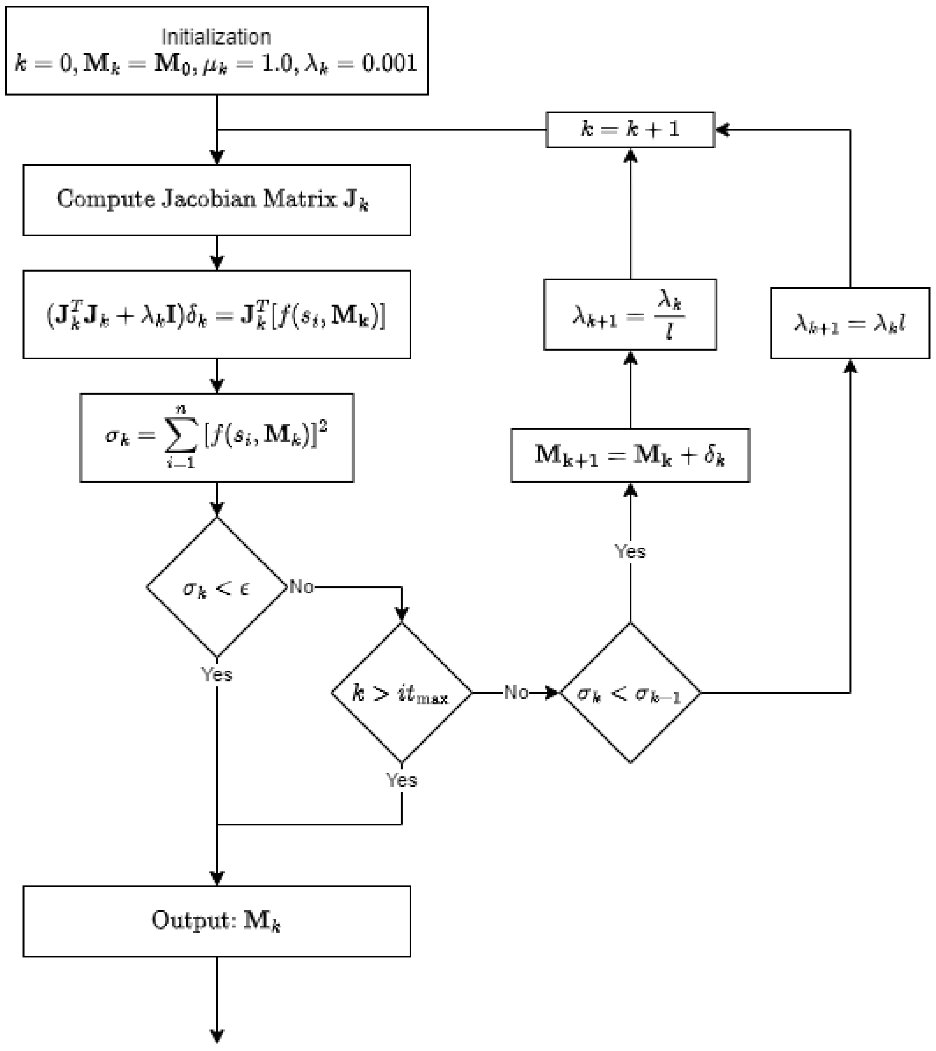

Initialization
$k = 0, \mathbf{M}_k = \mathbf{M}_0, \mu_k = 1.0, \lambda_k = 0.001$
Compute Jacobian Matrix $\mathbf{J}_k$
$(\mathbf{J}_k^T\mathbf{J}_k + \lambda_k\mathbf{I})\delta_k = \mathbf{J}_k^T[f(s_i, \mathbf{M}_k)]$
$\sigma_k = \sum_{i=1}^{n}[f(s_i, \mathbf{M}_k)]^2$
$\sigma_k < \epsilon$
No
Yes
$k > it_{\max}$
No
Yes
$\sigma_k < \sigma_{k-1}$
Yes
$\mathbf{M}_{k+1} = \mathbf{M}_k + \delta_k$
$\lambda_{k+1} = \frac{\lambda_k}{l}$
$\lambda_{k+1} = \lambda_k l$
$k = k + 1$
Output: $\mathbf{M}_k$

Figure 3.29: LM algorithm steps

3.5.2.5 Sensor Alignment

Once the projection matrim **M** is obtained, it becomes possible to project LiDAR data onto cameras, as demonstrated in Figure 3.30 and Figure 3.31.

Algorithm 4 Levenberg-Marquardt Algorithm

```
Require: Initial value $M_0$, damping parameter $l$, dataset $\mathbf{S}$
Require: max iteration $it_{\max}$, stopping threshold $\varepsilon$
  while $k \leq it_{\max}$ do
    Compute $\mathbf{J}_k$(Equation 3.44), $\delta_k$(Equation 3.45), and $\sigma_k$ (Equation 3.46)
    if $\sigma_k < \varepsilon$ then
      Return $\mathbf{M}_k$
    else
      if $\sigma_k < \sigma_{k-1}$ then
        $\mathbf{M_{k+1}} = \mathbf{M_k} + \delta_k$
        $\lambda_{k+1} = \frac{\lambda_k}{l}$
      else
        $\lambda_{k+1} = \lambda_k l$
      end if
    end if
    $k = k + 1$
  end while
```

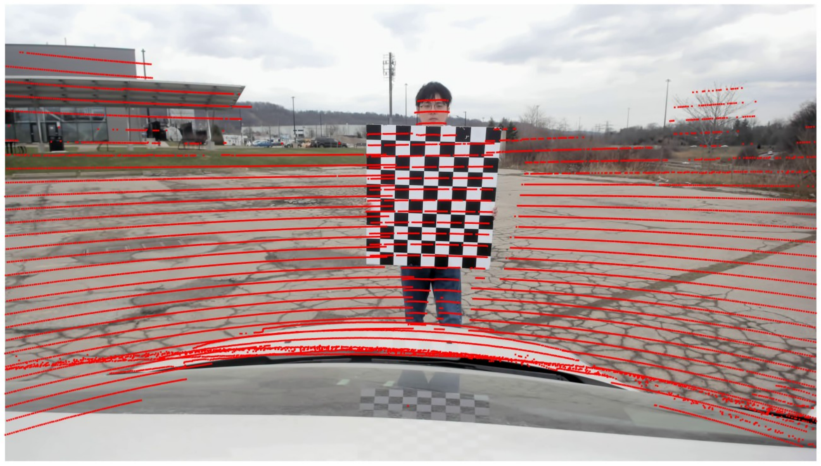


Figure 3.30: Camera overlay example

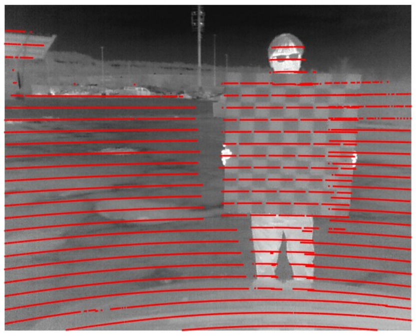


Figure 3.31: IR camera overlay example

3.5.3 Result

To evaluate the algorithm's performance, 40 calibration samples (Figure 3.32) were hand-selected from a pool of over 1500 captured frames. These samples were collected at different ranges and angles, approximately 3m, 5m, and 7m away from the vehicle, and were manually labeled as ground truth measurements. Due to the limitations posed by the field of view (FOV) and the presence of a vehicle mounting rack, the part of the vehicle engine in the camera's FOV had to be cropped out. Consequently, the original camera image dimensions were adjusted to 1280×500, while the IR camera image had effective dimensions 640×425 (Figure 3.33). For better results and faster processing times, the self-vehicle cluster ($2.5m \times 5m \times 2.5m$) and its surroundings were trimmed to generate only the captured ROI for LiDAR inputs.

To evaluate the algorithm's performance, 40 calibration samples (Figure 3.32) were hand-selected from a pool of over 1500 captured frames. These samples were collected at different ranges and angles, approximately 3m, 5m, and 7m away from the vehicle, and were manually labeled as ground truth measurements. Due to the limitations posed by the field of view (FOV) and the presence of a vehicle mounting rack, the vehicle engine part from the camera's FOV had to be cropped out. Consequently, the original camera image dimensions were adjusted to 1280×500, while the IR camera image had effective dimensions 640×425 (Figure 3.33). For better results and faster processing times, the self-vehicle cluster ($2.5m \times 5m \times 2.5m$) and its surroundings were trimmed to generate only the captured Region of Interest (ROI) for LiDAR inputs.

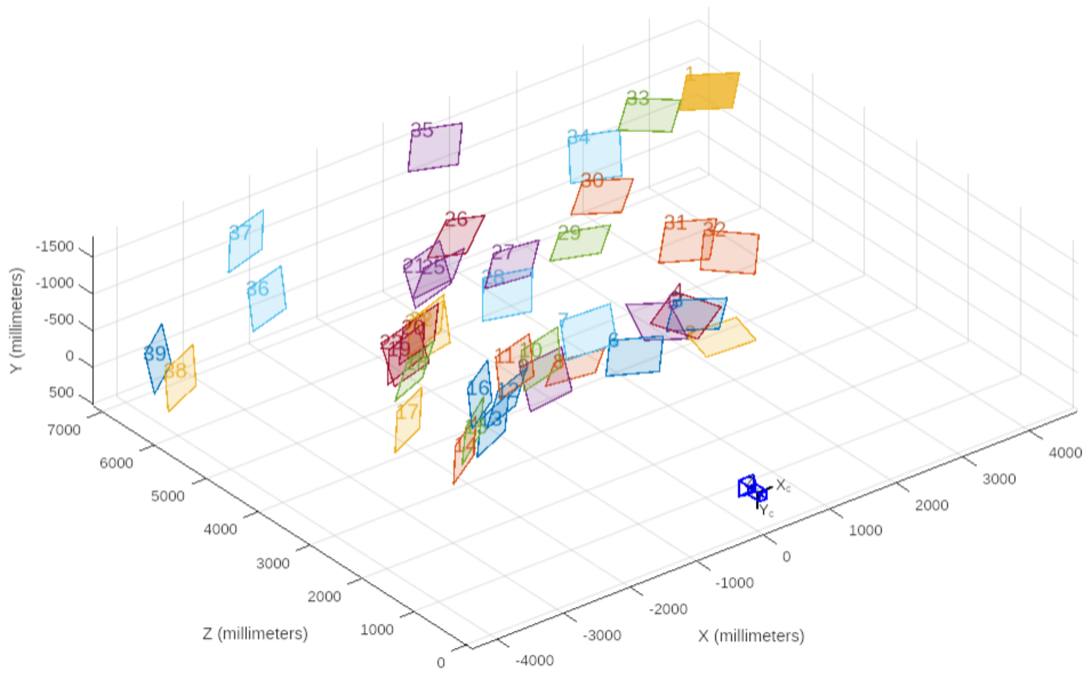
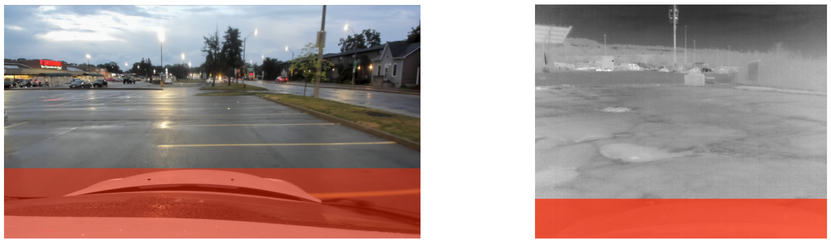


Figure 3.32: Calibration Samples



(a) Camera (b) IR Camera

Figure 3.33: Camera deadzone in the calibration area, marked in red. Left: camera, Right: IR camera.

The error ε of each sample is computed by taking the average pixel distance of each vertex (Equation 3.48).

$$\varepsilon = \frac{1}{4}\sum_{i=1}^{4}\sqrt{(U_i - u_i)^2 + (V_i - v_i)^2} \tag{3.48}$$

where $< u, v >$ represents the projected vertex, and $< U, V >$ represents the ground truth.

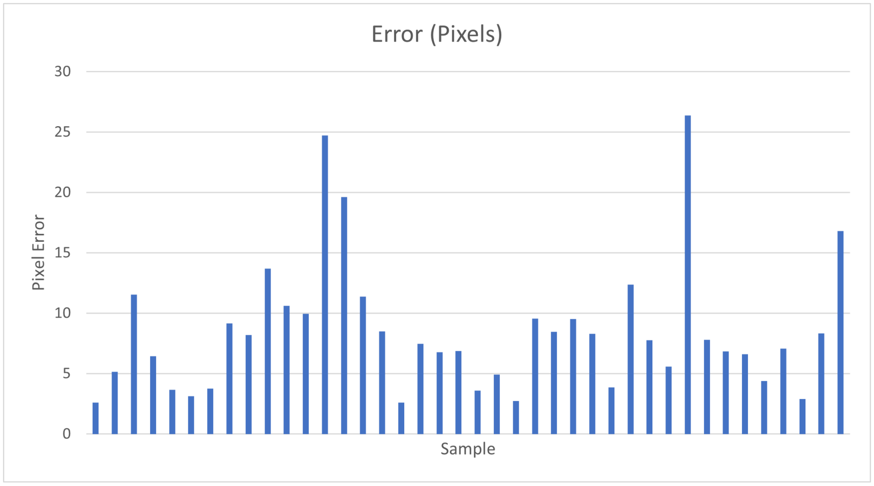


Figure 3.34: Pixel errors

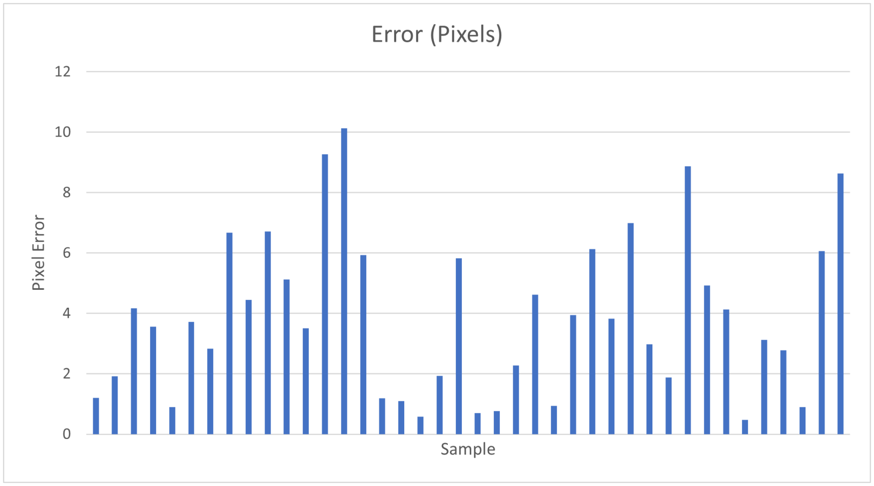


Figure 3.35: IR Camera pixel errors

Due to the IR camera having a much lower resolution (640×425) compared to the regular camera (1280×500), the camera calibration result is actually worse for the regular camera. The error for the regular camera is approximately 0.66% with respect to the horizontal pixel counts and 1.70% with respect to the vertical pixel counts, while the IR camera has errors of 0.61% horizontally and 0.91% vertically, respectively. This discrepancy is expected due to the IR camera's lower resolution, resulting in higher error tolerances. Figure 3.34 and figure 3.35 shows the error in pixels of each sample.

On the other hand, both the IR camera and the regular camera exhibit a trend where the pixel error increases as the object moves further away from the camera. This trend is likely caused by a slightly inaccurate approximation of the focal length f.

Additionally, both cameras display higher errors in the top-left corner, which may be attributed to their setup being off-center from the LiDAR and biased towards the right side. To improve this, it is advisable to explore the possibility of moving the camera closer to the LiDAR for better alignment and reduced errors in that region.

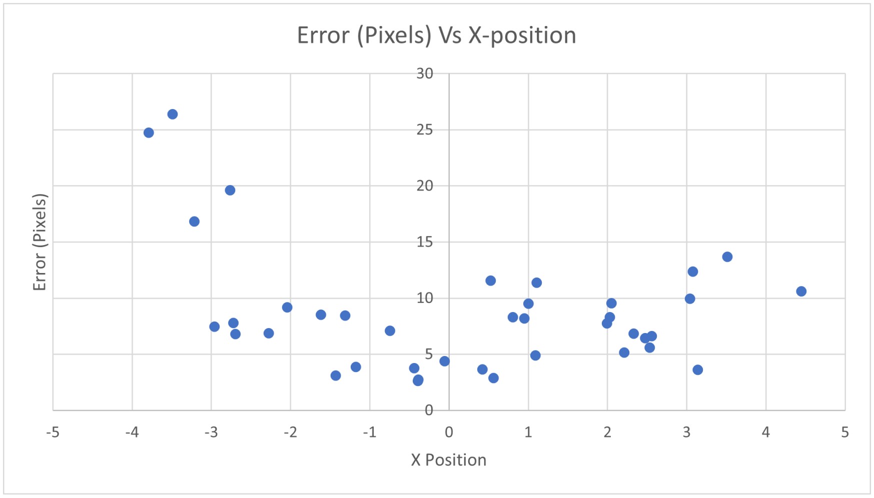


Figure 3.36: Camera pixel errors in relation to the X-position

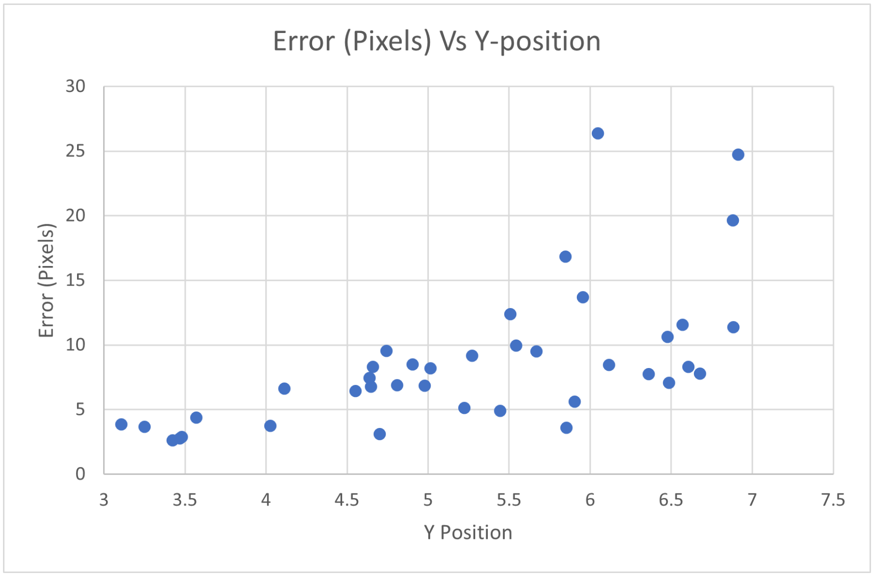


Figure 3.37: Camera pixel errors in relation to the Y-position

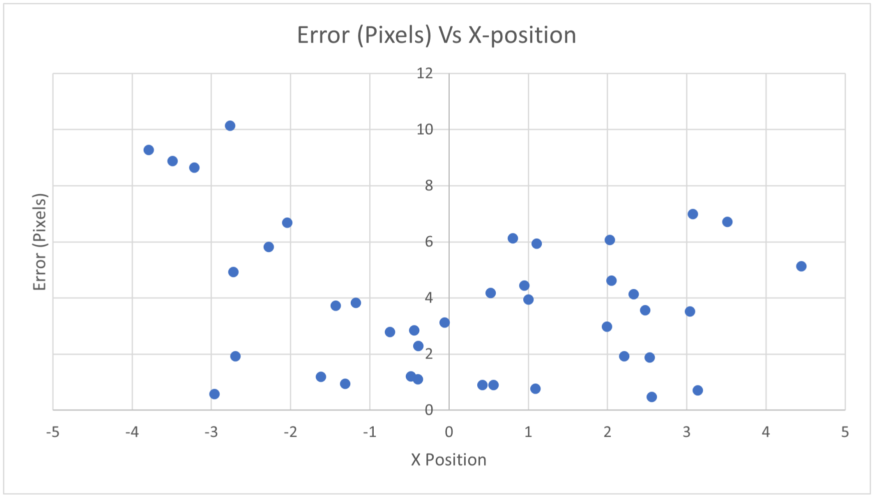


Figure 3.38: IR Camera pixel errors in relation to the X-position

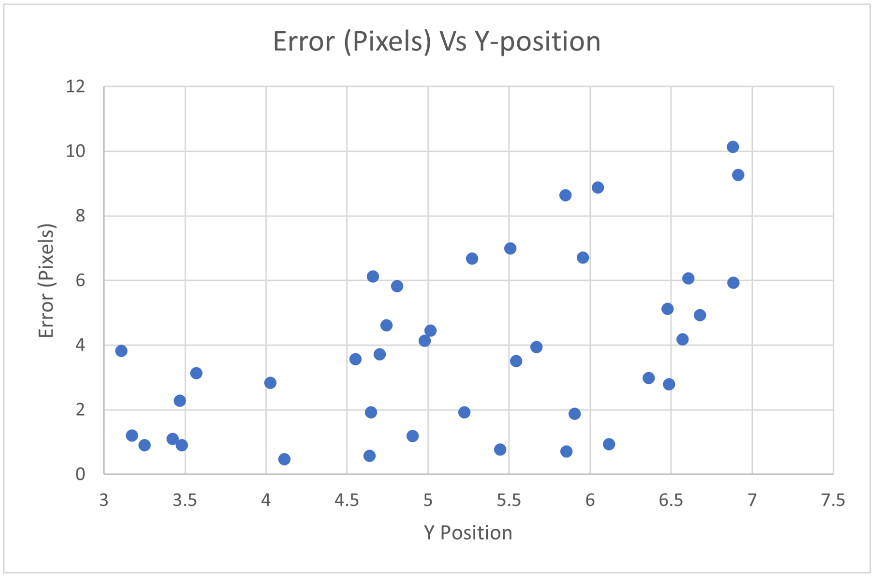


Figure 3.39: IR Camera pixel errors in relation to the Y-position

3.6 Conclusion

In this chapter, a multi-step process for calibrating camera-LiDAR systems is presented and evaluated. The procedure begins with the correction and calibration of the camera lens, followed by the automatic detection of features in both the LiDAR and the cameras. For LiDAR feature detection, the DBSCAN and RANSAC algorithms are utilized for clustering and identifying the checkerboard clusters used in the calibration process. Additionally, the vertex is detected using spatial relations. For the cameras, the image is converted to greyscale if needed, and then Gaussian filtering is performed. The Harris corner detection algorithm is employed to identify the checkerboard pattern and detect the vertices. Subsequently, the features detected by the LiDAR and camera are matched, and the projection matrix is approximated using the LM algorithm. An experiment was conducted to conclude that the outcome of this detailed procedure can achieve a relative pixel error performance of less than 2%, a result that is deemed successful. The main contributions of this chapter are the tailored calibration and fusion process pipeline design for the CMHT sensors and the implementation of the discussed pipeline.

Chapter 4

CMHT Dataset

4.1 Introduction

Using the vehicle and system discussed in previous chapters, datasets were collected in both urban and highway settings in Hamilton. The synchronized sensor capture system with ROS2 support enabled the capture of data from five distinct sensor types: LiDAR, IR camera, camera, IMU, and Radar. The resulting datasets were manually labeled and showcase several notable features:

- CMHT dataset is the first open-source research dataset for synchronized autonomous vehicles that includes IR images and Radar data captured under a variety of weather, road, and light conditions.
- The dataset has been manually labelled, including the labelling of IR camera, camera, and LiDAR, which provides object class and bounding box information for use in neural network training.
- The dataset showcases the performance of the sensors under different weather and light conditions.

CMHT dataset captures three distinct scenarios:

- A busy local mall parking lot captured during the rain at dusk, with overcast clouds and significant water accumulation on the ground.

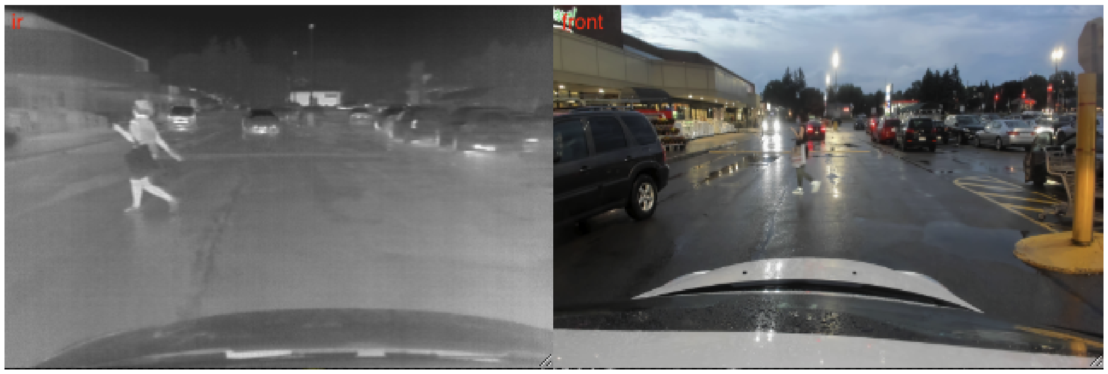


Figure 4.1: A data sample from the parking lot.

- A mixed scenario segment captured in two parts, with the first part taking place on a busy city street at night with numerous vehicles, and the second part taking place on a highway with fewer vehicles. Both segments were captured after complete darkness, with the only light sources being vehicle headlights and road lights. Additionally, the capture was done after heavy rainfall, resulting in water accumulation on the road.

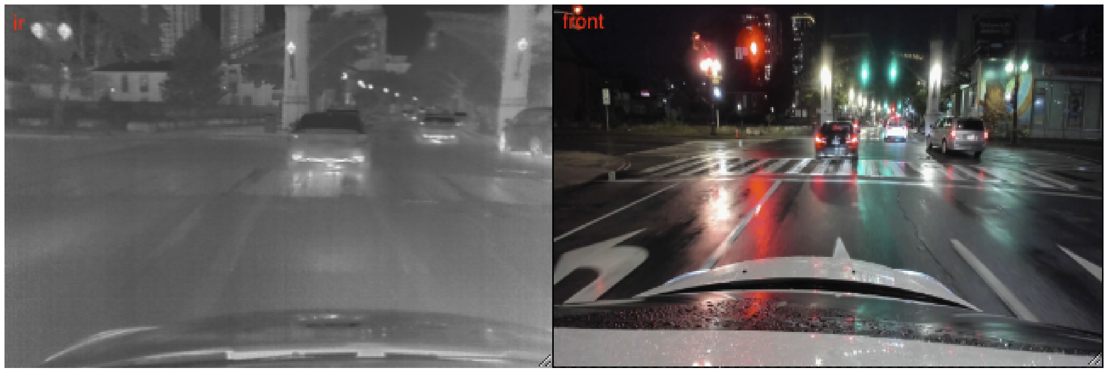


Figure 4.2: A data sample from the down town dataset

- A dense residential area at night with no moonlight and many parked vehicles on the side.

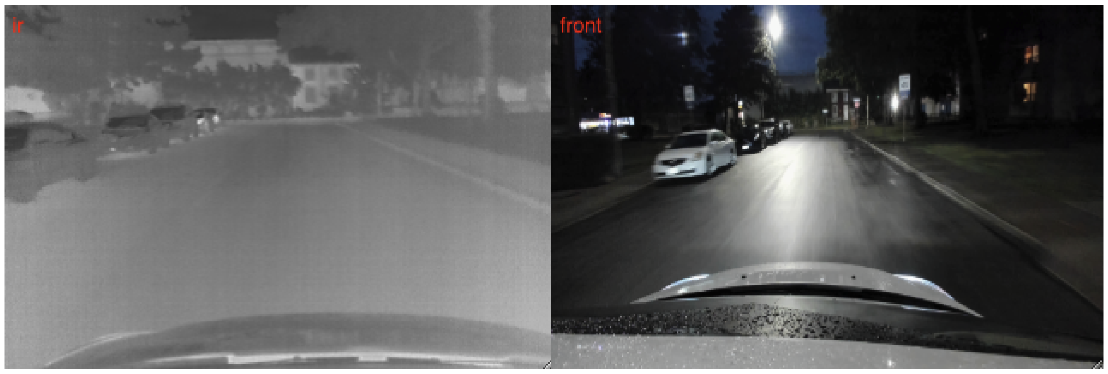


Figure 4.3: A data sample from the residential area dataset

This research selected specific locations and weather conditions for several reasons. Firstly, the scenarios in the selected locations comprised a high number of vehicles with some pedestrians, providing a diverse range of samples. Additionally, the weather conditions and lighting were not optimal, which was different from other datasets. This unique characteristic allowed for a comprehensive investigation of how weather impacts sensor readings. Furthermore, the study examined the differences in infrared (IR) readings between parked and running vehicles. Parking lots were deemed to be a suitable location for this comparison as they contain a mixture of stationary and moving cars. The chosen locations and conditions provide a valuable dataset for researchers to explore and gain insights into the impact of weather and vehicle status on sensor readings.

The data collection process involves a team of two: a car driver and an operator who manages the equipment. Before beginning, the team drafts a driving plan. Once the vehicle reaches the data collection location, the operator will start the sensor perception system in data capture mode, as described in the previous section. This step is necessary because the immense volume of raw data captured per second exceeds the hard drive's write speed. To manage this, the system is designed with a buffer to temporarily hold the data in memory. After the perception system is stopped in recording mode, it may take a few minutes to process all the stored buffer and write it to the hard drive. Therefore, due to the memory configuration of the processing laptop running the capturing system, there is a limit on the capturing time per run, which, in our configuration, is up to 20 minutes.

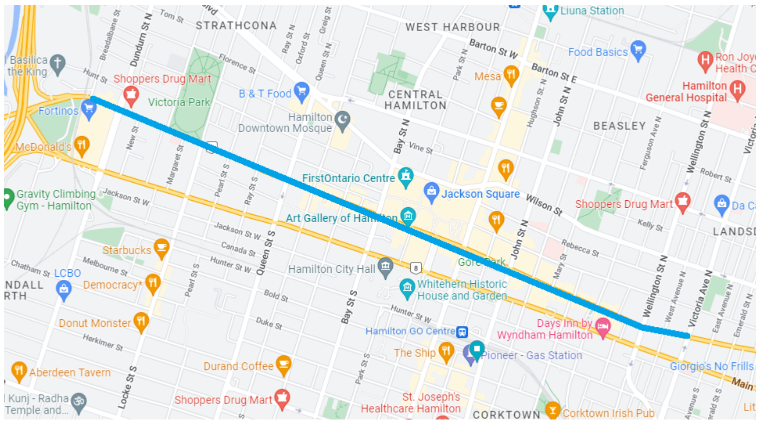


Figure 4.4: Scenario 2: Urban, downtown, after rain and dark.

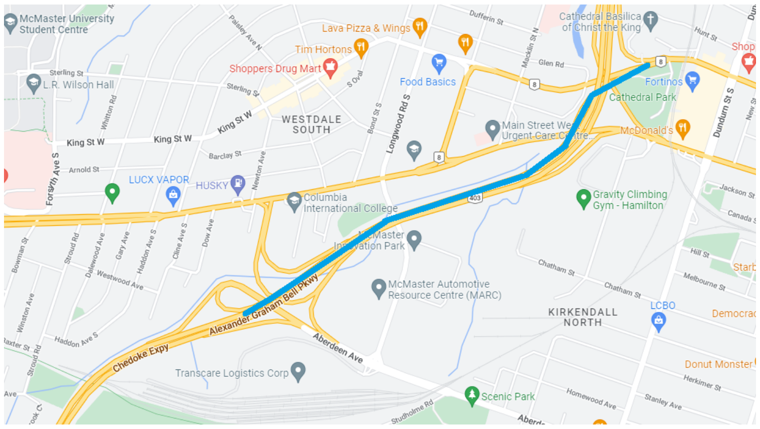


Figure 4.5: Scenario 2: Highway, after rain and dark..

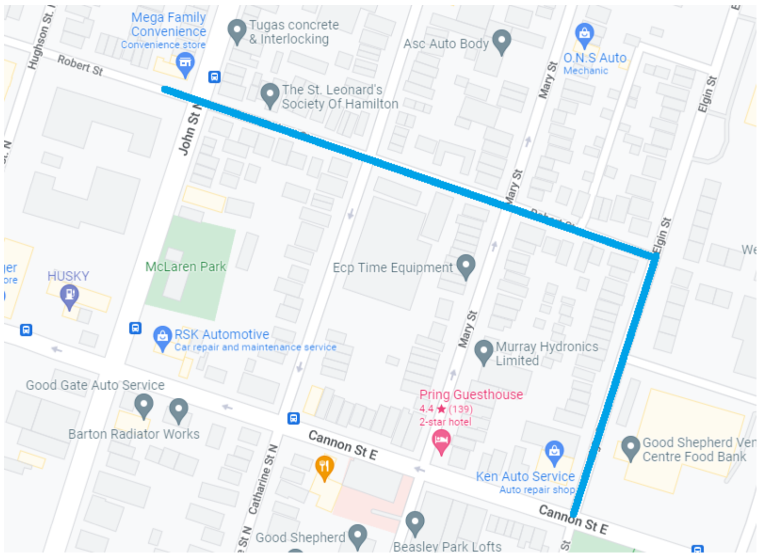


Figure 4.6: Scenario 3: Local, dark and after rain.

Table 4.1 showed the detailed statistic of the dataset, table 4.2 shows the statics on labelled objects from different driving scenario.

Scenarios	Scenario 1	Scenario 2	Scenario 3
Traffic Info	N/A	Busy	Quiet
Time	8:00PM	9:00PM	8:30PM
Data Size	2.4GB	8.7GB	3.7GB
Road Way	Parking Lot	Urban&Highway	Residential Area
Number of Labelled Objects	234	188	84
Type of Unique Objects	49	71	61
Number of Frames	1498	4100	2071
Number of Labelled Frames	85	56	24
Average Vehicle Speed	20km/h	60km/h	40km/h
Weather Condition	After Rain	Raining	After Rain
Daylight Condition	Dark	Dark	Dark

Table 4.1: Dataset statistic

4.2 Data Labelling

4.2.1 Labelling Tool

The dataset used in this study was labelled using the SUSTechPOINTS tool, as described in [101]. However, it should be noted that this labelling software employs a coordinate orientation that differs from that of the CMHT vehicle configuration. As such, a coordinate shift was necessary in order to align the two coordinate systems. This involved swapping the x and y axes, and defining the negative y axis as the front of the vehicle.

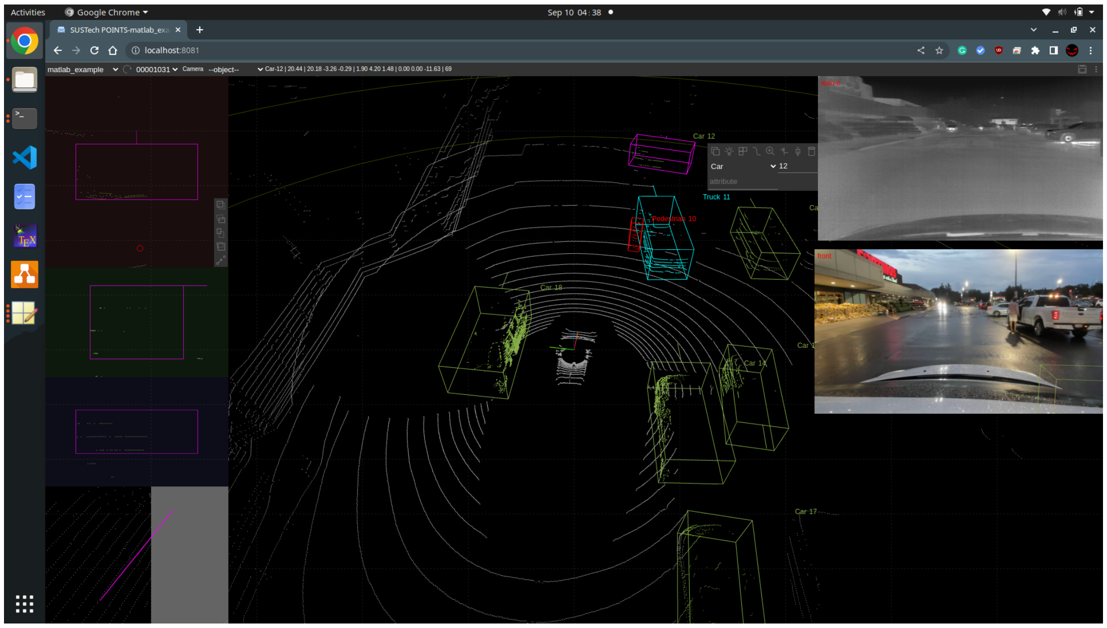


Figure 4.7: Labelling software interface

4.2.2 Labelling Format

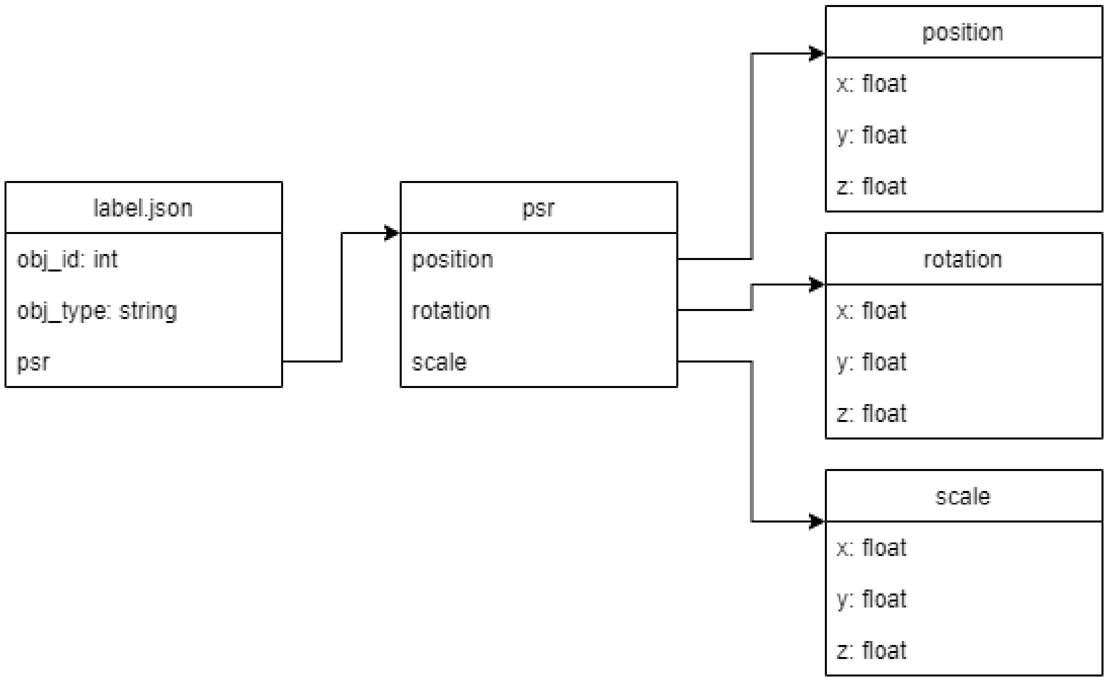


Figure 4.8: Json file data structure

The object is stored in a JSON file, and all coordinates are in the LiDAR coordinate system. Each JSON file name matches its corresponding frame ID. The JSON data format is in Figure 4.8. **obj_id** is the unique identifier of the object, and they are unique in a set, which meants all the objects from different frames that contains the same id are the same object. **obj_type** is the object class in a string, and **psr** is the bounding box attribute associated with an object within the frame.

4.2.3 Labelling Results

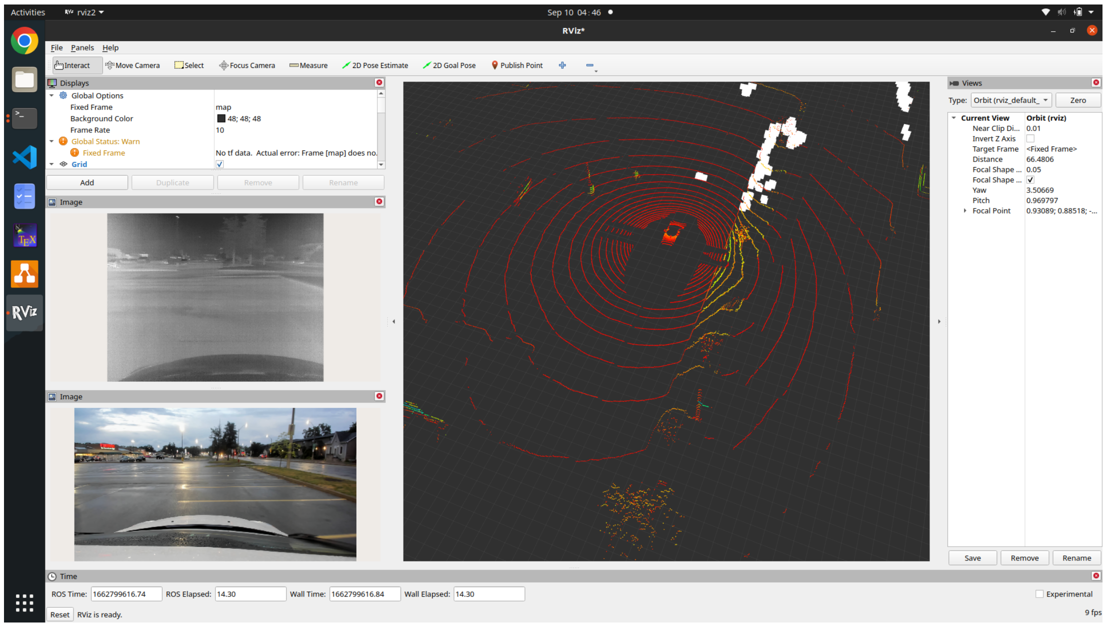
	Scenario 1	Scenario 2	Scenario 3
Car	182	133	71
Truck	11	15	5
Van	10	2	0
Pedestrian	31	15	4
Long Vehicle	0	6	0
Bus	0	6	1
Cyclist	0	4	1

Table 4.2: Number of unique objects in the scenario

4.3 SDKs

ROS2 and Matlab SDK packages is also included with the package with the following capabilities:

- Synchronized sensor data loader
- LiDAR and other sensor overlays
- Access to labelled object information includes bounding box and class ID.
- Bounding box projection onto the camera image
- Visualization



Displays	
Global Options	
Fixed Frame	map
Background Color	■ 48; 48; 48
Frame Rate	10
Global Status: Warn	
Fixed Frame	No tf data. Actual error: Frame [map] does no.
Grid	✓

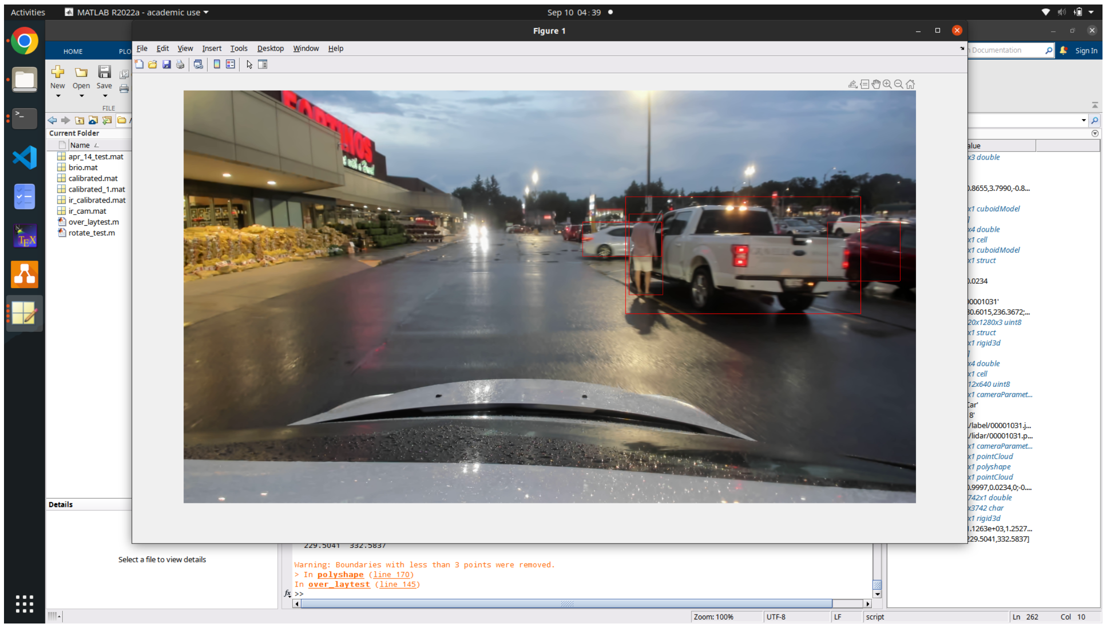
Type: Orbit (rviz_default_   Zero

Current View	Orbit (rviz)
Near Clip Di...	0.01
Invert Z Axis	☐
Target Frame	<Fixed Frame>
Distance	66.4806
Focal Shape ...	0.05
Focal Shape ...	✓
Yaw	3.50669
Pitch	0.969797
Focal Point	0.93089; 0.88518; -...

ROS Time: 1662799616.74 ROS Elapsed: 14.30 Wall Time: 1662799616.84 Wall Elapsed: 14.30 ☐ Experimental

RViz is ready. 9 fps

Figure 4.9: ROS2 interface, visualized with rviz2



apr_14_test.mat
brio.mat
calibrated.mat
calibrated_1.mat
ir_calibrated.mat
ir_cam.mat
over_laytest.m
rotate_test.m

```
  229.5041  332.5837

Warning: Boundaries with less than 3 points were removed.
> In polyshape (line 170)
In over_laytest (line 145)
>>
```

```
alue
x3 double
.8655,3.7990,-0.8...
x1 cuboidModel
]
x4 double
x1 cell
x1 cuboidModel
x1 struct
.0234
0001031'
30.6015,236.3672;...
20x1280x3 uint8
x1 struct
x1 rigid3d
]
x4 double
x1 cell
12x640 uint8
x1 cameraParamet...
Car'
8'
/label/00001031.j...
/lidar/00001031.p...
x1 cameraParamet...
x1 pointCloud
x1 polyshape
x1 pointCloud
.9997,0.0234,0;-0....
742x1 double
x3742 char
x1 rigid3d
.1263e+03,1.2527...
229.5041,332.5837]
```

Figure 4.10: Matlab Visualization

4.3.1 PointCloudData Set

This non-serialized package comprises the data structures illustrated in Figure 4.11. Camera and IR camera data are stored in the standard PNG file format, while LiDAR data is stored in PCD format, which can be easily read by many other standard tools or libraries.

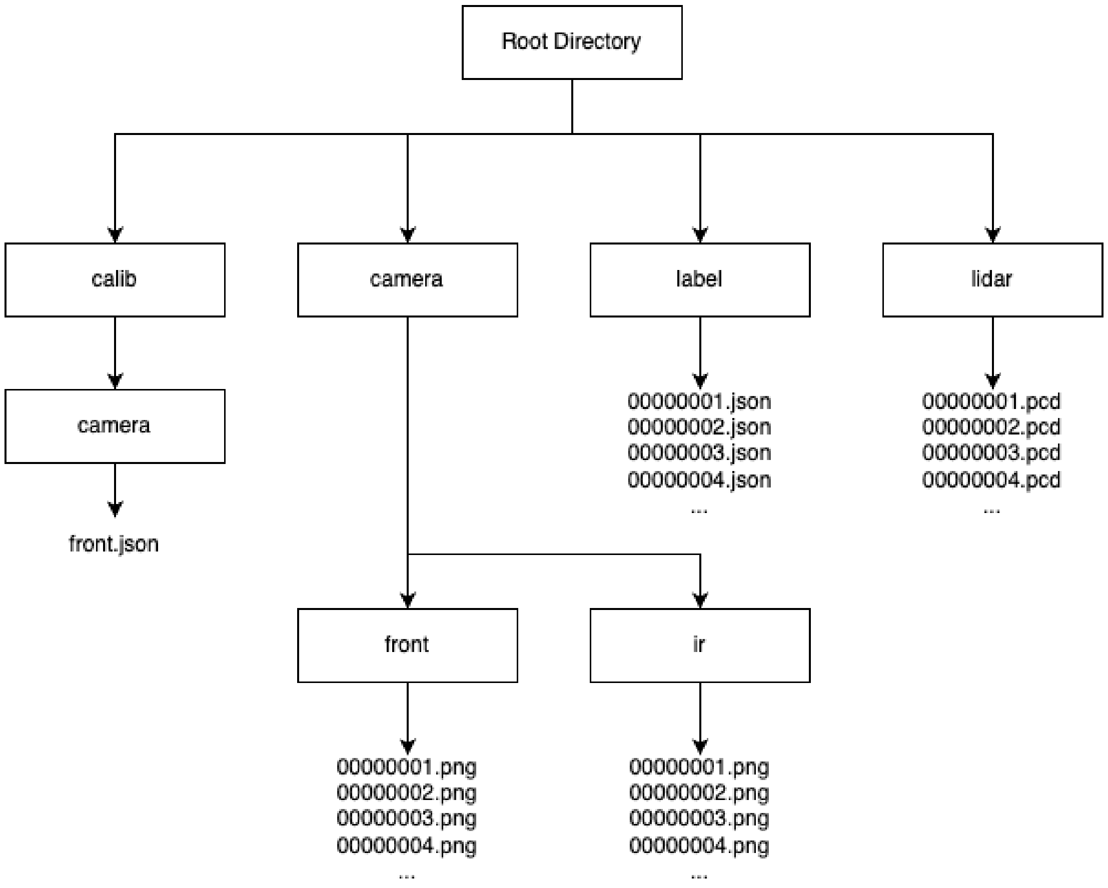


Figure 4.11: Directory structure, boxes are the folders. *front.json* file contains intrinsic matrix and extrinsic matrix(LiDAR-to-camera).

4.3.2 Utility Tools

4.3.2.1 Training Data Extractor

The development of a training data extractor for CNN training that included both image and LiDAR object inputs was a critical component of this research. The extractor was designed to crop out image data with 5% padding to the actual size to compensate for errors that may arise from the projection matrix used in

LiDAR to camera projection. Additionally, the extractor was able to extract LiDAR object clusters from the training data and store them in .PCD format. The extracted files were organized in a folder structure that made them easy to use, specifically for import by machine learning packages such as Tensorflow and Pytorch. This organizational structure facilitated the integration of the extracted data into the training pipeline and streamlined the training process.

4.3.2.2 Bounding Box Projection

A bounding box projection tool was included, developed using Python and Matlab. This tool was designed to translate bounding box objects from the 3D LiDAR coordinate to the 2D camera (u, v) coordinate. To achieve this, a series of calculations were necessary to project the bounding box object correctly onto the camera coordinate space.

Specifically, given a bounding box object with coordinates $p = \{x, y, z\}$, $r = \{x, y, z\}$, and $s = \{x, y, z\}$, first calculated the corresponding LiDAR coordinate corners. This involved using a set of geometric transformations to rotate and translate the bounding box object, such that its edges were aligned with the LiDAR coordinate axes. With the corners of the bounding box now defined in LiDAR space, then the calculation projected them onto the 2D camera coordinate space using a perspective transformation matrix.

$$\mathbf{pt}_{x,y,z} = \{p_x \pm \frac{s_x}{2}, p_y \pm \frac{s_y}{2}, p_z \pm \frac{s_z}{2}\} \tag{4.1}$$

Then the cuboid centre was moved to the origin.

$$\mathbf{pt}_{x,y,z}^{\text{de-centred}} = \mathbf{pt}_{x,y,z} - \mathbf{p} \tag{4.2}$$

Compute the rotation matrix $\mathbf{R}$.

$$\mathbf{R} = \begin{bmatrix} \cos(r_y)\cos(r_z) & \sin(r_x)\sin(r_y)\cos(r_z) - \cos(r_x)\sin(r_y) & \cos(r_z)\sin(r_y)\cos(r_z) + \sin(r_x)\sin(r_z) \\ \cos(r_y)\sin(r_z) & \sin(r_x)\sin(r_y)\sin(r_z) - \cos(r_x)\cos(r_y) & \cos(r_z)\sin(r_y)\sin(r_z) - \sin(r_x)\cos(r_z) \\ -\sin(r_y) & \sin(r_x)\cos(r_y) & \cos(r_x)\cos(r_z) \end{bmatrix} \tag{4.3}$$

Then rotate the corner set along the origin, and shift it back to the original position.

$$\mathbf{pt}_{x,y,z}^{\text{de-centred \& rotated}} = \mathbf{pt}_{x,y,z}^{\text{de-centred}} \cdot \mathbf{R} \tag{4.4}$$

$$\mathbf{pt}_{x,y,z}^{\text{rotated}} = \mathbf{pt}_{x,y,z}^{\text{de-centred \& rotated}} + \mathbf{p} \tag{4.5}$$

Thus $\mathbf{pt}_{x,y,z}^{\text{rotated}}$ is the computed coordinate points for the corners of the bounding box, and by using projection matrix M, the coordinate can be projected to the camera coordinate $\mathbf{U} = \{u_i, v_i\}, i \in \{0,7\}$. And final corners of the bounding box $\mathbf{U}_{\text{corners}}$ is computed using equation 4.6.

$$\mathbf{U}_{\text{corners}} = \begin{bmatrix} \max(u), \max(v) \\ \min(u), \max(v) \\ \min(u), \min(v) \\ \max(u), \min(v) \end{bmatrix} \tag{4.6}$$

4.4 Conclusion

This chapter introduced the novel CMHT dataset, explaining its labelling process and the format in which it is used. Additionally, the SDKs that work with the dataset for research purposes were also introduced.

Chapter 5

Road Object Detection

5.1 Introduction

This chapter and the following chapter presents a sensor fusion approach for the development of a road object detection and classification system. The methodology used in this research introduces a novel ground segmentation method, which facilitates efficient ground removal and object detection, and a hybrid neural network approach for object class identification. The proposed approach demonstrates promising results for road object detection and classification tasks.

An overview of the processing pipeline for the vehicle perception system is illustrated in Figure 5.1. The system comprises three main modules: the input module, which accepts synchronized sensor inputs; the pre-processing module, where each sensor undergoes specific pipelines for pre-processing; and the classifier module. In the pre-processing pipelines, the LiDAR data is initially segmented and clustered to derive object point clouds. Through the rectification process, the $\{x, y, z\}$ coordinates of each object point cloud are extracted to obtain the pixel coordinates on the image and IR image. An optional greyscale conversion can be applied to the camera images. Before transmitting the LiDAR object point cloud to the classifier, the 3D point cloud is transformed into a 2D side view. Following the side view mapping, an autoencoder is employed to fill gaps in the object, creating a dense 2D view.

All pre-processed sensor data is then input to the classifier. The LiDAR dense 2D view feeds into a modified VGG-like real-valued convolutional network. The camera sensors and IR camera sensors are fed into a special complex-valued neural network. The results from the different classifiers are then fed into a voter for the final class prediction. For implementation details on the classification modules, please refer to Chapter 6 of this thesis.

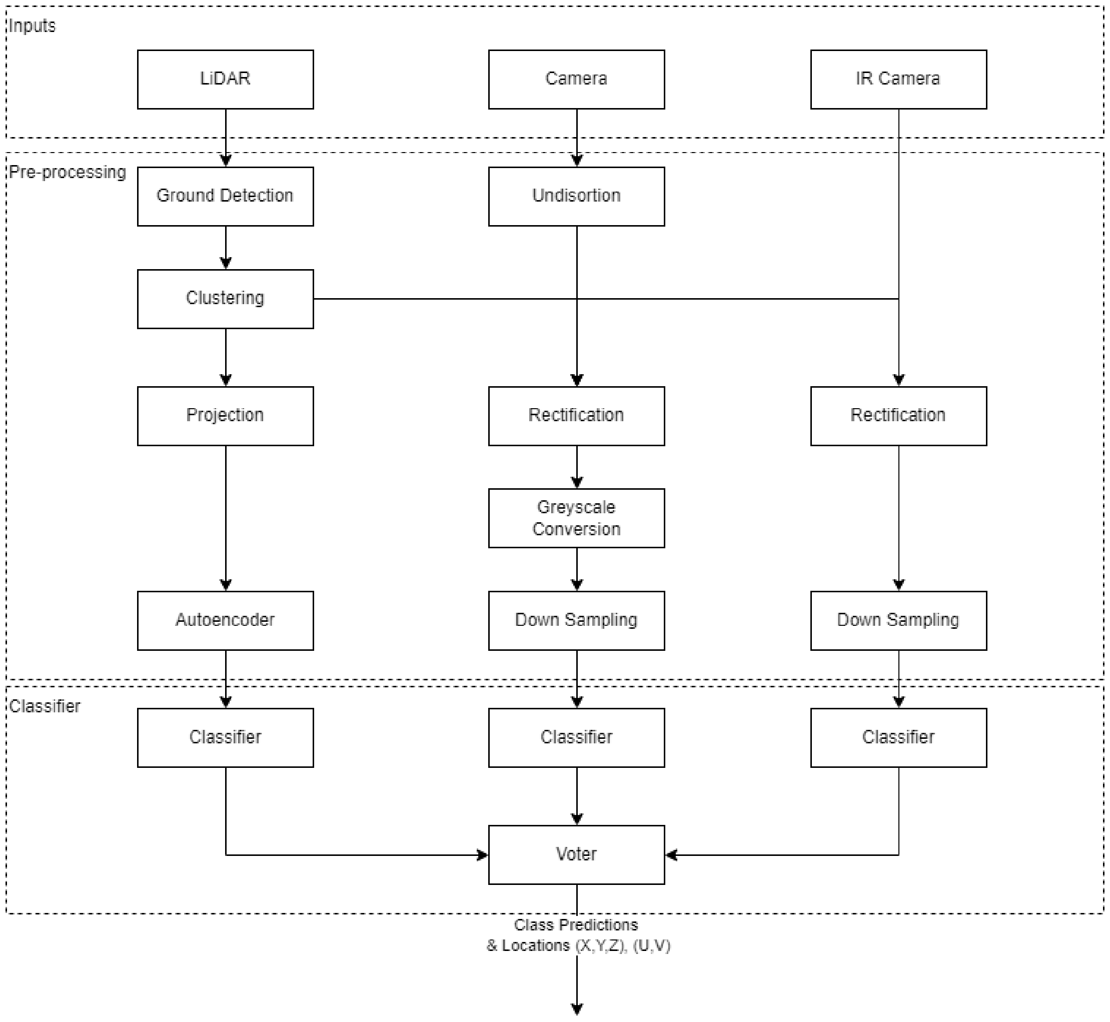


Figure 5.1: Overview

As its name suggests, the pre-processing phase is tasked with preparing the input data for further analysis. This phase encompasses seven procedures: sensor rectification, data filtering, and coarse-level feature extraction, among others.

Subsequently, the classification phase utilizes the output from the pre-processing phase. It employs a novel hybrid neural network to identify the object's class.

5.2 Related Research

5.2.1 Ground Segmentation

Ground segmentation is a widely researched area in the field of LiDAR. Its primary objective is to identify and segment the ground points within LiDAR point clouds data. The importance of this research lies in its ability to provide crucial information about the surrounding terrain to autonomous vehicles or other automated systems equipped with LiDAR. The information obtained through ground segmentation has many practical applications, including but not limited to path planning, object detection, SLAM, and road boundary detection.

The increasing accessibility of LiDAR sensors due to their decreasing costs has contributed to the growing popularity of this research area. As a result, researchers continue to investigate new approaches and algorithms to improve the accuracy and efficiency of ground segmentation methods. Popular ground segmentation methods can be classified into five categories based on their approaches:

- Elevation Map Approaches: The elevation map technique depends on the relationships between adjacent points to identify whether a specific point is a ground point or not, giving rise to the name "elevation map approach." Sebastian Thrun *et al.*'s study[157] was the first to develop a ground segmentation method using elevation maps. During that period, sensors were not as sophisticated as current LiDAR technology; nevertheless, the research ingeniously implemented five single-lane laser rangefinders mounted on the vehicle's roof, effectively mimicking the capabilities of modern LiDAR sensors. The ground points were identified by calculating the vertical distance between two nearby points; if the value exceeded a threshold, an obstacle was detected. This straightforward method had several drawbacks, particularly due to the sensor array's mounting on the vehicle; if the vehicle tilted, the readings could produce significant errors, resulting in misidentification. However, the study offered a solution to this problem by employing a first-order Markov model to enhance detection accuracy. B. Douillard *et al.*[45] utilized state-of-the-art LiDAR devices in their research, specifically focusing on ground segmentation. By utilizing the increased density of the point cloud produced by advanced LiDAR technology, the study introduced voxelization as an innovative approach to represent objects. In this method, objects were represented as intricate 3D voxels within a 2D cell of the terrain grid. This technique offers the advantage of generalizing the ground segmentation method and is applicable to high-resolution point clouds, allowing for the differentiation of objects from the ground. Zhihao Shen *et al.*[148] made significant advancements in the speed of ground segmentation using modern LiDAR devices with 64 or 128 beams and higher resolutions. They proposed a novel approach called Jump-Convolution-Process, which optimized the ground segmentation process. This method transformed the volumetric LiDAR ground segmentation problem into a 2D planar problem, thereby reducing the computational complexity. Additionally, by introducing the jump operation, the researchers were able to further decrease the computational load by solely processing the low-confidence points. Consequently, the proposed method achieved pseudo real-time performance, enhancing the efficiency of ground segmentation on high-resolution LiDAR systems.

- Methods of Occupancy Grid Maps: These were initially introduced by H. Moravec *et al.*[119] during the 1980s. Although the original work used wide-angle sonar sensors, the same methodology is applicable to modern LiDAR sensors. In their study, Moravec and his team created detailed grids to illustrate both occupied and free space within a noisy environment. This technique paved the way for the representation of terrestrial characteristics using grid-based occupancy maps, which are relevant to both traditional sonar sensors and current LiDAR sensors. Notably, Luo *et al.*[109] conducted research falls under this category. He utilized various features from the LiDAR sensor in his work to calculate the likelihood of a grid pertaining to the ground plane, and his methodd achieve highly accurate ground segmentation in real-time.

- Ground Modelling Method: This strategy views the ground as a planar surface and utilizes one of the three relationships to construct a model of the terrain: plane fitting, line extraction, or Gaussian Process Regression (GPR).

  Plane Fitting: the study conducted by Hu *et al.*[73] commences with the application of the RANSAC algorithm for plane estimation. Subsequently, a Gaussian model is employed on the projected LiDAR and mono-colour image to generate a predictive road probability map. Hu's work exhibits strength in robust road detection through the use of sensor fusion. However, due to the two-stage segmentation process and extensive data processing, its practical implementation in real-time scenarios poses a challenge. Patchwork [104] by Lim *et al.* and its subsequent variants [97] demonstrate real-time processing with relatively high accuracy. The method initially employs a concentric zone model by dividing LiDAR inputs into smaller fan-shaped regions. It then performs region-wise ground plane fitting on the divided regions using Principal Component Analysis (PCA), followed by the application of Ground Likelihood Estimation (GLE) to enhance estimation accuracy. In the Patchwork++ variant, the final stage of GLE is replaced with adaptive GLE to further improve performance. It is worth noting that in their paper [104], the authors specifically discuss the rationale behind choosing PCA over RANSAC for the GLE process. They draw the conclusion that although PCA slightly underperforms compared to RANSAC, it offers a significant speed advantage (up to two times faster) and is therefore a worthwhile trade-off for real-time applications.

Line Extraction: Himmelsbach *et al.*'s method [67] initially divides LiDAR data into smaller sections based on polar coordinates. Subsequently, for each "pie" section, the method performs local ground plane estimation, followed by clustering on the segmented objects. Stamos *et al.*'s work [151] focuses on ground segmentation using the scan line generated by the LiDAR sensor. This approach employs sets of rules to classify points into vegetation, horizontal surfaces, and vertical surfaces in a coarse classification. Finally, the results are refined using object relative position to the curbs to determine the classification of the object.

GPR: GPR is a well-developed and popular research topic in LiDAR ground segmentation. This method requires fine parameter tuning and often relies on specific sample sets for improved results. Several papers [46][32][95][82][158] showcase the use of GPR in the ground segmentation process. GPR is very close to the NN method, which requires a supervised training process. Lang *et al.*'s work [95] proposes a non-stationary covariance function to improve the ground detection accuracy in smooth or flat ground surfaces while preserving edges and corners, which sometimes is challenging to detect from naive GPR. Douillard *et al.*'s work [46] proposes two novel segmentation metrics in the process to improve its real-time performance and retain good accuracy. In his work, he introduced Gaussian Process Incremental Sample Consensus (GP-INSAC), an improvement to the GPR which improves the outlier rejection capability. Works like[32][82][158] are proposed to reducing computational complexity or introduce other features during the GPR process in order to improve the performance of ground segmentation.

- Methods based on the relationship between the adjacent points: Relation-based method utilizes the relation between local regions and features. Based on the LiDAR features, there are three approaches:

  Channel-Based: The Channel-Based approach utilizes the relationship between each laser head-/channel to perform ground segmentation. An example of this can be found in the work of Chu *et al.* [37], where they employ a divide-and-conquer technique. Initially, the point cloud is partitioned into smaller groups based on the vertical line/LiDAR channel. Subsequently, each group is subjected to analysis based on gradient, lost threshold point, and abnormalities features. This iterative process is repeated for each group until the entire LiDAR frame is effectively segmented. Moreover, the Channel-Based approach can be effectively combined with other methodologies. For example the research conducted by Jimenez *et al.* [81], who propose a novel strategy termed Channel Based Markov Random Field (MRF). Their approach integrates channel-based techniques with the MRF approach, resulting in a synergistic fusion of methodologies for better segmentation accuracy. Works such as [99][38] also belong to this category. While such a method is relatively fast and applicable in real-time applications, it naturally comes with a flaw. If the data comes from multiple LiDAR sensors or mixed sources, the channel relation would not exist, rendering the method inapplicable.

Region-Growing: Methods in this category often employ graph search algorithms and explore the relationship between adjacent points. The work by Moosmann *et al.* [118] suggested transforming the LiDAR scan into an undirected graph, where the LiDAR scan points in Cartesian notation serve as vertices. After constructing the graph, they proposed performing a search on the graph to segment the LiDAR scan based on local convexity. Na *et al.* [121] proposed a similar, albeit slightly different, approach. Firstly, they projected the LiDAR data onto a range image. Then, they applied features such as normal difference, height difference, and gradient to partition the LiDAR points, starting from a randomly selected seed point. Lastly, they refined the segmentation by considering average normal vectors and average position features. Works such as [88][164] also falls into this category.

Clustering: The clustering method is another approach to tackle the segmentation problem, employing a divided-and-conquer strategy similar to the Region-Growing method. However, unlike the Region-Growing method, which primarily relies on exploring the relationships between neighboring points, the clustering method is not solely dependent on such relationships. For instance, in Sagi Filin's work [51], a point clustering algorithm based on point position, tangent plane parameters, and relative height difference features is proposed to extract homogeneous segments from aerial LiDAR data. Biosca *et al.*'s work [19] suggests the use of fuzzy clustering methods on LiDAR data to eliminate the need for defining samples or predefined clusters. Their algorithm also operates effectively with ground-captured LiDAR data. Yang *et al.*'s work [173] follows a similar approach, wherein LiDAR data is initially grouped by features, followed by Support Vector Machines (SVM) classification of the LiDAR points.

Range Images: Typically presented in Cartesian coordinates, LiDAR data takes the form $p = < x, y, z >$. However, an alternative representation exists in polar form as $p = < \phi, \theta, d >$. This polar representation is a direct outcome of the data acquisition process. Analyzing the inherent relationships between data points allows visualization of this polar representation as range images (see Example 5.2). Several works [25][33] fall within this category.

- **MRF**: The MRF approach leverages the spatial continuity among adjacent LiDAR points. It employs the MRF model to categorize points into either ground or non-ground classes. Notable works in this domain include [175] and [81].

- **Neural Network Approach**: In recent years, there has been a significant increase in research on neural networks (NN), leading to their widespread application in LiDAR ground segmentation. The prevalent approach in LiDAR ground segmentation involves converting LiDAR sparse data into LiDAR-image, LiDAR-image-like channel-based feature maps, or other 3D to 2D transformations. Subsequently, convolutional neural networks (CNN) are employed, treating the transformed data as images. Several works, including [162] [3] [130] [162] [150] [75], have adopted this approach. Often, NN ground segmentation methods also include object classification for single-stage classification tasks. However, due to the limitations of the depth of neural networks and computational constraints, these methods often lack generalizability and require specific training to achieve satisfactory results.

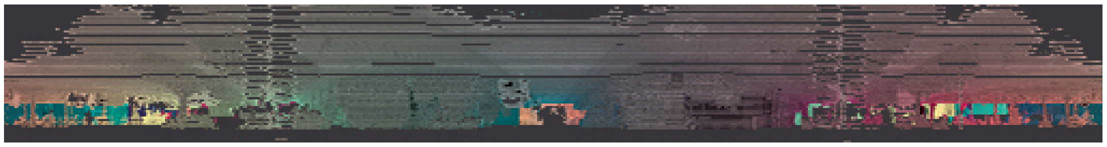


Figure 5.2: Example of LiDAR data converted into range image, x,y,z data is embedded into R,G,B channels, thus it creates a coloured image

5.3 Implementation

5.3.1 Ground Detection Using Savitzky-Golay Filter

This research utilizes a modified version of the grid-based ground detection method. The ring-shaped characteristic of LiDAR scans is exploited, leading to the creation of trapezoid zones within each segment. Through identification of local maximum and minimum points within each section, efficient projection of a ground plane and differentiation between object points and ground points is achieved. The main goal of ground segmentation is prompt detection of an object serving as the region of interest (ROI) for subsequent classification. As a result, the method prioritizes computational speed over attaining high accuracy.

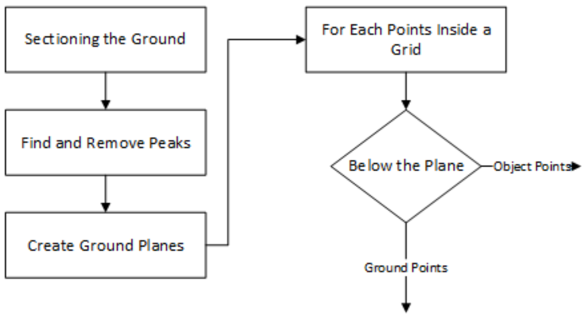


Figure 5.3: A process overview of modified grid-based ground detection method

- The first step involves segmenting the input into trapezoid-shaped grids(Figure 5.4).
- The scan set is then projected onto a 2D plane , resulting in a new set of coordinates $\begin{bmatrix} x' & y \end{bmatrix}$.
- Next, a novel peak detection algorithm is applied to each trapezoid region. The identified peak regions correspond to objects, while non-peak points are considered as estimates of the ground.
- After removing all the identified peaks from the section set, the remaining points are assumed to be ground points. A ground plane is then constructed based on the remaining points of the adjacent regions.
- For each point within the grid, a pre-defined ground plane is employed to ascertain whether the point belongs to the ground or an object.

5.3.1.1 Point Sectioning

This choice of shape segmentation is rooted in the inherent nature of LiDAR scans, where scans taken at closer distances tend to exhibit higher density than those obtained from further distances. By employing trapezoid grids, this density variation is taken into consideration, ensuring that the size of each grid area is smaller for closer scans in comparison to those at greater distances. Define θ is the number of sectors per ring, and δ is the distance per ring sector.

$$d\theta = \frac{360.0}{\theta} \tag{5.1}$$

$$\begin{bmatrix} x_{i,j,1} & y_{i,j,1} \\ x_{i,j,2} & y_{i,j,2} \\ x_{i,j,3} & y_{i,j,3} \\ x_{i,j,4} & y_{i,j,4} \end{bmatrix} = \begin{bmatrix} -d\delta \times (j) \times \cos(d\theta \times (i)) & d\delta \times (j) \times \sin(d\theta \times (i)) \\ -d\delta \times (j+1) \times \cos(d\theta \times (i)) & d\delta \times (j+1) \times \sin(d\theta \times (i)) \\ -d\delta \times (j+1) \times \cos(d\theta \times (i+1)) & d\delta \times (j+1) \times \sin(d\theta \times (i+1)) \\ -d\delta \times (j) \times \cos(d\theta \times (i+1)) & d\delta \times (j) \times \sin(d\theta \times (i+1)) \end{bmatrix} \tag{5.2}$$

Equation 5.2 defines 4 boundary points to create the trapezoid grid.

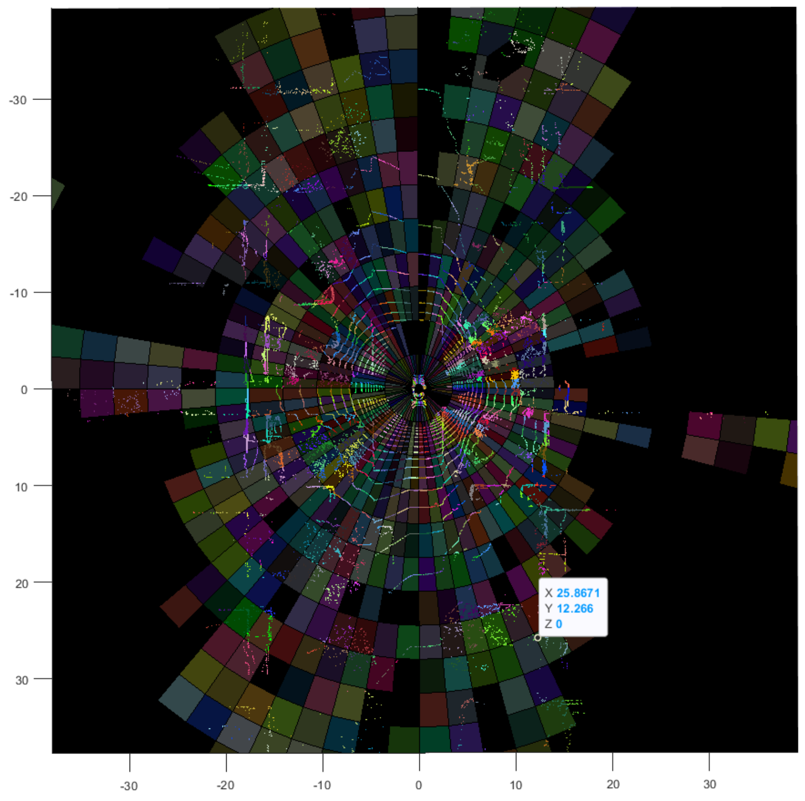


Figure 5.4: Sectioning the LiDAR cluster points into grids.

After defining the planer grids, the cluster points (x, y, z) are placed into a defined trapezoid grid.

$$\left[\mathrm{G}_{i,j}\right] = \begin{bmatrix} x_{i,j,1} & y_{i,j,1} & z_{i,j,1} \\ x_{i,j,2} & y_{i,j,2} & z_{i,j,2} \\ x_{i,j,3} & y_{i,j,3} & z_{i,j,3} \\ x_{i,j,4} & y_{i,j,4} & z_{i,j,4} \end{bmatrix} \tag{5.3}$$

$\mathrm{G}_{i,j}$ is the set of coordinates to define the grid plane. To calculate the height $z_{i,j,k}$, use equation 5.4. In the case of a missing grid due to lack of scan points, LiDAR placement height z_{camera} will be used.

$$\begin{bmatrix} z_{i,j,1} \\ z_{i,j,2} \\ z_{i,j,3} \\ z_{i,j,4} \end{bmatrix} = \begin{bmatrix} \min\left(z_{\text{cluster points inside } \mathrm{G}_{i,j}}\right) \\ \min\left(z_{\text{cluster points inside } \mathrm{G}_{i,j+1}}\right) \\ \min\left(z_{\text{cluster points inside } \mathrm{G}_{i+1,j+1}}\right) \\ \min\left(z_{\text{cluster points inside } \mathrm{G}_{i+1,j}}\right) \end{bmatrix} \tag{5.4}$$

5.3.1.2 Peak Detection

Once all the points have been partitioned into grids, the peak detection method is applied to identify any peak points concerning the z-axis. In the absence of any detected peak point coordinates, the value z_{peak} is substituted with the adjusted z-value. Before peak detection, all values are first converted to their absolute values to enable the identification of local minima using the same general approach simultaneously.

A method to extract the peaks was developed using four steps: 1) The first derivative was approximated by filtering the data through Savitzky-Golay coefficients[142] ; 2) The result was filtered by using the N-points average filter; 3) all local maxima and minima were located and grouped; 4) The location of peaks were found using the maxima and minima.

Savitzky-Golay Filter The filter uses Savitzky-Golay coefficients to perform the convolution by fitting data into a low degree polynomial by the method of linear least squares. It can effectively reduce white noises in the power spectrum while maintaining the correct shape of the curve. Savitzky-Golay coefficients can be pre-computed, but the steps are long. However, they only need to be computed once per different window size m and polynomial degree n. Once they are calculated, they can be reused many times until new parameters are set. To get the Savitzky-Golay coefficients, a special $(2m+1)\times(n+1)$ Vandermonde matrix v (5.6) was computed. Note this matrix is extremely ill-conditioned and one should be cautious when using it under IEEE double precision system.

$$w_i = -M + i, \forall i \in \{0,1,...,2M\} \tag{5.5}$$

$$v_{i,j} = \begin{cases} 1, & \text{if } j = 0. \\ w_i \times v_{i,j-1}, & \text{if } j \neq 0. \end{cases} , \forall i \in \{0,1,...,2M\}, \forall j \in \{0,1,...,N\} \tag{5.6}$$

Figure 5.5: Vandermonde matrix v generated from window matrix w with window size m, polynomial degree n.

After the upper triangular matrix has QR de-composited from the Vandermonde matrix v using the Gram-Schmidt process, the coefficient matrix $C = vR^{-1}(R^{-1})'$ was calculated. One unique feature about the Savitzky-Golay coefficients is the first row of the coefficient matrix C will compute filtered Y of the input function $y(x)$, the second row of the coefficient matrix C will compute the first derivative of Y with respect to

x and then the third row of the coefficient matrix C will compute the second derivative of Y with respect to x, and so on.

The second row was used from pre-computed Savitzky-Golay coefficients to calculate the filtered first derivative value of the elevation with respect to the distance.

$$Y_j = \sum_{i=\frac{-m-1}{2}}^{\frac{m-1}{2}} C_{i,2} y_{j+i}, \frac{m+1}{2} \leq j \leq n - \frac{m-1}{2} \tag{5.7}$$

Figure 5.6: Apply Savitzky-Golay coefficients to the input to calculate the first derivative.

N-point Moving Average Filter At this stage, the first derivative of the power spectrum is not “smooth” enough for locating the maximum and minimum, therefore a N-point moving average filter was used to “smooth” the input.

$$Y_j = \sum_{i=\frac{-m-1}{2}}^{\frac{m-1}{2}} \frac{y_{j+i}}{m}, \frac{m+1}{2} \leq j \leq n - \frac{m-1}{2} \tag{5.8}$$

Figure 5.7: Apply a N-point moving average filter with window size m to the input y_j.

Local Maxima and Minima Grouping The next step is looking for all the local maxima and minima. In this context, a maximum is the value that is higher than the previous and next values, and a minimum is the value that is less than the previous and next values. Ideally, the maximum and minimum would be easy to identify on a smoothed-out curve, but it is not always a guarantee the N-points moving average filter can smooth the curve to such a degree. Nevertheless, it is still useful since it greatly reduces the number of calculations in this step. All detected maximum and minimum were grouped together, otherwise excessive numbers of minimum and maximum in a small area would have resulted in same peak detected multiply times.

In this grouping algorithm, it is the sequence X which contains all the maximum or minimum within the range σ. The first x_1 is always the first maxima or minima detected from the previous part, then moving alone all detected local maxima and minima. This grouping method ensures there are no two adjusting peaks within a distance less than σ.

X is a sequence, contains $\{(x_1,y_1),(x_2,y_2)...(x_n,y_n)\}$

with order $x_i < x_{i+1}$, and range $x_n - x_1 \leq \delta$

(x_i,y_i) is either a detected local maximum or a local minimum

If exist $|y_i - y_{i+1}| \leq \delta$ in X

then remove (x_i,y_i) from X, replace (x_{i+1},y_{i+1}) with $(\frac{1}{2}(x_i+x_{i+1}),\frac{1}{2}(y_i+y_{i+1}))$

Repeat the process until no more $(x_i,y_i), |y_i - y_{i+1}| \leq \delta$ exist in X

Locating Peaks Once all the local maxima and minima were found, the peaks were located using the following rules: if a maximum is followed by a minimum and $(y_{maximum} - y_{minimum}) > \psi$, then the peak is located at $\frac{1}{2}(x_{maximum} + x_{minimum})$. Threshold ψ is the value required to be determined as a valid peak. The idea behind this peak detection method is to look for a steep change of the first derivative to determine if it is a valid peak.

5.3.1.3 Creating Ground Planes

A ground plane is a hyper-plane defined by three points, $\mathbf{P},\mathbf{Q},\mathbf{R}$, where these points are randomly selected from the remaining points within the same trapezoidal grid $G_{i,j}$. It is mathematically described by the following equations:

$$\mathbf{PQ}_{i,j} = \begin{bmatrix} x_{i,j,1} - x_{i,j,2} & y_{i,j,1} - y_{i,j,2} & z_{i,j,1} - z_{i,j,2} \end{bmatrix} \tag{5.9}$$

$$\mathbf{PR}_{i,j} = \begin{bmatrix} x_{i,j,1} - x_{i,j,3} & y_{i,j,1} - y_{i,j,3} & z_{i,j,1} - z_{i,j,3} \end{bmatrix} \tag{5.10}$$

$$\mathbf{N}_{i,j} = \mathbf{PQ}_{i,j} \times \mathbf{PR}_{i,j} \tag{5.11}$$

Normal vector $\mathbf{N}_{i,j}$ is calculated from the pre-defined plane coordinates.

5.3.1.4 Determine the Ground Points

To determine if point (x_k,y_k,z_k) inside a grid $G_{i,j}$ is a ground point, check if the point is below the plane by a threshold σ amount.

$$r = \begin{bmatrix} x_{i,j,4} - x_k & y_{i,j,4} - y_k & z_{i,j,4} - z_k \end{bmatrix} \cdot \mathbf{N}_{i,j} \tag{5.12}$$

Thus if $r > \sigma$, then the point (x_k,y_k,z_k) is an object point, otherwise it is a ground point.

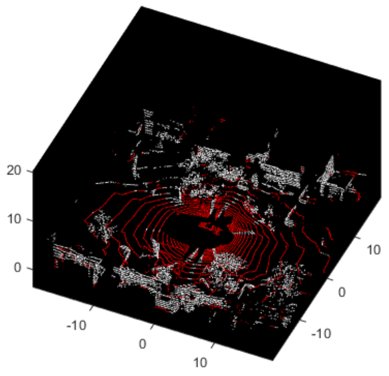


Figure 5.8: Detected ground points (coloured in red).

5.4 Results

In this research, the timing of Ground Segmentation is crucial for achieving real-time performance. The system is constrained to a 100ms processing time, meaning that the quicker the ground is segmented and objects are clustered, the better the overall performance will be. With this constraint in mind, the benchmark prioritizes speed over accuracy, recognizing that reduced processing time is key to the success of thesystem.

In this benchmark test, a performance similar to the setup in the work conducted by Oh *et al.*[126] was achieved. Specifically, a total of 10 different ground segmentation methods were evaluated, including the proposed one, across 5 driving scenarios. Among these scenarios, 3 were residential, 1 highway, and 2 city captures. The data used for the benchmark came from the KITTI semantic benchmark[56]. The result is shown in Table 5.1, 5.2, 5.3, and 5.4, with details on the benchmark run provided in the figures from the Ground Segmentation Performance Diagram section in the Appendix.

	Sequence 00 Residential	Sequence 01 Highway	Sequence 02 City	Sequence 03 Residential	Sequence 04 City	Sequence 05 Residential
chen2014gaussian[32]	14.42ms	15.26ms	16.20ms	17.02ms	17.98ms	14.79ms
githubplaneground[165]	14.43ms	12.93ms	16.00ms	15.18ms	15.83ms	15.48ms
githubransac[44]	67.54ms	38.18ms	69.61ms	67.56ms	57.02ms	68.80ms
himmelsbach2010fast[67]	9.60ms	8.05ms	9.11ms	9.50ms	10.38ms	11.29ms
lim2021erasor[103]	23.93ms	20.40ms	24.02ms	25.61ms	24.87ms	26.80ms
lim2021patchwork[104]	21.07ms	17.95ms	21.13ms	21.80ms	21.65ms	23.60ms
narksri2018slope[122]	69.38ms	57.39ms	72.18ms	77.56ms	74.96ms	78.85ms
proposedmethod	11.28ms	8.32ms	12.93ms	11.53ms	12.17ms	11.67ms
shen2021fast[148]	14.15ms	13.29ms	15.06ms	13.45ms	13.40ms	13.83ms
zermas2017fast[175]	15.19ms	13.01ms	15.27ms	16.70ms	16.25ms	15.90ms

Table 5.1: Average processing time

	Sequence 00 Residential	Sequence 01 Highway	Sequence 02 City	Sequence 03 Residential	Sequence 04 City	Sequence 05 Residential
chen2014gaussian[32]	1.99ms	2.79ms	2.52ms	2.24ms	2.66ms	2.50ms
githubplaneground[165]	1.74ms	1.71ms	1.68ms	1.39ms	1.44ms	1.91ms
githubransac[44]	7.06ms	19.51ms	7.24ms	8.88ms	12.60ms	9.58ms
himmelsbach2010fast[67]	2.10ms	1.98ms	1.97ms	1.82ms	2.54ms	4.27ms
lim2021erasor[103]	2.59ms	3.54ms	2.13ms	2.00ms	1.90ms	2.70ms
lim2021patchwork[104]	2.30ms	3.03ms	1.90ms	1.79ms	1.42ms	2.43ms
narksri2018slope[122]	12.10ms	19.89ms	10.91ms	10.65ms	8.68ms	10.72ms
proposedmethod	2.73ms	3.14ms	2.66ms	2.33ms	2.75ms	2.63ms
shen2021fast[148]	2.33ms	1.77ms	2.12ms	1.99ms	1.55ms	1.63ms
zermas2017fast[175]	1.80ms	2.22ms	1.50ms	1.93ms	1.77ms	1.80ms

Table 5.2: Standard deviation of the average processing time

	Sequence 00 Residential	Sequence 01 Highway	Sequence 02 City	Sequence 03 Residential	Sequence 04 City	Sequence 05 Residential
chen2014gaussian[32]	95.94	89.59	96.33	96.43	93.41	96.35
githubplaneground[165]	96.19	65.71	91.00	92.73	95.32	95.83
githubransac[44]	93.85	95.91	90.37	89.83	96.61	93.46
himmelsbach2010fast[67]	83.05	74.22	84.13	80.87	80.07	85.37
lim2021erasor[103]	96.57	89.70	96.77	97.61	93.43	96.83
lim2021patchwork[104]	94.61	89.07	93.58	95.69	91.25	95.54
narksri2018slope[122]	70.69	73.45	69.39	68.36	72.51	68.02
proposedmethod	94.74	84.98	88.16	89.80	92.78	93.78
shen2021fast[148]	40.61	64.29	58.52	61.71	75.81	57.36
zermas2017fast[175]	79.03	23.52	74.80	75.25	72.55	87.58

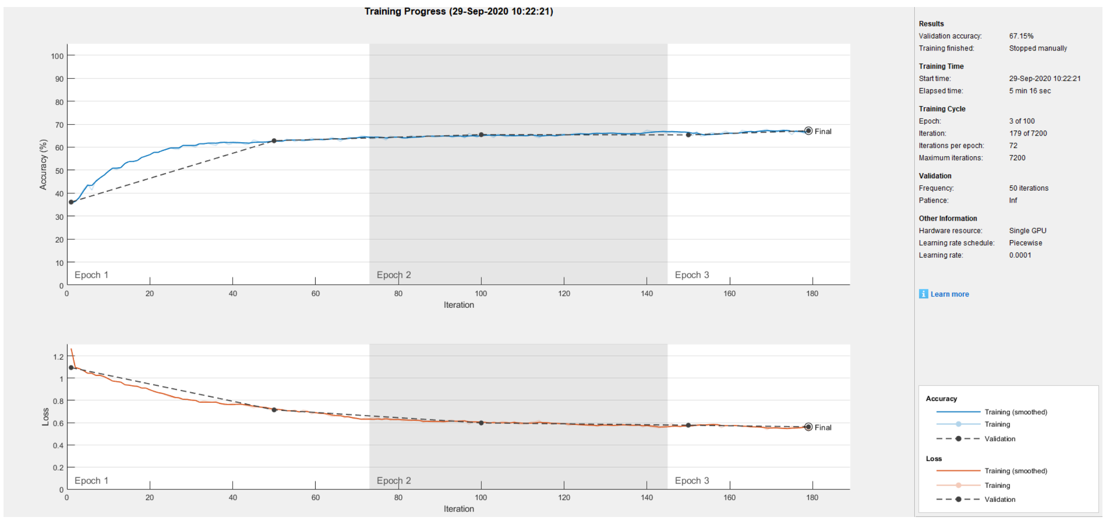
Table 5.3: Average recall

	Sequence 00 Residential	Sequence 01 Highway	Sequence 02 City	Sequence 03 Residential	Sequence 04 City	Sequence 05 Residential
chen2014gaussian[32]	81.03	92.47	83.99	83.18	94.99	80.53
githubplaneground[165]	91.87	91.06	93.81	93.93	95.70	88.43
githubransac[44]	88.03	96.46	91.03	92.80	95.68	85.75
himmelsbach2010fast[67]	98.15	98.89	97.96	95.38	99.51	96.65
lim2021erasor[103]	59.21	89.71	69.03	74.47	84.56	60.45
lim2021patchwork[104]	92.34	95.89	93.24	90.45	97.43	89.22
narksri2018slope[122]	91.66	97.10	91.89	92.45	98.17	86.35
proposedmethod	80.77	91.03	88.34	88.25	91.93	78.78
shen2021fast[148]	95.25	98.28	96.89	97.33	98.74	89.74
zermas2017fast[175]	94.96	92.06	96.21	97.71	95.42	93.62

Table 5.4: Average precision

From the table, the proposed method ranks as the 2nd fastest among all tested methods, only slightly falling behind the method by Himmelsbach *et al.*[67]. However, in terms of accuracy, the proposed method boasts a superior recall rate compared to Himmelsbach's method with a slightly lower precision. Since the work centres on the ROI, and the extraction of the ground serves as a form of noise filtering (since the ground regions are not required), partially misidentified object points are not overly concerning. This lack of concern arises from the presence of an autoencoder to repair and reconstruct objects with missing points further down the pipeline, making recall more valuable than precision in this context. Consequently, the proposed method is deemed the most appropriate for the research application. In situations where false detection of objects has occurred, they will be addressed in the classification chapter. Often, these false detections will be labeled as the NaV (not a vehicle) class.

In addition to the main study, exploration into one-stage ground-segmentation and object classification methods was conducted, such as the work by Wu *et al.*[172]. Unfortunately, after numerous attempts, better results were not obtained by applying their method to the dataset. The outcome of these efforts is depicted in Figure 5.10, where it is apparent that many detections have large chunks of the ground section missing from the detection. Furthermore, achieving the training accuracy stated by the original paper proved unattainable, as shown in Figure 5.9.



Results
Validation accuracy: 67.15%
Training finished: Stopped manually

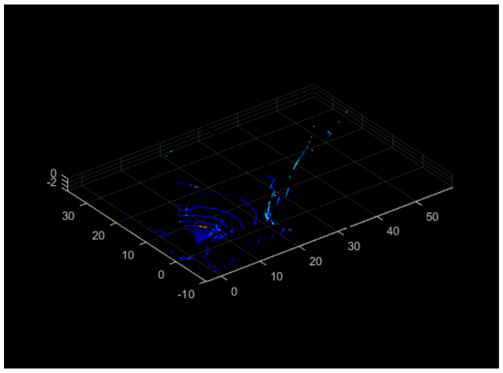
Training Time
Start time: 29-Sep-2020 10:22:21
Elapsed time: 5 min 16 sec

Training Cycle
Epoch: 3 of 100
Iteration: 179 of 7200
Iterations per epoch: 72
Maximum iterations: 7200

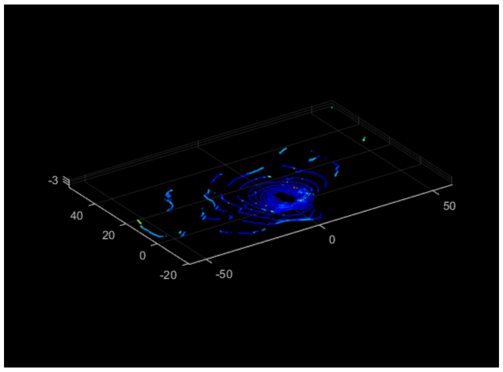
Validation
Frequency: 50 iterations
Patience: Inf

Other Information
Hardware resource: Single GPU
Learning rate schedule: Piecewise
Learning rate: 0.0001

Figure 5.9: Training log



(a) Result from the precision



(b) Ground truth

Figure 5.10: Example of unusable one-stage detector

5.5 Conclusion

In this chapter, a rapid ground segmentation method is introduced by utilizing a novel peak detection approach that leverages the Savitzky-Golay filter. The proposed method was applied to the KITTI benchmark dataset and compared with 9 other top-tier ground-segmentation methods. This technique succeeded in achieving a swift processing time while maintaining a high recall rate, fulfilling the expectations and requirements for this vehicle perception research.

Chapter 6

Road Object Classification

6.1 Introduction

This chapter continues the vehicle perception process, focusing on the clustering and classification of the ground-removed LiDAR data that was introduced in the previous chapter. By using the transform matrix obtained from Chapter 3, the LiDAR cluster data is projected onto the camera and IR camera data to isolate the ROI. Following the processing of the ROI, the data is then input into a hybrid neural network (Figure 6.1) for the purpose of classification. Finally, the data is passed through a voter to determine the object class.

In addition to the above, this chapter provides a detailed exposition of the neural network utilized in the research. It thoroughly explains the composition of the individual layers, and elaborates on the supervised training process. Finally, it exhibits the benchmark performance of each classifier individually, as well as the overall performance of the entire perception system using the captured data from Chapter 4.

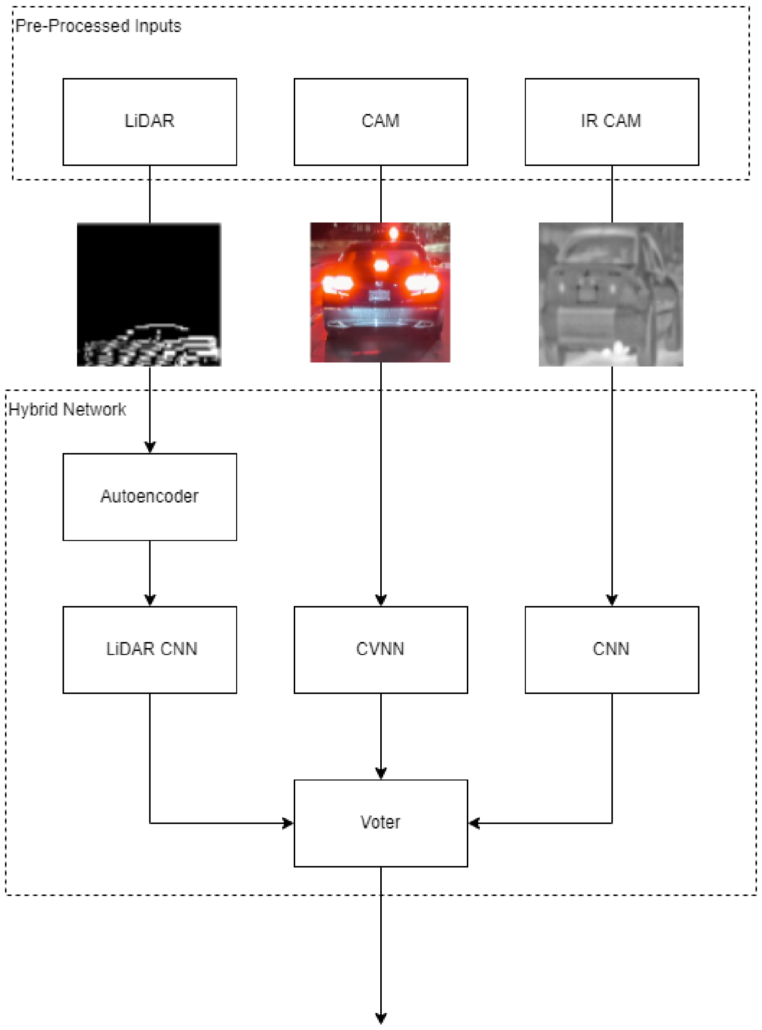


Figure 6.1: An overview of the Hybrid Neural Network

6.2 Related Works

6.2.1 Object Classification In Vehicle Perception

LiDAR-based classification LiDAR-based vehicle classification is not as prevalent as camera-based classification. This is primarily due to the limited resolution of LiDAR sensors, which hampers their ability to achieve high-performance results. Additionally, the cost of LiDAR sensors restricts their widespread availability. LiDAR-based classification papers [138, 139] mostly focus on leveraging the 3D features unique to LiDAR, distinguishing them from 2D camera images. 3D neural networks have been developed to tackle this task, but their effectiveness is limited due to scaling issues. However, not all approaches rely solely on 3D features. Some papers [89] segment the LiDAR 3D point clouds into camera-like images and apply CNN on these LiDAR-images. This method treats the 3D LiDAR clusters as low-resolution, noiseless images, yielding only satisfactory results. A notable paper [139] takes advantage of the 3D input by creating multi-views based on the 3D data and feeding them into a classifier. This technique achieved outstanding results and outperformed volumetric CNN approaches. However, implementing the method described in the paper is not entirely feasible in this research due to the practical limitation of acquiring complete 3D scans of specific objects. Nonetheless, the paper compared its approach with volumetric methods and reported faster processing times and improved prediction rates.

Camera-based classification The standard procedure for camera-based classification involves first removing the background and shadows to extract the region of interest (ROI), followed by applying various classification techniques on the extracted ROI to predict the class of the object. Unlike LiDAR-based ground detection, extracting the ROI from camera images requires additional steps to filter out shadows. This shadow removal algorithm is still under development and introduces instability to the system. A paper by Hsieh et al. [72] introduced a feature-based classification method using camera images. The paper proposed a method that eliminated the shadow regions in the image by utilizing lanes as references and predicting shadows based on the differences observed. Another paper by Zhang et al. [176] employs a simplified classifier to determine the class of a vehicle based on its physical properties. The method involves counting the pixel length of the vehicle and classifying it as either a large or small vehicle. This approach is useful to some extent but cannot determine the detailed class of the car.

Sensor fusion classification Vehicle classification through sensor fusion is currently concentrated primarily on data-level sensor fusion. The more intuitive approach is to correlate data, often done by employing a rectification process, and then feeding the rectified data into a neural network. This process pipeline is far from perfect: features may be redundant, and missing information from specific sensors can lead to complete failure.

A paper published in 2018 by Waltner *et al.*[167] proposes the idea of using an autoencoder to reconstruct LiDAR 3D points with camera images to form a mixed-depth view. This approach remedies the problem of sometimes having incomplete scans of a vehicle, with the potential for improved classification rates. However, a drawback of this reconstruction is that it comes only from one angle, which can be limiting in certain situations, such as when the input data is corrupted to the extent that it cannot be reconstructed correctly.

Another paper by Giering *et al.*[59], employs a traditional approach by utilizing optical flow to detect the ROI, and creates a fused input of ROI that combines RGB from a camera, UV from optical flow, and L (depth) from the LiDAR. The object is then fed into a classifier.

Alternatively, Asvadi *et al.*'s research[8] presents a unique fusion approach: the ROI is detected through ground detection followed by object extraction, unlike the previous paper where a camera was used. This paper then suggests combining image and LiDAR point clouds to form a depth-based image, which is subsequently fed into an NN.

6.2.2 Neural Networks

6.2.2.1 Complex-valued neural network (CVNN)

CVNN network defines a very similar structure as a Real-valued neural network (RVNN) counter part, with exception that inputs are usually complex-valued, or converted into the complex domain from the real domain through various method (for example Fourier transform.)

Intuitively, CVNN were initially developed mainly for applications in signal processing, as highlighted in [69] and [30], where the focus is predominantly on complex numbers. Recent studies, such as [49] and [34], have extended the application of CVNN to broader fields, including MRI and radar detection. Research works like [6] and [5] have shown that shallow CVNN can outperform RVNN in applications beyond those inherently associated with complex numbers, mainly excelling in faster convergence.

A paper[161] published by Tygert *et al.* in 2016 gives mathematics motivation and proper definition for a possibility of applying CVNN specifically Complex-valued convolutional network (CVCN) in real-world applications, for example, image processing and signal processing. This paper describes CVCN as a form of calculating non-linear multi-wavelet packets, thus defining the network structure as a variant of ”multi-wavelet transform”. Inspiring by Mark Tygert and others’ work, another paper[134] published in 2017 contributed to the line by deriving first order backpropagation training (gradient descent) for such CVCN. However, the paper is proposing to accept a real-value image as input, yet not testing nor working on a complex-value input. Never the less, the article still found CVCN performs better than Deep Neural Network (DNN) in MINST and CIFAR-10, and conclude that CVCN can greater outperform DNN in a larger network.

Fourier neural networks (FNN) Paper[129] discuss the possibility of using Fourier or wavelet transform results to form descriptors as the input for the neural network. The experiment in the paper is very significant as it describes the possibility of using such techniques on images.

Paper[107] propose a FNN structure for pattern matching. The novelty of the paper is its interesting k-neural define: an input is decomposed into a sequence of sine and cosine terms, learning coefficients are the different frequencies of the sine and cosines. The paper shows the experiment was a success and FNN structure not only fast to train, but also has better accuracy.

Fourier Convolutional Neural Networks In their work [137], Harry *et al.* defined Fourier Convolutional Neural Networks (FCNN). This paper advocates for the incorporation of the Fast Fourier Transform (FFT) within the convolutional layer, aiming to reduce computational demands. Furthermore, it introduces the novel concept of high-pass pooling. The authors argue that the high-frequency band holds more critical information than the low-frequency band. By implementing high-pass pooling, the neural network is able to retain more information compared to the traditional spatial max-pooling method.

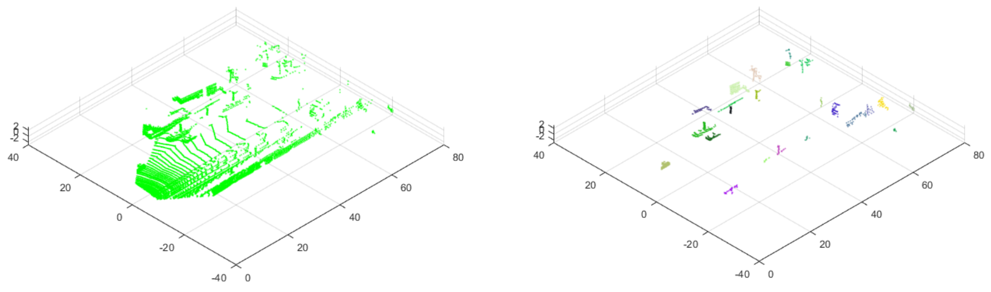
A distinct structured Fourier Convolutional Neural Network is proposed in [135], differing from the FCNN described above. This paper is a continuation of work based on the previous year’s paper by the same author [134]. In this version of the FCNN, a 2D Fourier transform is performed on the input, and the remaining aspects are inherited from the previous work. Unlike the application of a holomorphic activation function in the earlier structure, this paper applies the Rectified linear unit (ReLU) activation function separately to the real and imaginary components of the input, and then combines them.

6.3 Implementation

6.3.1 Pre-processing

6.3.1.1 Clustering

Density-based Spatial Clustering of Applications with Noise DBSCAN[50] is a clustering algorithm that groups points based on their distance to neighboring points. Unlike other methods that require knowledge of the number of clusters, DBSCAN has a complexity of $O(n^2)$, making it suitable for the given application. In previous research[108], Luo utilized the K-means algorithm for segmentation. However, after comprehensive testing, it was determined that DBSCAN yielded better results, albeit at the cost of slightly slower speed. Advancements in CPU processing power have significantly mitigated this computational delay, compensating for the increase in computational complexity. The algorithm itself is detailed in Chapter 3.



(a) Front only LiDAR-view. (b) Clustered objects (removed ground points)

Figure 6.2: Object clustered.

After removing all the ground points from the LiDAR input set, remaining points are assumed to be object points. Grouping those points into smaller clusters which are likely belonging to one object allows us to extract position and dimensional features such as length, width, and height of the object. By carefully tweaked thresholds, the result is visually shown in figure 6.2.

6.3.1.2 ROI Cropping

Utilizing the object clusters detected by the LiDAR, the ROI within the image can be identified by employing the transform matrix computed in Chapter 3. To account for potential errors in projection, a bounding box slightly larger than the designated cluster points of the LiDAR is drawn on the image. Subsequently, the image is segmented into various ROIs, as illustrated in Figure 6.4, to facilitate further analysis. The decision to draw an expanded bounding box is made to offset the synchronization discrepancies between the camera and LiDAR systems.

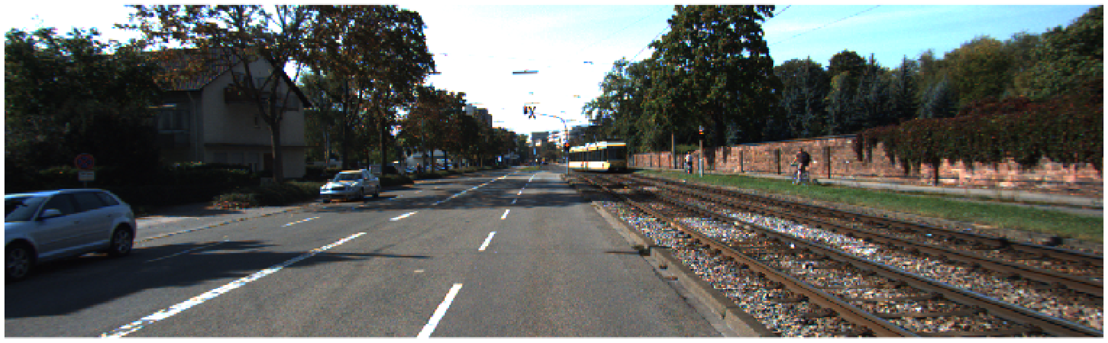


Figure 6.3: KITTI data: camera original view.

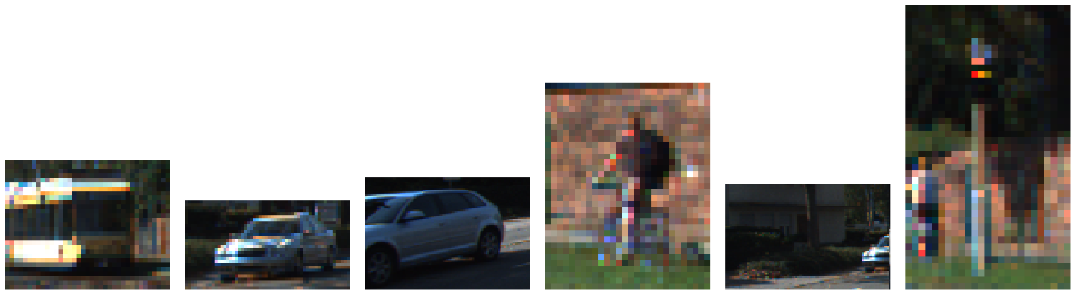


Figure 6.4: Various object ROIs cropped from the original image (figure 6.3).

6.3.1.3 Down Sampling

In instances where the extracted ROI is excessively large, it may be necessary to downsample it to a more manageable size through the application of a 2D Fourier transform. When downsampling the image in the spatial domain via max pooling, it becomes apparent that a substantial amount of information is likely to be lost, a phenomenon visually demonstrated in Figure 6.5. Conversely, performing downsampling within the frequency domain confers two notable advantages. Firstly, frequency-based downsampling preserves a more visually recognizable image compared to spatially downsampled representations, thereby implying a greater retention of information throughout the process. Secondly, as the subsequent procedure necessitates a complex-valued input, this method of downsampling facilitates the conversion of real-valued data into a meaningful complex-valued form, aligning with the requirements of the next processing stage.

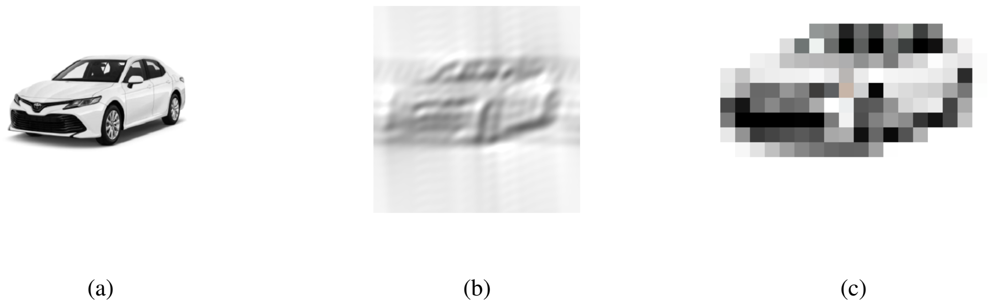


Figure 6.5: Extreme down sampling comparison: Figure 6.5a is the original 400 × 400 pixels image. Figure 6.5b is a 20 × 20 pixels image from the inverses 2D Fourier transformation of the down sampled complex values. Figure 6.5c is a 20 × 20 pixel image down sampled in spatial domain by max pooling.

6.3.1.4 Projection

The LiDAR point cloud cluster is projected into three different views, top, side, and rear (Figure 6.6), thus creating three 2D-LiDAR clusters. Then they are projected on a grid to form a range image.

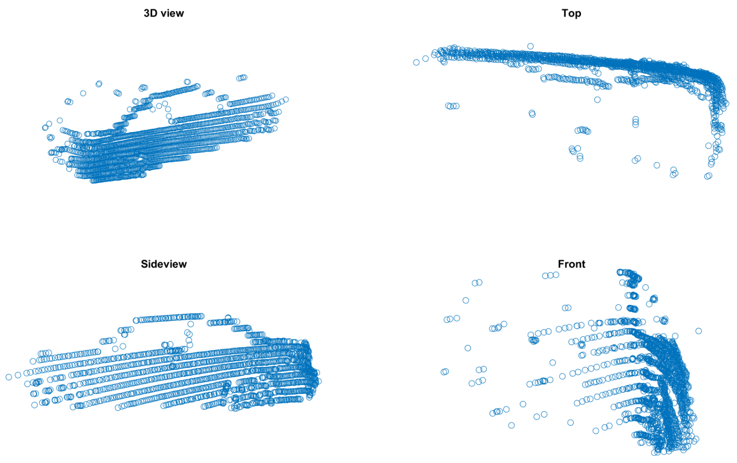


Figure 6.6: View projection

For example, the side view will take y-z points (Equation 6.1) and projects them onto a plane(Figure 6.7).

$$P_{side} = \{y_k^j, z_k^j \,|1 \leq j \leq 32; 0 \leq k < 2\pi\} \tag{6.1}$$

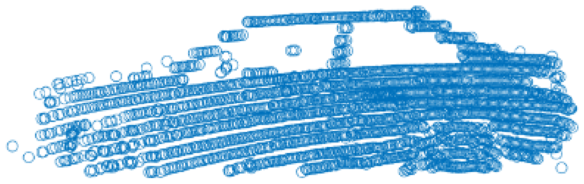


Figure 6.7: Side view LiDAR image.

In the context of vehicular navigation at intersections, discrepancies in orientation may be detected by LiDAR scans. To rectify these orientation differences, a specific approach is implemented when an object exhibits a width w greater than its length l, and a measurement exceeding 2.2 meters. Specifically, the x axis is employed for side view projection rather than the y axis. This decision rests on the underlying assumption that no vehicle will exceed a width of 2.2 meters, thus ensuring a consistent framework for orientation alignment.

6.3.2 CNN

A Convolutional Neural Network (CNN) [94] is a particular structure of neural network that incorporates local connectivities within its defining convolutional layer, as illustrated in Figure 6.8. By sharing kernel parameters across different regions of the input, this layer enables the neural network to handle larger inputs with a reduced number of neurons. This property not only contributes to the efficiency of the model but also significantly diminishes the overall size of the neural network.

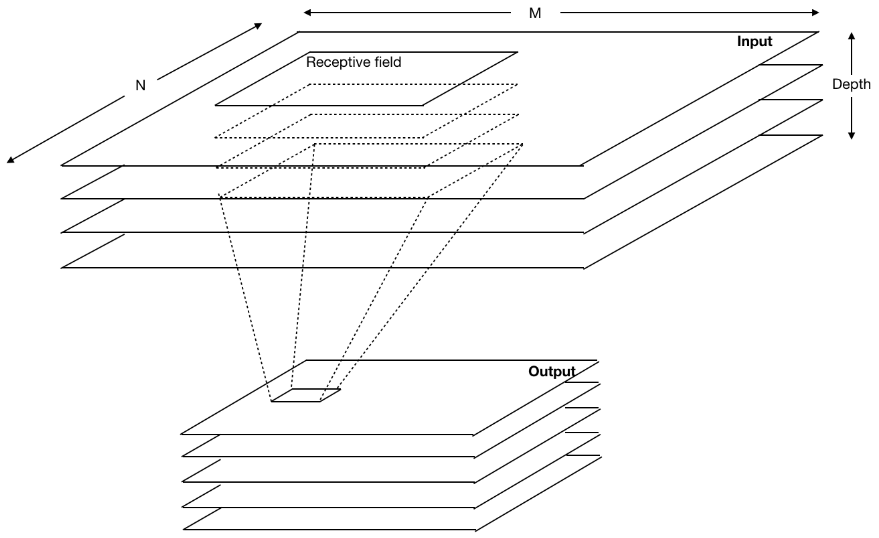


Figure 6.8: Convolving Process

Figure 6.9 shows a typical CNN layout.

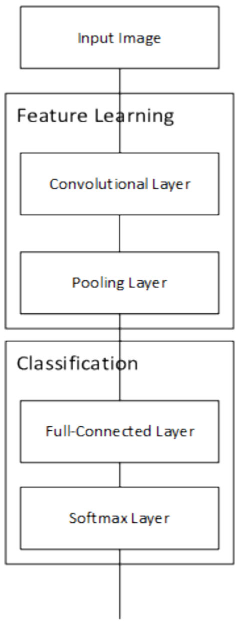


Figure 6.9: CNN structure diagram

6.3.2.1 Input Normalization

Usually, inputs require normalization to achieve the best result. In this context,a zero-mean normalization was used and after normalizing the input, the average of the input became zero.

$$x' = \frac{(x - \hat{x})}{\sigma_x} \tag{6.2}$$

Normalization (Equation 6.2) is applied to all the inputs X, with each input calculates with means $\hat{x}$ and standard derivation σ_x individually. x is the input.

6.3.2.2 Activation Functions

An activation function serves as a gating mechanism, controlling whether to enable or disable the input from the preceding layer. In certain literature, the activation function is delineated from the convolutional layer and classified into a distinct activation layer. It may also be referred to by its specific function name, such as the ReLU layer.

Ideally, an activation function should possess specific characteristics. It must be either bounded or locally bounded, ensuring that its output remains within a specified range. Furthermore, it should exhibit monotonic behaviour, meaning that it either consistently increases or decreases across its domain. Lastly, it must be differentiable at every point within the bounded region, allowing for the application of gradient-based optimization techniques.

ReLU Function The Rectified Linear Unit (ReLU) function, described by Equation 6.3, is a widely used activation function within the hidden layers, specifically in the feature extraction stage. This function is characterized by a hard threshold at the value of 0, retaining all positive values while nullifying the negative ones. However, a limitation of the ReLU function is that it can sometimes lead to the vanishing gradient problem. This occurs because the gradient is 0 for negative input values, which can result in ”dead neurons” where the corresponding weights are never updated due to the zero gradient. To address this issue, a variant known as Leaky-ReLU (Equation 6.4) was introduced. Unlike the standard ReLU, the Leaky-ReLU function assigns a small positive value to negative inputs during the gradient computation. This modification helps to mitigate the problem of dead neurons by ensuring that the weights can still be updated even when the input falls into the negative region, thus preventing complete stagnation in those parts of the network.

$$\mathrm{ReLU}(x) = \max(0, x) \tag{6.3}$$

$$\text{Leaky-ReLU}(x) = \max(0.01x, x) \tag{6.4}$$

6.3.2.3 Kernel Initialization

The initialization of the kernels was performed using the well-established Glorot initialization method, as elaborated in the seminal work of Glorot and Bengio [60]. This approach is widely adopted for kernel initialization, due to its nuanced strategy in selecting initial values that carefully balance between being too small or too large.

The importance of this method can be highlighted through the following considerations. When initializing a kernel, excessively small starting values may cause the gradients to vanish or exert an insignificant impact on the output. This could, in turn, lead to a protracted training process. Conversely, excessively large values for kernel initialization can induce instability within the entire artificial neural network (NN), causing the gradients to explode. The Glorot initialization method mitigates these issues by selecting the initial values based on the number of input and output connections, thereby ensuring that the chosen values are neither too small nor too large.

$$K_{ij} \sim U\big[-\frac{1}{\sqrt{n}}, -\frac{1}{\sqrt{n}}\big] \tag{6.5}$$

$U\big[-a, a\big]$ is the uniform distribution, and n is the number of outputs from the previous layer.

6.3.3 Layers

6.3.3.1 Convolutional Layer

A convolutional layer cross-correlates the inputs with kernels to extract information. The resulting output is then selectively fed into the next layer by an activation function. If the input size is not divisible by the kernel size (convolving window), it is zero-padded to match the required length.

$$\text{Conv}_{\mathbf{K}^l, b^l}(x) = \text{F}(\sum_{i=0}^{d-1} K_i^l * x_i + b^l) \tag{6.6}$$

$*$ is the cross-correlation operation, K_i^l is the layer l i^{th} kernel, x is the input, b^l is the layer l bias, and F is the selected activation function.

Cross-correlation Cross-correlation (Equation 6.7) takes the most calculations inside a convolutional neural network. Performing cross-correlation in the spatial domain is a costly computational process with complexity $O(n^2)$. Note in neural network, cross-correlation is often loosely called convolution.

$$y^{i,j} = f \sum_{l_1=\lfloor -m_k/2 \rfloor}^{\lfloor m_k/2 \rfloor} \sum_{l_2=\lfloor -n_k/2 \rfloor}^{\lfloor n_k/2 \rfloor} x^i_{l_1,l_2} k^j_{k_1-l_1,k_2-l_2} + b^{i,j} \tag{6.7}$$

m_k, n_k are the convolving matrix size, x, k are the input matrices, and y is the result.

Parameters r is the row of the receptive field, and c is the column of the receptive field. s_r stride is number of row element skips between each convolutional iteration. Respectively, s_c stride is number of column element skips between each convolutional iteration, and f fibre is the depth of the receptive field. p_c is the number of padding zeros added to the input.

Stride Stride is a parameter that allows the filter to skip inputs in order to downsample and speed up the calculation process. Parameters are (s_r, s_c), which is the number of inputs skipped during row calculation. Respectively, s_c is number of inputs skipped during the column calculation.

Padding When inputs cannot be evenly divided by the receptive field, it will produce cyclic wrap-around effect. This undesired behaviour could be fixed by adding zeros to the input, thus the process is called padding. Typically in a CNN, same-padding technique are deployed to make the output dimension the same as input.

Output The output of the convolutional layer will be a new tensor with size ($m' \times n' \times d'$) calculated from the equations 6.8.

$$\begin{aligned} m' &= \frac{m - r + 2 \times p}{s_r} + 1 \\ n' &= \frac{n - c + 2 \times p}{s_c} + 1 \\ d' &= d - f \end{aligned} \tag{6.8}$$

6.3.3.2 Inverse-Convolutional Layer

An inverse-convolutional layer is an opposite of convolutional layer (which thus having the name inverse convolutional layer), it features cross-correlating the inputs with kernels to expand the information. The obtained output is then selectively fed into the next layer by an activation function. If the input size is not dividable by the kernel size(convolving window), then it is zero-padded to match the required length.

$$\text{Conv}_{\mathbf{K}^l,b^l}(x) = \text{F}(\sum_{i=0}^{d-1} K_i^l * x_i + b^l) \tag{6.9}$$

$*$ is the cross-correlation operation, K_i^l is the layer l i^{th} kernel, x is the input, b^l is the layer l bias, and F is the selected activation function. Inverse-Convolutional layer uses the exactly the same arithmetic as convolutional layer, but the calculation application is slightly different.

6.3.3.3 Pooling Layer

Pooling layer reduces the size of the input. Usual downsize methods in spatial domain are average pooling(Equation 6.11) and max pooling(Equation 6.10)

$$x' = \arg\max_{\mathbf{x}}(\mathbf{x})) \tag{6.10}$$

$$x' = \frac{\sum \mathbf{x}}{n} \tag{6.11}$$

x' is the result, $\mathbf{x}$ is the input, and n is number of input elements.

Outputs A $m' \times n' \times d$ tensor is produced after pooling the input.

$$\begin{aligned} m' &= \frac{1+(m-r)}{s_r} \\ n' &= \frac{1+(n-c)}{s_c} \\ d &= d \end{aligned} \tag{6.12}$$

The input depth remains unchanged.

6.3.3.4 Dense Layer

In a CNN, a dense (or fully-connected) layer is typically positioned at the conclusion of the neural network, serving as a pivotal high-level decision-making component. Figure 6.10 presents an illustration depicting a fully-connected layer with 3 inputs and 2 outputs.

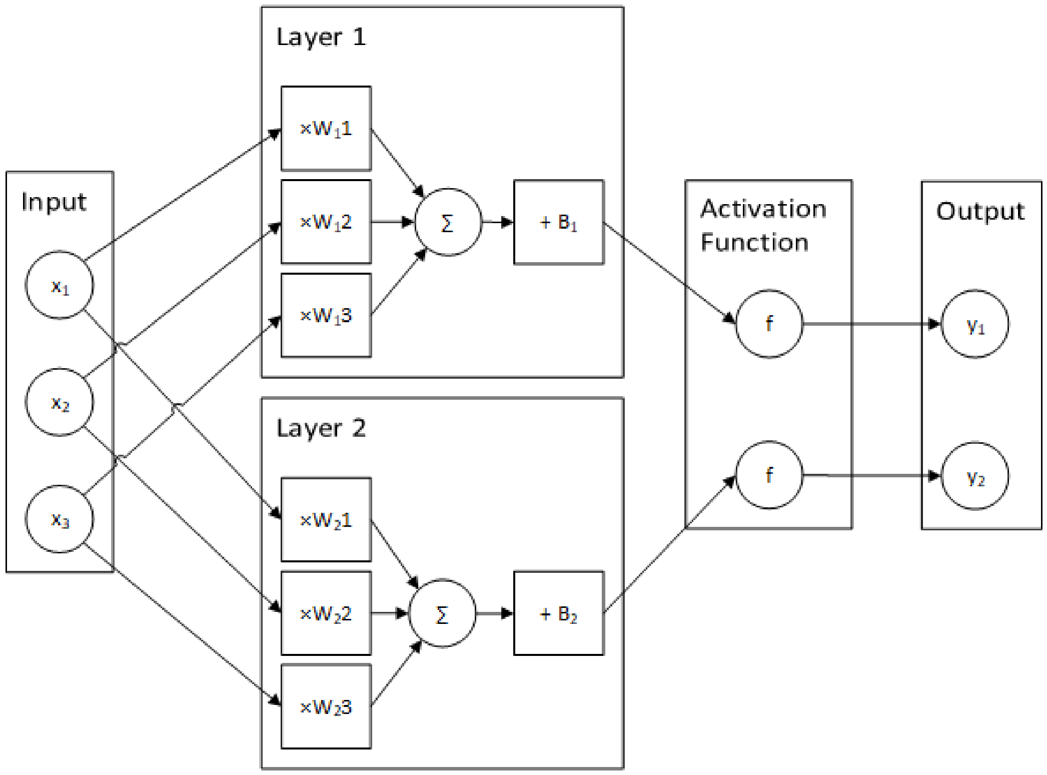


Figure 6.10: Full-connected layer illustration

$$\mathrm{FC}_{\mathbf{K}^l,b^l}(x) = \mathrm{F}^*(\sum_{i=0}^{m\times n\times d-1}(K_i^l \cdot x_i) + b^l) \tag{6.13}$$

The input tensor x is linearised first. $\mathbf{K}^l$ is the kernel vector, and b^l is the bias. They both shared the same length as linearised input vector. F* is the activation function.

Output The output vector is pre-defined by the parameter N.

6.3.3.5 Dropout Layer

Dropout Dropout is a regularization technique typically inserted in fully-connected layers during training to mitigate overfitting by inhibiting the co-adaptation of neurons [68]. During this process, a random subset of neurons within the layer is deactivated by setting their weights to zero. In the context of this research, the proportion of deactivated neurons was fixed at 50

6.3.3.6 Softmax Layer

The softmax layer is a common choice for the terminal layer in a neural network designed to address classification problems. Its purpose is to normalize the input vector into a new vector of the same length, where the sum of all elements is equal to 1. The length of the input vector to this layer is determined by the number of classes to be classified, with the output vector retaining this length. The essential function of the softmax layer is to transform the input values into a proportional representation, thereby signifying the probability associated with each class.

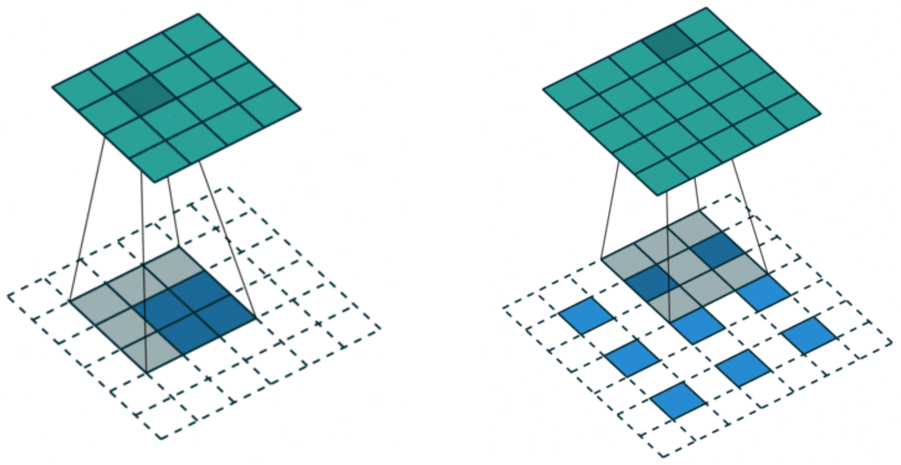
$$\text{softmax}_N(x_i) = \frac{e^{x_j}}{\sum_{j=1}^{N} e^{x_j^2}} \tag{6.14}$$

Where N is the total number of classes need to be classified.

6.3.3.7 Transposed Convolutional Layer

Transposed convolutional layer is mainly used for up-sampling the input(or a feature map). It is no different from the convolutional layer mathematically wise. Unlike the convolutional layer, transposed convolutional layer pads and insert zeros in the input, then perform the same convolutional calculation on the new sparse input with a kernel.

$$\text{Conv}^{\text{T}}_{\mathbf{K}^l,b^l}(x) = \text{F}(\sum_{i=0}^{d-1} K_i^l * x_i + b^l) \tag{6.15}$$



(a) Zero stride and no-padding (b) Centred padding and 1 stride

Figure 6.11: Transpose convolutional layer examples[54]. Green colour output, and blue colour is the input.

6.3.4 Training

NN training is a term used to describe how an NN takes processed data and, through a training process, updates the NN's internal parameters. The training process is considered complete once it reaches a predefined stopping condition, preparing the NN for predictions. Common stopping conditions include reaching a predetermined number of training epochs or achieving a specific loss threshold.

In this research, supervised mini-batch training was utilized to train the neural network. This method is favoured in scenarios with hardware constraints. Mini-batch training involves processing smaller subsets of the training data in each learning step, which not only reduces VRAM usage but also helps to prevent overfitting, a common issue in neural network training.

Figure 6.12 shows the standard mini-batch training procedure.

The training process begins by feeding a randomly selected set of training samples into the NN. The NN then generates results and computes the loss function based on these samples. If the loss value meets the stopping condition, or if the number of training epochs reaches its limit, the training concludes. If not, the NN's parameters are recalculated using back-propagation and an optimizer. After updating these parameters, the NN receives a new set of training samples, and the process repeats.

Loss Function

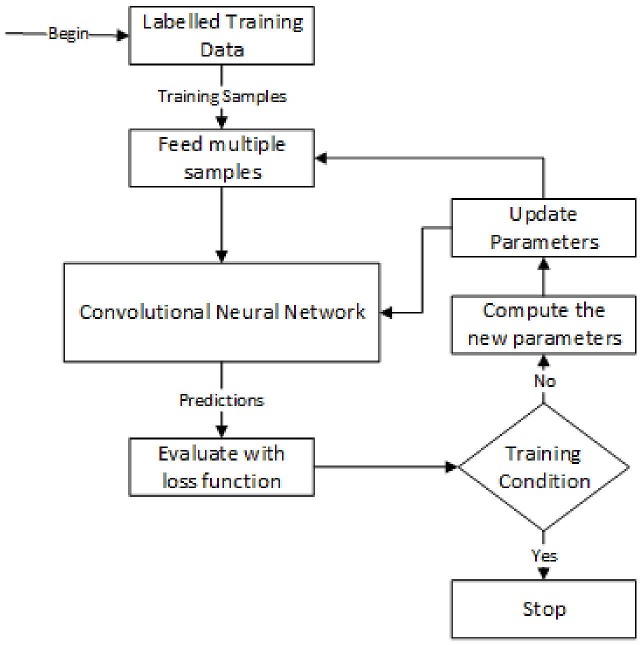


Figure 6.12: Training Process

Cross-entropy Loss The cross-entropy loss stands as a prominent choice among loss functions, particularly revered for its utility in various domains. This function imposes penalties on predictions that deviate from the ground truth, rendering it particularly advantageous within the context of classification-oriented neural networks.

$$-\sum_{c=1}^{M} y_{i,c} \log(p_{i,c}) \tag{6.16}$$

Where M is the number of classes, p is the predicted probability of input i, and y is the target class indicator of the input i and its value can only be 0 or 1.

There is a modified version of the cross-entropy function (Equation 6.17) with added weights to balance the difference across different number of training samples.

$$-\sum_{c=1}^{M} y_{i,c} \log(p_{i,c}) w_c \tag{6.17}$$

Where w_c is the weight of the class, computed by the following equation:

$$w_c = \frac{\mathbf{N}}{\mathbf{C}N_c} \tag{6.18}$$

N is the total number of training samples, N_c is the number of training samples in class c, and **C** is the total number of classes.

For mini-batch gradient descend, the loss function is the mean of all outputs.

$$\text{Loss} = -\frac{1}{n}\sum_{c=1}^{n}\sum_{c=1}^{M} y_{i,c}\log(p_{i,c}) \tag{6.19}$$

Where n is the number of samples per batch.

Mean Square Error The MSE loss calculates the average squared difference between the ground truth values and the predicted values. This loss function is frequently employed in regression tasks to gauge the disparity between predicted values and the actual targets. In this research, this loss function is utilized to assess the performance of the autoencoder.

$$MSE = \frac{\sum y_i - p_i{}^2}{M} \tag{6.20}$$

Where M is the number of classes, p is the predicted value of input i, and y is the ground truth of the input i.

Updating Parameters Parameter update can be generalize in equation 6.21 and equation 6.22.

$$K^{(l+1)} = K^{(l)} - \eta\frac{\partial L}{\partial K} \tag{6.21}$$

$$b^{(l+1)} = b^{(l)} - \eta\frac{\partial L}{\partial b} \tag{6.22}$$

K^l is the kernel of the layer l, and b^l is the bias of the layer l. η is the learning rate, set to 1 in the naive method. Computing the gradient $\frac{\partial L}{\partial K}$ and $\frac{\partial L}{\partial b}$ uses full chain rule. This involves all the layers after the layer l.

Optimization Adam optimization[91] is a recent developed popular optimization strategy in training a neural network. Adam optimization is designed to take an advantage of scaling the learning rate and movements, thus making it a superior method of optimizing the training.

$$m = \beta_1 m + (1-\beta_1)g \tag{6.23}$$

$$v = \beta_2 v + (1-\beta_2)g^2 \tag{6.24}$$

$$\hat{m} = \frac{m}{(1-\beta_1^t)} \tag{6.25}$$

$$\hat{v} = \frac{v}{(1-\beta_2^t)} \tag{6.26}$$

$$K' = K - \eta\frac{\hat{m}}{\hat{v}^{\frac{1}{2}} + \varepsilon} \tag{6.27}$$

$\hat{m}$ and $\hat{v}$ are the moving averages, g is the current computed gradient and K is the kernel. Parameters for the are: η is the learning rate, β_1, β_2 are the parameter sets for the exponential decay rate for the first and second moment estimates respectively, ε is a small value to prevent dividing by zero.

Training Condition When feeding the validation dataset(which is different from the training dataset) into the neural network and the result from the loss function is smaller than a pre-determined value, then stop the training. This process is called validation.

6.3.5 Autoencoder

An autoencoder is a type of neural network architecture that is designed to replicate its input at the output. This model primarily serves two distinct purposes in this research: dimensionality reduction and noise removal or input reconstruction.

An autoencoder can be considered a specialized variant of a convolutional neural network (CNN), distinguished by the presence of an inverse-convolutional layer. Such a structure is instrumental in transforming a lower-order input through a deconvolutional process into a higher-order output. When appropriately configured, an autoencoder has the capacity to perform unsupervised training.

There exists a substantial body of research that substantiates the effectiveness of autoencoders in the de-noising process and the restoration of partially missing image data [149][18][174]. These studies have employed diverse autoencoder architectures to achieve promising results across various applications. The exploration of autoencoders has extended beyond image processing into domains such as natural language processing (NLP).

6.3.5.1 Network Structure

The structure of an autoencoder is bifurcated into two main components: the encoder and the decoder, as depicted in Figure 6.13.

The encoder receives an input and processes it into a feature map, which essentially represents a dimensionally reduced form of the original input. The decoder, in turn, accepts this feature map and reconstructs the corresponding input from it. The synergy of these two components enables the autoencoder to carry out its specialized tasks, such as noise reduction or input reconstruction, thereby rendering it a valuable tool in various fields of research and application.

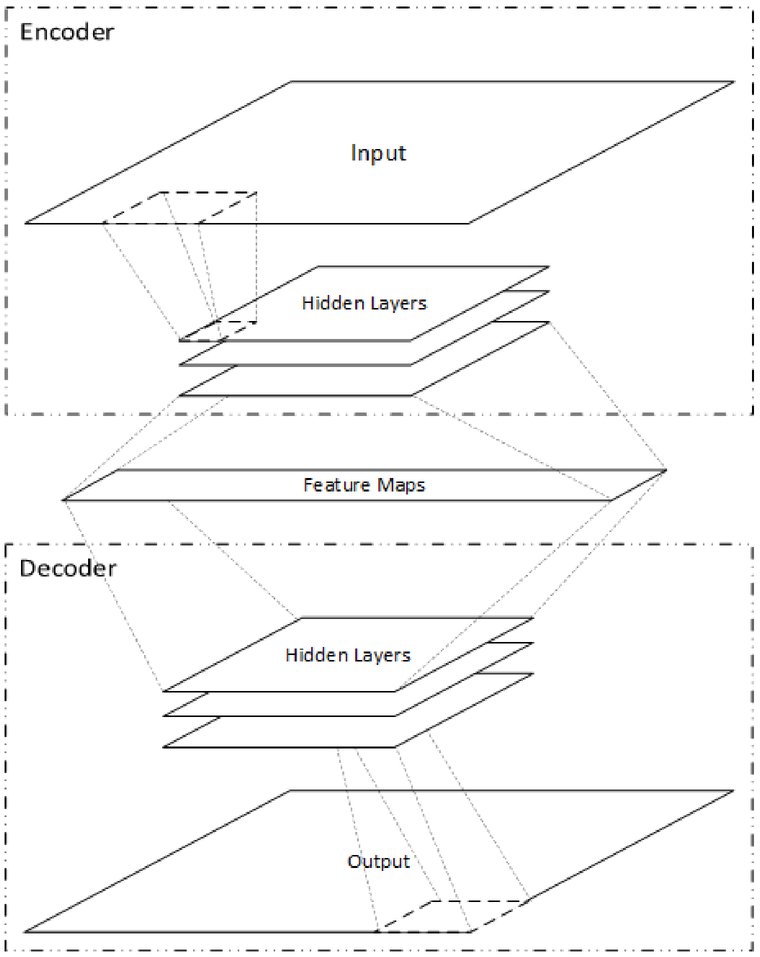


Figure 6.13: Autoencoder model

Encoder Encoder features input layer, convolutional layer(which then it is classified as convolutional autoencoder if it features this kind of layer in its structure), pooling layer, and full-connected layer.

Decoder Decoder features transposed convolutional layer (figure 6.11) and up sampling layer.

6.4 CVNN Implementation and Algorithm

6.4.1 Overview

CVNN is a complex variant of NN that accepts complex values as inputs. This NN follows a similar process to its real counterpart. The mini-batch training flow is exactly the same as its real counterpart.

6.4.2 CVNN Layers

6.4.2.1 Convolutional Layer

Like a RVNN, convolutional layer in CVCN follows the similar calculation except with modified activation function.

$$\mathrm{Conv}_{\mathbf{K},\mathbf{b}}(\mathbf{z}) = \mathrm{F}^*(\sum_{i=1}^{d} K_i * z_i + \mathbf{b}) \tag{6.28}$$

$*$ is the convolution operation, $\mathbf{z}$ is a $m \times n \times d$ complex tensor input, z_i is the corresponding $m \times n$ complex matrix. K_i^l is the corresponding size $p \times q$ kernel, and $\mathbf{b}$ is $m \times n$ bias. F^* is an activation function applied to the matrix element wise.

Convolution $*$ can be defined by summation in equation 6.29.

$$(\sum_{i=1}^{d} K_i * z_i)_{r,s} = \sum_{j=1}^{d} \sum_{k=0}^{q-1} \sum_{l=0}^{p-1} K_{j,k,l} \cdot z_{r+m,s+n,c} \tag{6.29}$$

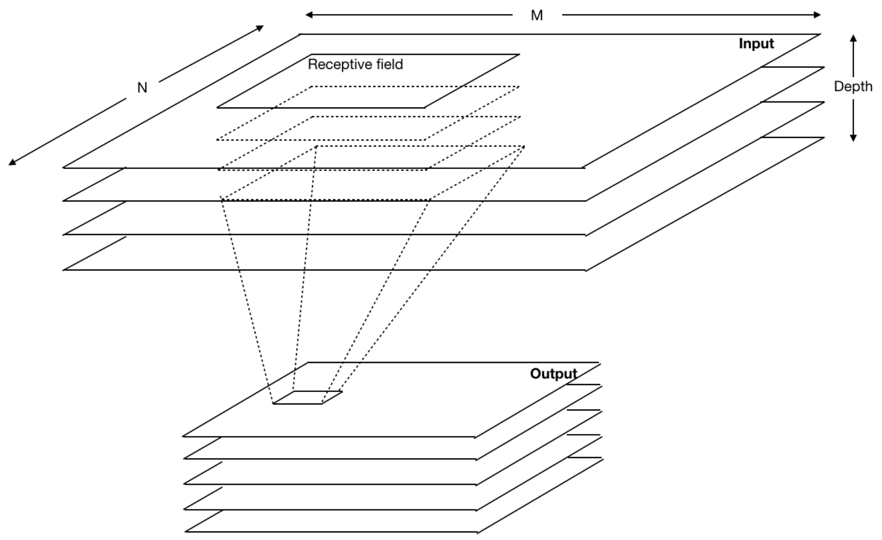


Figure 6.14: Convolutional layer filter illustration

Parameters Similiar to the real valued CNN, CVCN convolutional layer is also accepting a few parameters for the receptive field. The row of the receptive field r, the column of the receptive field c, and s_r stride is number of row element skips between each convolutional iteration. Respectively, s_c stride is number of column element skips between each convolutional iteration, and f fibre is the depth of the receptive field. p_c is the number of padding zeros added to the input.

Stride Stride is a quicker method to down-sampling and skip calculations. Skip calculations are based on the preset parameters (s_r, s_c). It is worth noting that some literatures[55][11] proposed Complex-Valued neural network without pooling layer and instead they used strides to achieve the down-sampling effect.

Padding Similarly to the real-valued counter part, the same padding strategy is used again in here.

Output The output produced by the convolutional layer will be a new tensor with size $(m' \times n' \times d')$.

$$\begin{aligned} m' &= \frac{m - r + 2 \times p}{s_r} + 1 \\ n' &= \frac{n - c + 2 \times p}{s_c} + 1 \\ d' &= d - f \end{aligned} \tag{6.30}$$

6.4.2.2 Pooling Layer

Pooling layer decreases the size of the input to reduce the amount of calculations. This strategy is commonly adapted in the CNN by known its capability of increasing the network performance. In this study, the pooling operation is specifically executed using max pooling. Figure 6.15 illustrates the pooling process, which involves moving a sliding window across the input tensor and selecting the largest value within that window. In the complex domain, the selection of the maximum value can follow either Equation 6.31 or Equation 6.32[111].

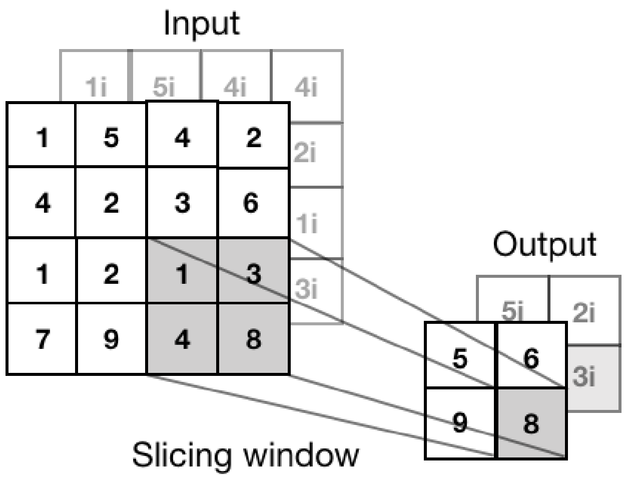


Figure 6.15: Max pooling by projecting the complex number onto the real axis (Equation 6.32)

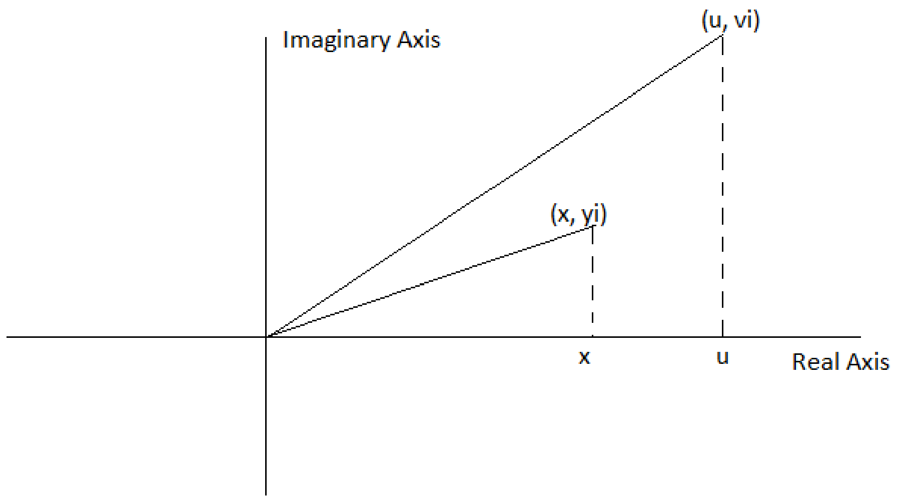


Figure 6.16: Complex number projection(Equation 6.32)

$$z' = \arg\max_{\mathbf{z}} \left(|\mathbf{Re}(\mathbf{z})| + |\mathbf{Im}(\mathbf{z})|\right) \tag{6.31}$$

$$z' = \arg\max_{\mathbf{z}} \left(\mathbf{Re}(\mathbf{z})\right) \tag{6.32}$$

Parameters Parameters for pooling layer includes strides (s_r, s_c) and window size (r, c).

Output The output of the pooling layer will be a new tensor with size $(m' \times n' \times d)$ calculated from the following equation:

$$\begin{aligned} m' &= \frac{1 + (m - r)}{s_r} \\ n' &= \frac{1 + (n - c)}{s_c} \\ d &= d \end{aligned} \tag{6.33}$$

Input depth remain unchanged.

6.4.2.3 Full-Connected Layer

Similar to the real-valued counter part, full-connected layer in CVCN is proposed in one of the last few layers for the high level decision making.

$$\mathrm{FC}_{\mathbf{K}^l,b^l}(z) = \mathrm{F}^*(\sum_{i=0}^{m\times n\times d-1}(K_i^l \cdot z_i) + b^l) \tag{6.34}$$

The input complex tensor z is linearised first. $\mathbf{K}^l$ is the kernel vector, and b^l is the bias. They both shared the same length as linearised input vector. F* is the activation function.

Output The output vector is pre-defined by the parameter N.

Dropout Similar dropout mechanic is proposed in here too.

6.4.2.4 Softmax Layer

A modified softmax function[144] is used to deal with complex values in the network. This layer will project all the complex values inputs into real axis.

$$\mathrm{softmax}_N(z_i) = \frac{e^{\mathrm{Re}(z_i)^2+\mathrm{Im}(z_i)^2}}{\sum_{j=1}^{N} e^{\mathrm{Re}(z_j)^2+\mathrm{Im}(z_j)^2}} \tag{6.35}$$

Input The input $Z, z_n \in Z$ for the layer is a length N vector, where N is total number of classifying classes.

Output The output Z' of the layer is a normalized length N vector, where $\sum_{i=1}^{N} z_i' = 1, z_i' \in Z', Z' \in \Re$.

6.4.2.5 Dropout Layers

The concept of a dropout layer was first proposed by Krizhevsky *et al.* [94]. In this layer, a random selection of p of the total elements is omitted, typically set to 50%. This approach is effective in addressing the problem of overfitting DNN.

$$\mathrm{Dropout}_r(\mathbf{z}) = \mathbf{z}' = \mathbf{B} \cdot (\frac{\mathbf{z}}{r}) \tag{6.36}$$

The input $\mathbf{z}$ is scaled by dividing by the dropout rate to compensate for the zero features resulting from the dropout process.

Input $\mathbf{z}$ is the $m \times n \times d$ input tensor, $\mathbf{B}$ is a $m \times n \times d$ binary tensor that only contains 0 or 1. $r \in [0,1]$ is the drop out rate, between 0 to 100%.

Output Output tensor $\mathbf{z}'$ shape will be $m \times n \times d$, which the same as input tensor.

6.4.2.6 Normalization Layers

A normalization layer is typically used to adjust the input tensor based on the tensor's mean and standard deviation. This adjustment can help mitigate the phenomenon known as *internal covariate shift*, where the standard distribution of each output from the previous layer shifts due to changes in the layer's parameters during the training process.

$$\mu_\beta = \frac{1}{m}\sum_{i=1}^{m} z_i \tag{6.37}$$

$$\sigma_\beta^2 = \frac{1}{m}\sum_{i=1}^{m}(z_i - \mu_\beta)^2 \tag{6.38}$$

$$\hat{z}_i = \frac{z_i - \mu_\beta}{\sqrt{\sigma_\beta^2 + \varepsilon}} \tag{6.39}$$

$$\text{Normal}_{\gamma,\beta}(z_i) = \gamma\hat{z}_i + \beta \tag{6.40}$$

Input $z_i \in \mathbf{z}$ is the $m \times n \times d$ input matrix, γ, β are the trainable parameters, and ε is the a small constant to prevent dividing by 0.

Output Output tensor $\mathbf{z}'$ is a normalized input tensor $\mathbf{z}$, where the shape is $m \times n \times d$ which is the same as input tensor.

6.4.2.7 Fourier Transform Layers

Discrete Fourier Transform (DFT) transforms discrete input from the spatial domain into the frequency domain.

1D FFT 1D FFT is common in signal processing, transforms signal from the time domain to the frequency domain.

$$X_i = \sum_{j=1}^{N} e^{-(\frac{i2\pi}{N})ij} x_j \tag{6.41}$$

2D FFT This process transform 2D inputs from spatial domain to frequency domain. Computing 2D Discrete Fourier Transform(Equation 6.42) requires $O(n^2)$, and there are Fast Fourier Transforms (FFT) algorithms those can improve the speed to $O(n \log n)$.

$$X_{i_1,i_2} = \sum_{j_1=1}^{m_x}\sum_{j_2=1}^{n_x} e^{-2i\pi(\frac{i_1 j_1 n_x + i_2 j_2 m_x}{m_x n_x})} x_{j_1,j_2} \tag{6.42}$$

Input

1D FFT $\mathbf{z}$ is the $m \times d$ input real-valued matrix.

2D FFT $\mathbf{z}$ is the $m \times n \times d$ input real-valued tensor.

Output

1D FFT $\mathbf{z}'$ is the Fourier transformed $m \times d$ complex-valued matrix.

2D FFT $\mathbf{z}'$ is the Fourier transformed $m \times n \times d$ complex-valued tensor.

6.4.2.8 Split Real Layers

One of the utility layers that output the real part of the complex-value.

$$L_{\text{Real}}(z) = \Re(z) \tag{6.43}$$

Input $\mathbf{z}$ is the $m \times n \times d$ input complex-valued tensor.

Output $\mathbf{z}'$ is the a real-valued tensor $m \times n \times d$, and it is the real part of the input $\mathbf{z}$

6.4.2.9 Split Complex Layers

Another utility layer separates the complex part from the complex value. Similar to a split real layer, this layer is useful in specialized CVNN neural networks for the purpose of splitting complex values.

$$L_{\text{Comp}}(z) = \Im(z) \tag{6.44}$$

Input $\mathbf{z}$ is the $m \times n \times d$ input complex-valued tensor.

Output $\mathbf{z}'$ is the a real-valued tensor $m \times n \times d$, and it is the complex part of the input $\mathbf{z}$

6.4.2.10 Short Layers

A short layer is used to connect two layers directly without performing any calculations. This layer is primarily utilized to mitigate vanishing gradient issues in very deep DNNs. Additionally, this layer speeds up the training process of such NNs.

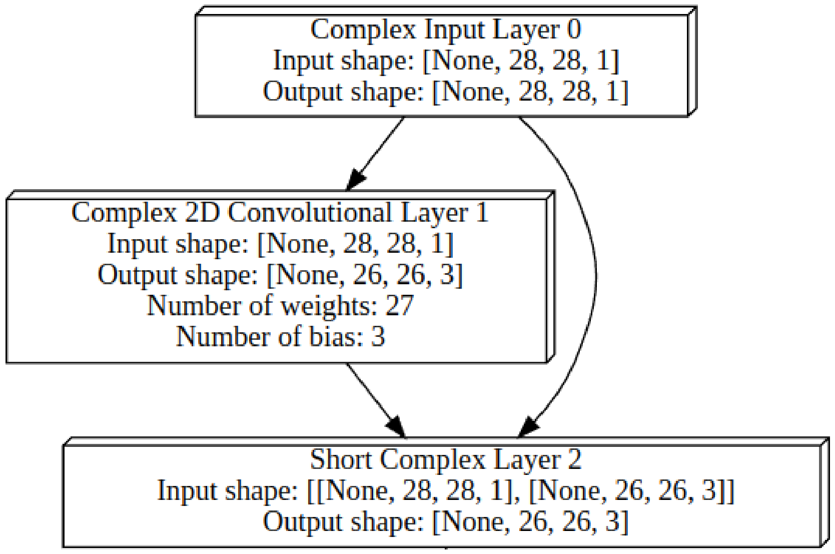

Complex Input Layer 0
Input shape: [None, 28, 28, 1]
Output shape: [None, 28, 28, 1]
Complex 2D Convolutional Layer 1
Input shape: [None, 28, 28, 1]
Output shape: [None, 26, 26, 3]
Number of weights: 27
Number of bias: 3
Short Complex Layer 2
Input shape: [[None, 28, 28, 1], [None, 26, 26, 3]]
Output shape: [None, 26, 26, 3]

Figure 6.17: Example of a non-padding short layer

Input $\mathbf{z}_l$ are the $m_l \times n_l \times d_l$ input tensors.

Output

Padding Output matrix $\mathbf{z}'$ will be $\max m \times \max n \times \max d$, taken the max dimension size amount all the input tensors.

Non-padding Output matrix $\mathbf{z}'$ will be $\min m \times \min n \times \min d$, taken the smallest dimension size amount all the input tensors, corp

6.4.2.11 Reshape Layers

An Utility layer that changes the input tensor shape to a designated one, this is useful in many complex neural network structures.

Input $\mathbf{z}$ is the $m \times n \times d$ input tensor, given new parameters to reshape the tensor.

Output $\mathbf{z}'$ in the new shape specified.

6.4.2.12 Upsampling Layer

The incorporation of an upsampling layer finds its primary application within neural networks tailored for image processing. In contrast to downsampling, this layer facilitates an expansion of the input dimensions through the utilization of interpolation techniques. As such, it is often regarded as a complementary component to the pooling layer. Within the realm of upsampling, a variety of interpolation techniques are at one's disposal. For the sake of simplicity, this research employs the bilinear interpolation method, as depicted in Equation 6.45.

$$f(x,y) \approx \frac{1}{(x_2-x_1)(y_2-y_1)} \begin{bmatrix} x_2-x & x-x_1 \end{bmatrix} \begin{bmatrix} f(x_1,y_1) & f(x_1,y_2) \\ f(x_2,y_1) & f(x_2,y_2) \end{bmatrix} \begin{bmatrix} y_2-y \\ y-y_1 \end{bmatrix} \tag{6.45}$$

Where x,y is the coordinate of the interpolated value, x_1,y_1 is the bottom left reference point, x_2,y_2 is the top right reference point, x_2,y_1 is the bottom right reference point, and x_1,y_2 is the top left reference point.

6.4.2.13 Kernel Initialization

A complex-variant Glorot initialization method[159] (equation 6.64) is used to initialize the kernel K_{ij} for CVCN.

$$\sigma_K = \frac{1}{\sqrt{n_{\text{in}} + n_{\text{out}}}} \tag{6.46}$$

$$K_{ij} = \text{Guassian}(\mu = 0.0, \sigma_K) \tag{6.47}$$

Where n_{in} is the layer input and n_{out} is the layer output, and in the form of Gaussian distribution.

6.4.2.14 Validation and Measurement

Validation is the same as CNN since the output from the CVCN is also real values, therefore real value validation measurement techniques are applied in this case. For the measurement, please refer to the CNN validation chapter.

6.4.2.15 Back Propagation Training

CR-Calculus effectively defines all operations, functions, and their derivatives for complex numbers within the neural network. As a result, backpropagation training is applicable to the network.

Training Procedure Training procedure follows the real-valued CNN counterpart, with slightly modified steps to compute complex-valued derivative.

CR-Calculus Any function(Equation 6.48) that takes complex value input is considered as a Holomorphic function only if it satisfied Cauchy-Riemann conditions(Equation 6.49 and Equation 6.50). And this means the function is analytic in the given domain, otherwise it is considered as an non-holomorphic function.

$$f(z) = u(x,y) + iv(x,y) \tag{6.48}$$

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y} \tag{6.49}$$

$$\frac{\partial v}{\partial x} = -\frac{\partial u}{\partial y} \tag{6.50}$$

Wirtinger Calculus Wirtinger Calculus is developed for generalizing derivatives in complex domain providing a method to deal with derivative of non-holomorphic function in the complex domain by mapping complex number in conjugate coordinate system(Equation 6.51).

$$\begin{bmatrix} z \\ z^* \end{bmatrix} = \begin{bmatrix} 1 & i \\ 1 & -i \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{6.51}$$

A pair of partial derivatives $\mathbb{R}$-derivative (Equation 6.52) and conjugate $\mathbb{R}$-derivative (Equation 6.53) are defined for a function $f(z,z^*)$ based on the mapping.

$$\frac{\partial f}{\partial z} = \frac{1}{2}(\frac{\partial f}{\partial x} - i\frac{\partial f}{\partial y}) \tag{6.52}$$

$$\frac{\partial f}{\partial z^*} = \frac{1}{2}(\frac{\partial f}{\partial x} + i\frac{\partial f}{\partial y}) \tag{6.53}$$

Complex Gradient Descent Algorithm First there is a real value output function f (Equation 6.54), which is the neural network function.

$$f : \mathbb{C}^n \implies \mathbb{R} \tag{6.54}$$

$$e = y' - y \tag{6.55}$$

e is the error, y is the system output, and y' is the desired output.

A real value loss function(Equation 6.56) is defined, where z is the parameter vector, and H is the Hermitian transpose operation.

$$\begin{bmatrix} -\nabla_{\mathbf{z}^*}\xi \\ -\nabla_{\mathbf{z}}\xi \end{bmatrix} = \begin{bmatrix} \mathbf{J_z} & \mathbf{J_{z^*}} \\ (\mathbf{J_{z^*}})^* & (\mathbf{J_z})^* \end{bmatrix}^H \begin{bmatrix} \mathbf{e} \\ \mathbf{e}^* \end{bmatrix} \tag{6.56}$$

The complex gradient (Equation 6.56) and its complex conjugate are defined in Wirtinger Calculus, e is the error (Equation 6.55), $\mathbf{z}$ is the parameter vector, $\mathbf{J_z}$ is the Jacobian with respect to $\mathbf{z}$, and $\mathbf{J_{z^*}}$ is the Jacobian with respect to $\mathbf{z}^*$.

$$\mathbf{J_z} = \frac{\partial y}{\partial \mathbf{z}^T} \tag{6.57}$$

$$\mathbf{J_z^*} = \frac{\partial y}{\partial \mathbf{z}^H} \tag{6.58}$$

The bias, kernel update, and Jacobian calculation are performed in accordance with the design of the neural network. The general rules for calculating the update values of the bias $\mathbf{b}$ (as shown in Equation 6.57) and the kernel $\mathbf{K}$ (as shown in Equation 6.58) are as follows:

$$\Delta\mathbf{b}_{\text{layer}} = \mu(\mathbf{J}^H_{\text{layer}_\mathbf{b}}\mathbf{e} + (\mathbf{J}^H_{\text{layer}_{\mathbf{b}^*}}\mathbf{e})^*) \tag{6.59}$$

$$\Delta\mathbf{K}_{\text{layer}} = (\delta_{layer}\mathbf{b})\mathbf{v}^H_{\text{previous layer output}} \tag{6.60}$$

Where μ is the learning rate.

Since the neural network is sharing the kernel and bias, therefore updating them are slightly different from conventional neural networks. Each bias and kernel must be treated independently, and calculate the update value (Equation 6.59 and Equation 6.60), then the actual update is the summation (Equation 6.61 and 6.62) of all update values where the kernel or bias appeared.

$$\mathbf{b} = \sum \Delta \mathbf{b} \tag{6.61}$$

$$\mathbf{K} = \sum \Delta \mathbf{K} \tag{6.62}$$

6.4.3 Kernel Initialization

A complex-variant Glorot initialization method[159] (equation 6.64) is used to initialize the weight w_{ij} for CVNN.

$$\sigma_w = \frac{1}{\sqrt{n_{\text{in}} + n_{\text{out}}}} \tag{6.63}$$

$$w_{ij} = \text{Guassian}(\mu = 0.0, \sigma_w) \tag{6.64}$$

Where n_{in} is the layer input and n_{out} is the layer output, and in the form of Gaussian distribution.

6.4.4 Input Normalization

Usually, inputs require normalization to achieve the best result. In this proposal, zero-mean normalization was used and therefore, after normalizing the input, the average becomes zero.

$$z' = \frac{(z - \hat{Z})}{\sigma_Z} \tag{6.65}$$

Normalization (Equation 6.65) is applied to all the inputs Z, with each input calculates with means $\hat{Z}$ and standard derivation σ_Z individually. Z is the 2D Fourier transformed image inputs.

6.4.5 Forward-propagation

Section 6.4.2 lays out the equations for each individual layer in the network and specifies the behavior of each layer. These equations are systematically arranged to simplify the calculation of the final output. Refer to the illustrative figure shown in 6.18, which presents a toy example to show the concept of layer-wise cascading.

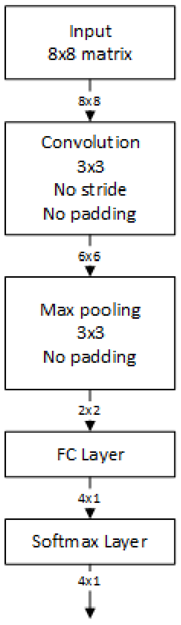


Figure 6.18: A toy example of convolutional neural network

In the given example, with input $\mathbf{X}$, each layer is expressed as follows, noting that the convolutional layer is divided into two layers using the unrolling method. The full forward propagation calculation of the neural network is shown from Equation 6.66 to Equation 6.70. New layers can be easily added to the neural network by simply cascading layer equations into the appropriate spot.

$$\mathbf{Y}_{\text{Conv}}^{\text{step}}(\mathbf{X}) = \sum x \cdot k_{\text{conv}} \tag{6.66}$$

$$\mathbf{Y}_{\text{Conv}}(\mathbf{Y}_{\text{Conv}}^{\text{step}}) = \phi(\sum y_{\text{Conv}}^{\text{step}} + b_{\text{conv}}) \tag{6.67}$$

$$\mathbf{Y}_{\text{Max}}(\mathbf{Y}_{\text{Conv}}) = \sum(y_{\text{Conv}} \cdot f_{\text{max}}) \tag{6.68}$$

$$\mathbf{Y}_{\text{FC}}(\mathbf{Y}_{\text{Max}}) = \sum(y_{\text{Max}} \cdot k_{\text{FC}}) + b_{\text{FC}} \tag{6.69}$$

$$\mathbf{Y}(\mathbf{Y}_{\text{FC}}) = \frac{e^{y_{\text{FC}}}}{\sum e^{y_{\text{FC}}^2}} \tag{6.70}$$

Given another example with a two layers complex-valued classifier $f_{\text{CVNN}}(z) : \mathbb{C} \to \mathbb{R}$:

$$f_{\text{CVNN}}(z) = \phi_{\text{SM}}(\sum W_2 \phi_{\text{Activation}}(\sum W_1 z + b_1) + b_2) \tag{6.71}$$

$\phi_{\text{Activation}}$ is a holomorphic activation function, ϕ_{SM} is a complex valued softmax function. W_1, W_2 is the kernel of respect layers, and b_1, b_2 is the bias of the respect layers.

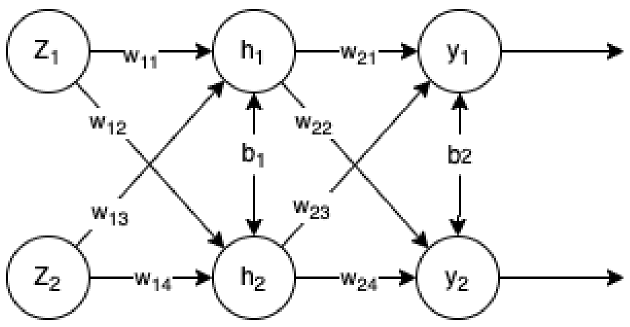


Figure 6.19: A simple example of 2 hidden layers complex-valued classifier

Using Figure 6.19 NN breakdown, $W_1 = [w_{11}, w_{12}, w_{13}, w_{14}]$ and $W_2 = [w_{21}, w_{22}, w_{23}, w_{24}]$ is formulated.

$$h_1 = \phi_{\text{Activation}}(w_{11}Z_1 + w_{13}Z_2 + b_1) \tag{6.72}$$

$$h_2 = \phi_{\text{Activation}}(w_{12}Z_1 + w_{14}Z_2 + b_1) \tag{6.73}$$

$$y_1 = \phi_{\text{SM}}(w_{21}h_1 + w_{23}h_2 + b_2) \tag{6.74}$$

$$y_2 = \phi_{\text{SM}}(w_{22}h_1 + w_{24}h_2 + b_2) \tag{6.75}$$

Thus forward propagation of the neural network has defined.

6.4.6 Error Function

The selection of the Mean Square Error (MSE), denoted by Equation 6.76, was employed as the error function for the autoencoder architecture. This particular function holds considerable prominence, being extensively utilized to gauge the performance benchmark of the autoencoder model.

$$\textbf{MSE} = \frac{\sum_{i=1}^{n}(x_i - t_i^p)^2}{n} \tag{6.76}$$

Where x_i is the predicted value, and t_i is the target value, n is the total number of predictions.

6.4.7 Loss Function

MSE Loss Function The Mean Square Error (MSE), represented by Equation 6.77, constitutes a symmetric quadratic function centered around the target value. This loss function finds common application in regression tasks, serving as a pivotal metric for assessing the accuracy of predictive models.

$$\textbf{MSE} = \frac{\sum_{i=1}^{n}(z_i - t_i^p)^2}{n} \tag{6.77}$$

Cross-entropy Cross-entropy loss is the most popular method measuring the performance of single class classification model due to its navery

$$-\sum_{c=1}^{M} y_{i,c}\log(p_{i,c}+C) \tag{6.78}$$

Where M is the number of classes, p is the predicted probability of input i, and y is the target class indicator of the input i and its value can only be 0 or 1. C is a close to zero constant (In my case $C = 0.1 \times 10^{-6}$) in place to prevent computing $\log(0)$.

6.4.8 Hybrid Neural Network

The Hybrid Neural Network (Figure 6.1) is the core component for classifying objects after receiving its LiDAR clusters, IR camera, and camera data. Each sensor's data is prepared through a different channel and fed into a distinct part of the neural network. The entire structure of the network comprises an autoencoder for LiDAR data processing, a CNN for LiDAR classification, a CVNN for camera classification, and a CNN for the IR camera classifier. All predictions from each classifier are then fed into a voter for the final prediction.

6.4.8.1 Autoencoder

The acquisition of LiDAR scans on objects often yields incomplete data due to various contributing factors, including the object's distance, surface reflectivity, and potential obstructions. However, a promising avenue for addressing this challenge involves the utilization of autoencoders in conjunction with curated training samples. This strategy presents the potential to partially or even comprehensively restore the integrity of LiDAR clusters and rectify gaps within said clusters. This restorative approach bears the capacity to markedly refine object detection capabilities by effectively compensating for the absent or erroneous data points that result from the inherent constraints of the LiDAR scanning process. The employed autoencoder architecture consists of two convolutional layers in the encoding phase, and two inverse convolutional layers along with a single convolutional layer in the decoding phase (refer to Figure 6.20, Table 6.1). This configuration encompasses a total of 61,761 trainable parameters.

The projection of all LiDAR data as binary range images onto a 32×32 grid is a fundamental step in the process. In this representation, each slot within the grid corresponds to a pixel, which is assigned a value of 1 if data exists within the corresponding LiDAR slot, and 0 otherwise. However, directly mapping the maximal value of 1 to display intensity on a computer screen is not conducive to effective visualization. Therefore, during the visualization phase, all pixel values are multiplied by 255 and converted to integers, ensuring proper representation of the monochromatic image for display purposes. Given the relatively simple input data, the neural network architecture does not necessitate a deep autoencoder structure to proficiently learn the side view configuration of the vehicle and accurately predict its original form.

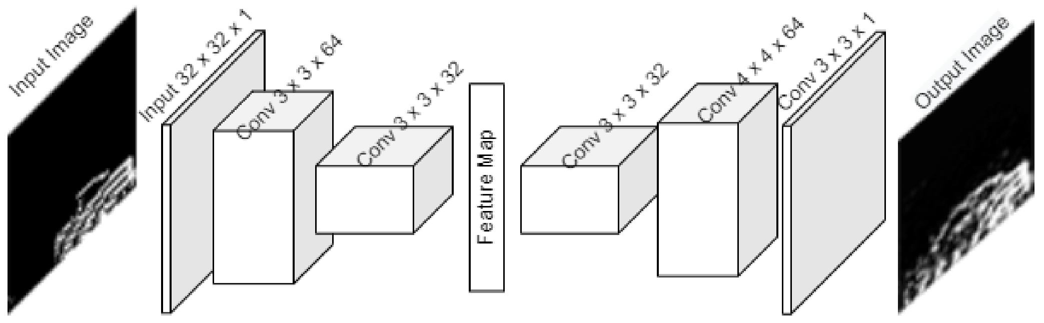


Figure 6.20: LiDAR autoencoder

	Input Shape	Kernel Window	# of Filters	Strides	# of Kernels	# of Bias	Activation Function
Conv1	[32,32,1]	[3,3]	64	[2,2]	576	64	ReLU
Conv2	[15,15,64]	[3, 3]	32	[2,2]	18432	32	ReLU
InvConv1	[7,7,32]	[3,3]	32	[2,2]	9216	32	ReLU
InvConv2	[15,15,32]	[4,4]	64	[2,2]	32768	64	ReLU
Conv3	[32,32,64]	[3,3]	1	[1,1]	576	1	Sigmoid

Table 6.1: Autoencoder Layer Configuration

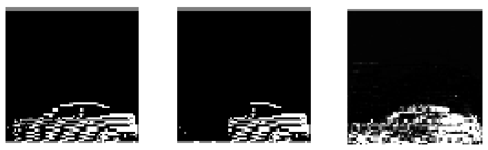


Figure 6.21: Autoencoder example: Left: Original LiDAR scan; Middle: artificially distorted scan; Right: Autoencoder fixed scan.

6.4.8.2 LiDAR Classifier

Previous research conducted jointly with Luo has studied and demonstrated that CNN is robust and effective in recognizing vehicle classes using simple LiDAR projection data. However, due to the sparse nature of LiDAR scans, it is unlikely that LiDAR can be useful in recognizing and classifying relatively small objects like pedestrians or cyclists at a distance. Thus, in this research, the LiDAR sensor is mainly used to search for objects and vehicle classifications. All other subjects such as pedestrians, cyclists will be labelled as NaV (Not a Vehicle). In continuation of the previous research[108], several modifications have been made to achieve better performance:

- The previous research did not fully utilize dropout and other training techniques, and therefore the training results often showed signs of overfitting (evidenced by practically high recall rates on the Van and SUV classes, even though both classes are arguably very similar in spatial appearance).
- The previous research neither balanced the training classes nor added weights to the loss function to help combat the high single class overfitting situation. In this research, the training was balanced by adding weights according to the number of samples during the loss calculation (equation).
- The input size was reduced for better performance. As previously input size of 128×128 is not necessary for the simplified classification.

The neural network structure utilized in this research is a conventional CNN configuration, incorporating filters for feature extraction, and a dense layer for classification based on the extracted features. Figure 6.22 illustrates the neural network proposed in this study. Autoencoder utilize MSE(Equation 6.20) loss function during the training.

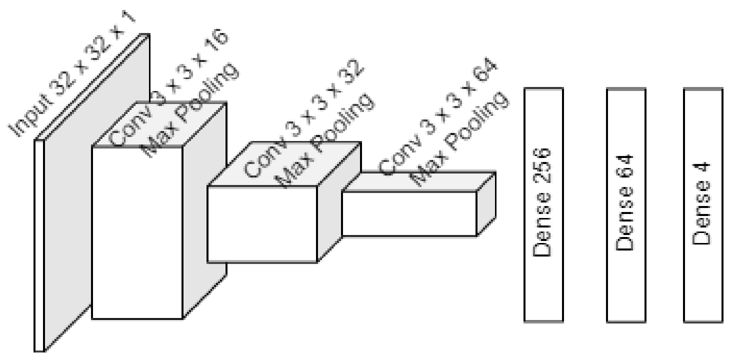


Figure 6.22: LiDAR object classifier

	Input Shape	Kernel Window	# of Filters	Strides	# of Kernels	# of Bias	Activation Function
Conv1	[32,32,1]	[3,3]	64	[2,2]	576	64	ReLU
Conv2	[15,15,64]	[3, 3]	32	[2,2]	18432	32	ReLU
InvConv1	[7,7,32]	[3,3]	32	[2,2]	9216	32	ReLU
InvConv2	[15,15,32]	[4,4]	64	[2,2]	32768	64	ReLU
Conv3	[32,32,64]	[3,3]	1	[1,1]	576	1	Sigmoid

Table 6.2: LiDAR Classifier Layer Configuration

NaV Object Classification In previous research[108], Luo utilized a specific class to denote the 'Not a Vehicle' (NaV) category, which comprised samples of non-vehicle items. However, despite this method, both recall and precision were relatively low. Luo highlighted in his thesis that these studies lacked sufficient NaV class samples for the neural network to effectively recognize this category. Training a NaV class with adequate samples can be costly. Since prior research primarily focused on highway driving, the NaV class likely included items like bushes and other objects found near highways. In contrast, this thesis encompasses research in both highway and urban driving environments, introducing a more complex array of objects such as road advertisement signs, street signs, fire hydrants, and shopping carts (noted in one parking lot dataset). This increased complexity alone significantly challenges the creation of dedicated NaV classes.

Thus, in this thesis, the problem is solved by checking the output of the neural network. Since the prediction is normalized with either the sigmoid function (for binary classification) or the softmax function (for multi-class classification), the result often comes close to a probabilistic prediction. A hard threshold can be implemented to determine if the prediction for a certain class is confident enough to be classified as that class. For example, if the prediction result is [0.3,0.25,0.45] after softmax activation across three labelled classes, but the threshold value for a successful prediction is 0.8, then this object can be classified as neither of the classes in the classifier, because class 3, which has a prediction of 0.45, is below the threshold value of 0.8.This approach effectively addresses the issue of insufficient NaV objects in the training dataset.

6.4.8.3 Camera ROI Classifier

For experimental purposes, a camera classifier was implemented in both CVNN and RVNN. The RVNN was implemented using the Keras package, while the CVNN version was implemented from scratch. For experimental comparison, an RVNN version of the NN was implemented using the same package as that used by CVNN, thus eliminating differences in programming efficiency and optimization.

RVNN Version This version of the neural network was implemented in TensorFlow using the Keras package, making it lightweight and portable across different machines running it. Additionally, since the package is officially maintained, the code is well-optimized, and the trained weights are easily portable for use. The network is composed of 7 main layers with a total of 1,777,829 trainable parameters. The first 3 layer blocks consist of convolutional layers with pooling layers for feature extraction, while the last 4 layer blocks are dense layers with dropout layers for classification purposes. All layers utilize the ReLU activation function, except for the last layer, which uses the softmax activation function for class prediction. The version implemented from scratch with CVNN packages has the exact same layout as the Keras version for experimental purposes. Dropout layers have been configured to have a 20% dropout ratio. Inputs can either be greyscale or colour. Using greyscale will reduce the channel to 1 and greatly reduce the number of trainable parameters, but during the experiment, it was found that greyscale performs slightly worse than the coloured version. Thus, this section will use the coloured version for the write-ups.

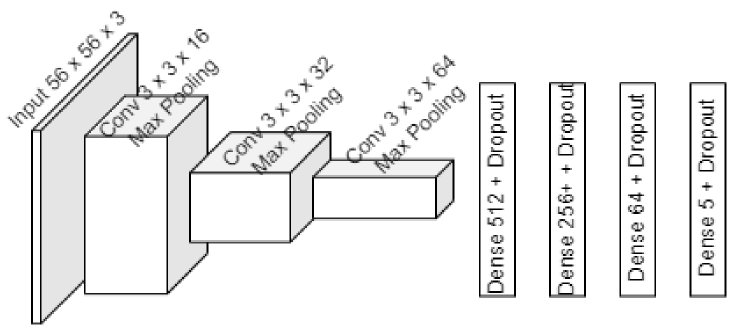


Figure 6.23: Camera ROI classifier

	Input Shape	Kernel Window	# of Filters	Strides	# of Kernels	# of Bias	Activation Function
Conv1+Maxpool	[56,56,3]	[3,3]	16	[1,1]	432	16	ReLU
Conv2+Maxpool	[28,28,16]	[3, 3]	32	[1,1]	4608	32	ReLU
Conv3+Maxpool	[14,14,32]	[3,3]	64	[1,1]	18432	64	ReLU
Dense1+Flatten	3136	-	512	-	1605632	512	ReLU
Dense2+Dropout	512	-	256	-	131072	256	ReLU
Dense 3+Dropout	256	-	64	-	16384	64	ReLU
Dense 4	64	-	5		320	5	Softmax

Table 6.3: Image classifier layer configuration

CVNN Version The CVNN version is implemented from scratch and utilizes the TensorFlow package with Nvidia CUDA package. The CVNN version has exactly the same layer setup as its RVNN counterpart, except that the activation functions are modified to be complex-valued. Due to the identical network setup, the number of trainable parameters is exactly the same, except that these parameters are complex-number trainables.

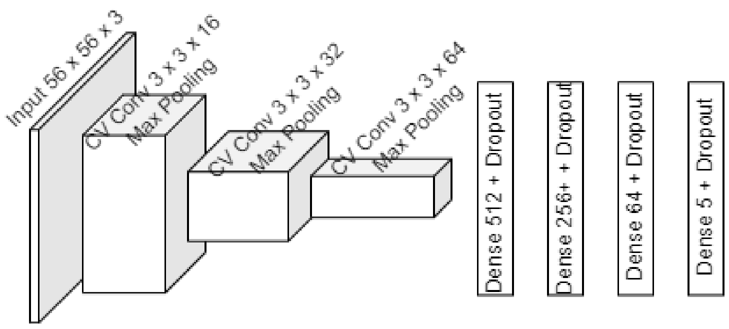


Figure 6.24: Camera ROI classifier

	Input Shape	Kernel Window	# of Filters	Strides	# of Kernels	# of Bias	Activation Function
Conv1+Maxpool	[56,56,3]	[3,3]	16	[1,1]	432	16	ReLU
Conv2+Maxpool	[28,28,16]	[3, 3]	32	[1,1]	4608	32	ReLU
Conv3+Maxpool	[14,14,32]	[3,3]	64	[1,1]	18432	64	ReLU
Dense1+Flatten	3136	-	512	-	1605632	512	ReLU
Dense2+Dropout	512	-	256	-	131072	256	ReLU
Dense 3+Dropout	256	-	64	-	16384	64	ReLU
Dense 4	64	-	5		320	5	Complex Softmax

Table 6.4: Complex-valued image classifier layer configuration

6.4.8.4 IR Camera ROI Classifier

The IR camera ROI classifier has a fewer number of trainable parameters, even though the network structure remains similar with a 7-layer block setup. Since the IR image is monochrome, it only has 1 channel for the input, resulting in a resized input size of $56 \times 56 \times 1$. The number of filters was reduced by half, as determined by the experiment, showing that IR camera data does not benefit much from having more trainable parameters. The dropout ratio is 20% as well, the same as the camera image CNN counterpart.

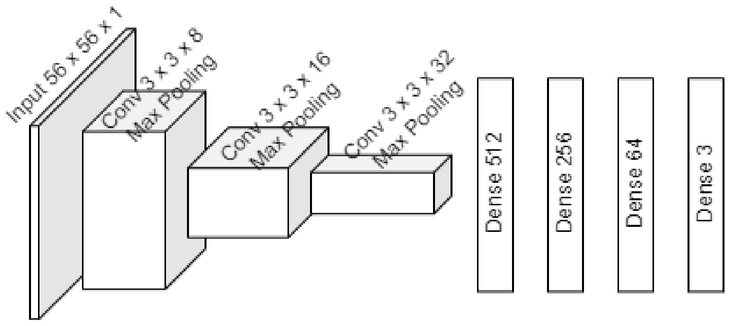


Figure 6.25: IR camera ROI classifier

	Input Shape	Kernel Window	# of Filters	Strides	# of Kernels	# of Bias	Activation Function
Conv1+Maxpool	[56,56,1]	[3,3]	8	[1,1]	72	8	ReLU
Conv2+Maxpool	[28,28,16]	[3, 3]	16	[1,1]	1152	16	ReLU
Conv3+Maxpool	[14,14,32]	[3,3]	32	[1,1]	4608	32	ReLU
Dense1+Flatten	1568	-	512	-	802816	512	ReLU
Dense2+Dropout	512	-	256	-	131072	256	ReLU
Dense 3+Dropout	256	-	64	-	16384	64	ReLU
Dense 4	64	-	3		192	3	Softmax

Table 6.5: IR image classifier layer configuration

6.4.8.5 Voter

Since all three sensors output different classes, the voter design is no trivial task here. First, LiDAR is effective at detecting whether an object is a vehicle or not, and it is robust in most situations (unless the object is really far away from the sensor). The IR camera, on the other hand, can accurately detect pedestrians regardless of lighting conditions, so if it detects a pedestrian, then it is very confident that the object is a pedestrian. The camera in this context is limited by many factors, such as lighting conditions and glares, all of which can affect classification and cause false detection. However, out of all three equipped sensors, the camera is the one that can distinguish subclasses (note that the IR camera's car class includes all kinds of vehicles, as sometimes it is nearly impossible to detect the vehicle type from the IR camera sensor alone). Figure 6.26 shows the relationship between each classifier class prediction and the final voter class prediction. LiDAR has a hyper class which contains ”Car”, ”Van”, ”Truck”, and ”NaO”. LiDAR's initial classifier does not contain ”NaO”, thus this hyper class is computed in Equation 6.79. When p_{NaO} is not zero, all other hyper LiDAR

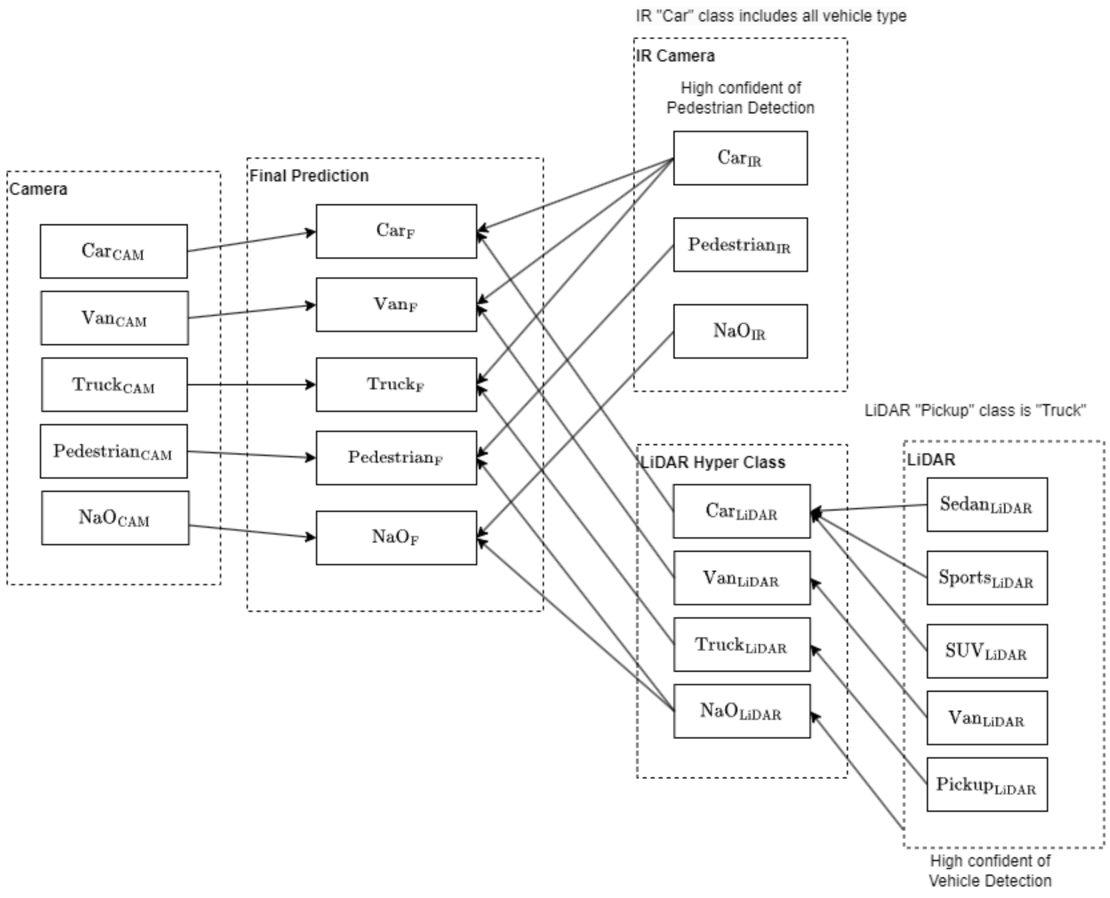


Figure 6.26: Voter class relation diagram

classes will have a prediction value of 0. The voter design was initially done through training, but the result was disastrous due to sensor reliability changes when scenario conditions change, thus neither of the sensors can be reliable in all situations without prior knowledge about the driving scenario, and not enough special weather condition data to train for a general model. In this work, those weights w are manually assigned based on trial and error. The final prediction $\mathbf{P}^{\mathrm{F}}$ can be computed from Equation 6.86 by summing all computed prediction of each individual classes.

$$p_{\mathrm{NaO}}^{\mathrm{LiDAR}} = \begin{cases} 1 & \text{if } \max(\mathbf{P}_{\mathrm{LiDAR})}) < th_{\mathrm{LiDAR}} \\ 0 & \text{if } \max(\mathbf{P}_{\mathrm{LiDAR}}) \geq th_{\mathrm{LiDAR}} \end{cases} \tag{6.79}$$

Where $\mathbf{P}_{\mathrm{LiDAR}}^{\mathrm{LiDAR}}$ is the prediction of LiDAR classifier, and th_{LiDAR} is the threshold value to determine if the object is not a value.

$$p_{\text{Car}}^{\text{LiDAR}} = p_{\text{Sedan}}^{\text{LiDAR}} + p_{\text{Sport}}^{\text{LiDAR}} + p_{\text{SUV}}^{\text{LiDAR}} \tag{6.80}$$

$p_{\text{Car}}^{\text{LiDAR}}$ is equal to the summation of the corresponding classes. Thus the computing is like the following:

$$p_{\text{Car}}^{\text{F}} = \frac{1}{3} w_{\text{Car}}^{\text{IR}} p_{\text{Car}}^{\text{IR}} + w_{\text{Car}}^{\text{LiDAR}} p_{\text{Car}}^{\text{LiDAR}} + w_{\text{Car}}^{\text{CAM}} p_{\text{Car}}^{\text{CAM}} \tag{6.81}$$

$$p_{\text{Van}}^{\text{F}} = \frac{1}{3} w_{\text{Car}}^{\text{IR}} p_{\text{Van}}^{\text{IR}} + w_{\text{Van}}^{\text{LiDAR}} p_{\text{Van}}^{\text{LiDAR}} + w_{\text{Van}}^{\text{CAM}} p_{\text{Van}}^{\text{CAM}} \tag{6.82}$$

$$p_{\text{Truck}}^{\text{F}} = \frac{1}{3} w_{\text{Car}}^{\text{IR}} p_{\text{Truck}}^{\text{IR}} + w_{\text{Truck}}^{\text{LiDAR}} p_{\text{Truck}}^{\text{LiDAR}} + w_{\text{Truck}}^{\text{CAM}} p_{\text{Truck}}^{\text{CAM}} \tag{6.83}$$

$$p_{\text{Pedestrian}}^{\text{F}} = w_{\text{Pedestrian}}^{\text{IR}} p_{\text{Pedestrian}}^{\text{IR}} + \frac{1}{2} w_{\text{NaO}}^{\text{LiDAR}} p_{\text{Pedestrian}}^{\text{LiDAR}} + w_{\text{Pedestrian}}^{\text{CAM}} p_{\text{Pedestrian}}^{\text{CAM}} \tag{6.84}$$

$$p_{\text{NaO}}^{\text{F}} = w_{\text{NaO}}^{\text{IR}} p_{\text{NaO}}^{\text{IR}} + \frac{1}{2} w_{\text{NaO}}^{\text{LiDAR}} p_{\text{NaO}}^{\text{LiDAR}} + w_{\text{NaO}}^{\text{CAM}} p_{\text{NaO}}^{\text{CAM}} \tag{6.85}$$

$$\mathbf{P}^{\text{F}} = \text{SOFTMAX}([p_{\text{Car}}^{\text{F}}, p_{\text{Van}}^{\text{F}}, p_{\text{Truck}}^{\text{F}}, p_{\text{Pedestrian}}^{\text{F}}, p_{\text{NaO}}^{\text{F}}]) \tag{6.86}$$

6.4.9 Training Dataset

6.4.9.1 Data Augmentation

In this research, data augmentation was implemented to expand the training set. Data augmentation leverages the fact that a CNN can handle inputs with distortions, rescaling, changes in orientation, or flipping. By artificially distorting, flipping, or rotating the training samples, these transformations can be utilized during the training process to help the CNN generalize better.

A list of augmentation was implemented in the process:

- Rotation: Rotate the image alone the centre.
- Shearing: deform the image alone one side.
- Zoom: Zooming in the image.
- Shifting: shifting the image pixel.

- Adding Noise: artificially adding Gaussian noise onto the training sample.

	LiDAR	Camera	IR Camera
Rotation	$\pm 10\%$ degree	$\pm 25\%$ degree	$\pm 25\%$ degree
Shearing	20% shearing angle	20% shearing angle	20% shearing angle
Zoom	$< 10\%$ zoom in	$< 10\%$ zoom in	$< 10\%$ zoom in
Horizontal Shifting	$\pm 5\%$ horizontal pixels	$\pm 10\%$ horizontal pixels	$\pm 10\%$ horizontal pixels
Vertical Shifting	$\pm 5\%$ vertical pixels	$\pm 5\%$ vertical pixels	$\pm 5\%$ vertical pixels
Noise	White noise	White noise	White noise

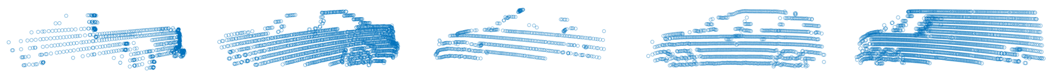
Table 6.6: Augmentation applied to each sensor classification training data

6.4.9.2 LiDAR Autoencoder Training

Autoencoders employ unsupervised training, which simplifies the preparation process as it does not involve labeling. A total of 2343 samples were prepared, encompassing 5 different vehicle classes, as illustrated in Table 6.7. Training was conducted in batch with size of 32, and training/validation split is set to 30%.

	Pickup	Sedan	Sports	SUV	Van	Total
Sample Counts	223	660	410	769	281	2343

Table 6.7: LiDAR training samples



(a) Pickup class (b) Sedan class (c) Sports class (d) SUV class (e) SUV class

Figure 6.27: Example of labelled LiDAR inputs.

6.4.9.3 LiDAR Classifier Training

The training dataset employed for the LiDAR autoencoder is also utilized to train the LiDAR classifier. The dataset was augmented during the training preparation. The LiDAR CNN was implemented in Keras for its portability and ease of deployment in the vehicle.

During the experimental process, the effectiveness of LiDAR data was found to be less than ideal for the fine classification of objects, such as pedestrians. Moreover, it demonstrated a lack of sensitivity in distinguishing between larger and smaller vehicles. Consequently, this portion of the NN was explicitly designed for the binary classification of vehicle objects. For example, if a prediction does not match either of the classes, like an object illustrated in Figure 6.28, the neural network's output was [0.26, 0.11, 0.08, 0.23, 0.33], with the maximum prediction of 0.33 relating to the ”van” class. However, this object clearly does not fall into any vehicle class and is instead classified as a ”NaV” class. Weight balancing techniques were employed during the training of the classes to correct the imbalance between samples. Training was conducted in batch with size of 32, and training/validation split is set to 30%.

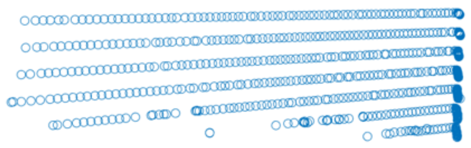


Figure 6.28: NaV example

6.4.9.4 Camera Classifier Training

To train the NN, a set of resized image inputs with labels has been prepared. To augment the training dataset, this research incorporated the KITTI semantic dataset[17] into the CMHT dataset. The original KITTI semantic dataset was labelled and intended for semantic training, but it can easily be converted into a classification training dataset using a cropping script. There are five classes currently labelled, including Not-an-Object (NaO), which means it is a detected object by LiDAR, but the class is not of concern. Normally, these are miscellaneous objects such as road signs and other random items. The reasoning behind this class is that during the experiment, it was found that even a well-trained NN has difficulty discarding objects-not-in-class due to the complexity of image training. Often, objects detected are associated with high confidence values, making it hard to discard them like LiDAR classifiers. Therefore, to combat this issue, a NaO class has been added to improve the generalization of the network. Training was conducted in batch with size of 32, and training/validation split is set to 30%.

	Car	NaO	Pedestrian	Truck	Van	Total
Sample Numbers	12214	602	2413	367	929	16525

Table 6.8: Image training sample count

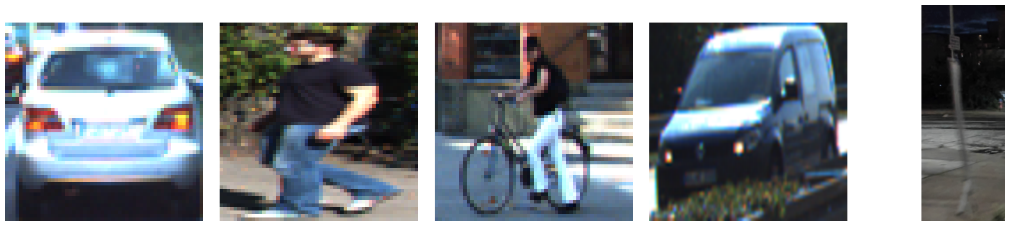

(a) Car (b) Pedestrian (c) Cyclist (d) Van (e) NaO

Figure 6.29: Example of labelled image classes

6.4.9.5 IR Camera Classifier Training

The IR camera dataset is another rare-to-find dataset. Aside from the CMHT dataset, FLIR data was also incorporated into the training samples. A weight balancing strategy was also deployed during the training. The IR ”Car” label includes sub-categories like ”Sedan”,”Van”, and ”SUV”. Training was conducted in batch with size of 32, and training/validation split is set to 30%.

	Car	Pedestrian	NaO	Total
Sample Numbers	69560	47122	1542	118224

Table 6.9: IR Image training sample count

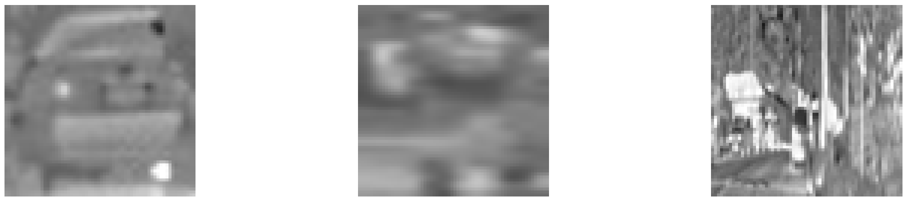

(a) Car (b) Pedestrian (c) NaO

Figure 6.30: Example of labelled IR image classes

6.5 Results

For the score marks and calculations, please refer to Appendix section B.

6.5.1 Autoencoder

Autoencoder performance can be determined by using a loss function from the validation; the smaller the loss is, the closer the reassembled object is to the original form. Testing was conducted in two ways: the first test was to simulate if the object is partially blocked by other objects, thus part of the object was randomly removed, up to 30%, of the object's points. The second test was to determine if the object is further away from the sensor, thus points on the object were randomly removed, up to 50% to simulate a sparse scan from a distance.

Due to the simplicity of the projected LiDAR images, the autoencoder captured the features really well and achieved an average loss of 0.151 across 200 testing samples, with a worst loss of 0.274, and a best loss value of 0.122 in the first test. In the second test, the average loss value is 0.129 across 200 training samples, with a worst loss of 0.235 and a best loss of 0.121. Both results are considered satisfactory.

6.5.2 Object Classification By Individual Sensors

6.5.2.1 LiDAR Classification

The LiDAR classification performs as expected; even when the neural network is downsized, the accuracy is not substantially hindered, maintaining a level of 83.20% when tested with a completely different testing dataset. When tested with randomly cropped clusters, the network prediction often results in a leading class prediction with confidence below 0.7. Therefore, the NaV detection threshold is set to 0.85. Any incoming cluster object that does not pass this leading cluster threshold will be classified as the NaV class in the output to feed into the voter.

Another significant aspect of the approach is the use of weights during training, which notably enhances the generalization of the NN. This strategy is particularly important given that the dataset is imbalanced, primarily consisting of the ”sedan” or ”SUV” classes. Without this balancing act, the trained NN could easily skew towards those two classes by mere random guessing, yet still achieve very acceptable accuracy, even with a 7:3 training to validation slice. The ”SUV” class has a fairly low recall rate due to natural difficulty to detect; from the shape, it's just a slightly smaller van, thus the ”Van” class and ”SUV” class have lower precision and recall, also many FP predictions between the two classes. This was observed during the previous studies. However, since the LiDAR classifier is primarily used for identifying the object and determining if it is a vehicle, the detailed sub-classification rate can be enhanced by employing the camera sensor. The testing dataset is different from the training data; therefore, the result is considered satisfactory.

Output\Target	Pickup	Sedan	Sports	SUV	Van
Pickup	162	1	13	2	5
Sedan	12	192	10	5	5
Sports	18	5	171	11	3
SUV	5	2	5	147	27
Van	3	0	1	35	160

Table 6.10: LiDAR classification confusion table

Class Name	Precision	1-Precision	Recall	1-Recall	f1-score
Pickup	0.8852	0.1148	0.8100	0.1900	0.8460
Sedan	0.8571	0.1429	0.9600	0.0400	0.9057
Sports	0.8221	0.1779	0.8550	0.1450	0.8382
SUV	0.7903	0.2097	0.7350	0.2650	0.7617
Van	0.8040	0.1960	0.8000	0.2000	0.8020
Accuracy	0.8320				
Misclassification Rate	0.1680				
Macro-F1	0.8307				
Weighted-F1	0.8307				

Table 6.11: LiDAR classification result

6.5.2.2 Camera Classification

RVNN Result The camera sensor stands out among the three sensors as the only one capable of visually identifying all different subclasses. However, it is also the least robust of the sensors, being sensitive to weather, lighting conditions, and other environmental factors. The subsequent section, which tests the proposed neural network against the state-of-the-art YOLO V8 detector using only a camera in dark or rainy conditions, underscores this data. Image classification is a well-studied area, and there is no doubt about the power of CNN image classifiers. With a performance of 94.32% on a mixed image dataset, the camera proves to be a strong performer, making it the ideal object classifier for general use when compared to LiDAR and IR cameras.

Output\Target	Car	NaO	Pedestrian	Truck	Van
Car	3952	15	53	102	92
NaO	10	556	32	3	1
Pedestrian	45	12	2327	0	29
Truck	8	1	0	358	0
Van	75	1	5	0	848

Table 6.12: Real-Valued Image classification confusion table

Class Name	Precision	1-Precision	Recall	1-Recall	f1-score
Car	0.9378	0.0622	0.9663	0.0337	0.9518
NaO	0.9236	0.0764	0.9504	0.0496	0.9368
Pedestrian	0.9644	0.0356	0.9628	0.0372	0.9636
Truck	0.9755	0.0245	0.7732	0.2268	0.8627
Van	0.9128	0.0872	0.8742	0.1258	0.8931
Accuracy	0.9432				
Misclassification Rate	0.0568				
Macro-F1	0.9216				
Weighted-F1	0.9426				

Table 6.13: Real-Valued Image classification result

CVNN Result The CVNN implementation, on the other hand, performs closely to another neural network but slightly worse (92.74% vs RVNN 94.32%). One hypothesis for this discrepancy is that the relationship between phase and magnitude raises difficulty during training, making it more challenging to converge. For more discussion related to CVNN and training, please refer to the next chapter of the thesis.

Output\Target	Car	NaO	Pedestrian	Truck	Van
Car	3913	11	141	53	96
NaO	10	546	45	1	0
Pedestrian	41	19	2333	0	20
Truck	19	0	3	345	0
Van	156	0	4	0	769

Table 6.14: Complex-Valued Image classification confusion table

Class Name	Precision	1-Precision	Recall	1-Recall	f1-score
Car	0.9286	0.0714	0.9454	0.0546	0.9369
NaO	0.9070	0.0930	0.9479	0.0521	0.9270
Pedestrian	0.9668	0.0332	0.9236	0.0764	0.9447
Truck	0.9401	0.0599	0.8647	0.1353	0.9008
Van	0.8278	0.1722	0.8689	0.1311	0.8479
Accuracy	0.9274				
Misclassification Rate	0.0726				
Macro-F1	0.9115				
Weighted-F1	0.9276				

Table 6.15: Complex-Valued Image classification result

6.5.2.3 IR Camera Classification

Testing with the IR classifier was performed on 1000 randomly selected instances from each class. Undoubtedly, the IR camera excels in detecting pedestrians, achieving close to 90% recall and precision, along with a 0.9045 F1 score. However, vehicle detection presents a more complex picture due to various factors. The appearance of a vehicle under IR imaging can dramatically differ depending on its state: a driving vehicle generally shows hot spots on the wheels, exhaust, and engine area, whereas an electric vehicle displays hot spots mainly on the battery compartment and tires. Parked vehicles add another layer of complexity; their temperature readings can vary greatly depending on exposure to sunlight and the length of time they have been stationary. These observations hint that the IR camera might be used to determine a vehicle's state, a potential area for future research. The myriad possible states and forms of vehicle imaging create challenges for a simple classifier in identifying car classes.

Output\Target	Car	Pedestrian	NaO
Car	754	15	157
Pedestrian	6	895	78
NaO	240	90	765

Table 6.16: IR image classification result

Class Name	Precision	1-Precision	Recall	1-Recall	f1-score
Car	0.8143	0.1857	0.7540	0.2460	0.7830
Pedestrian	0.9142	0.0858	0.8950	0.1050	0.9045
NaO	0.6986	0.3014	0.7650	0.2350	0.7303
Accuracy	0.8047				
Misclassification Rate	0.1953				
Macro-F1	0.8059				
Weighted-F1	0.8059				

Table 6.17: Image classification result

6.5.3 Full Integration and Performance Test

CMHT dataset was used to show the true strength of sensor fusion and showing how regular camera based detection can failed under challenging weather and lighting condition. In this research, a state of the art camera based detection algorithm yolov8 is put into the test again the proposed method. Dataset contains total 165 labelled frames with the following unique objects:

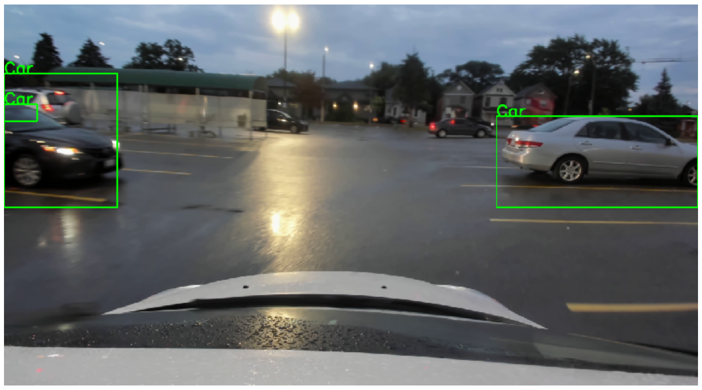


Figure 6.31: Example of the full system integration and visualization

	Scenario1 Parking Lot	Scenario2 Urban&Highway	Scenario3 Residential Area	Total
Car	182	133	71	386
Truck	11	15	5	31
Van	10	2	0	12
Pedestrian	31	15	4	50

Table 6.18: Benchmark dataset

It's worth noting that due to the proposed work, detection is conducted through the LiDAR sensor and projected onto the image. This leads to natural projection error, making the Intersection-Over-Union (IoU) (commonly used to benchmark pure image-based classifiers) score inapplicable here. The justification for this is that with sensor fusion, the LiDAR sensor can accurately provide the detected object's real-world coordinate, which is far more precise than image-to-3D-coordinate projection. As a result, the IoU loses its value in benchmarking this particular algorithm.

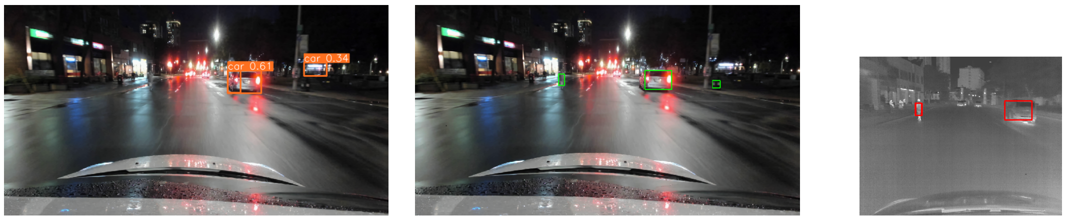
Output\Target	Car	Truck	Van	Pedestrian	NaO
Car	305	2	1	0	27
Truck	13	24	4	0	2
Van	15	0	6	0	0
Pedestrian	7	0	0	36	2
NaO	25	0	0	12	230
Detection	0.9456	0.8387	0.9167	0.9400	N/A

Table 6.19: Fusion result

Output\Target	Car	Truck	Van	Pedestrian	NaO
Car	227	2	5	0	28
Truck	16	12	3	0	3
Van	32	0	2	0	2
Pedestrian	0	0	0	21	5
NaO	12	2	0	15	560
Detection	0.7435	0.5161	0.8333	0.7200	N/A

Table 6.20: YoloV8 result

Yolo V8 YoloV8 exhibited significantly fewer detections due to its limitation as a camera-only detector. However, the proposed method's LiDAR can only generate valid clusters up to 15 meters away from the sensor, and beyond this distance, the object is likely discarded as noise during the clustering process due to too few scans presented. Thus, the proposed work cannot detect objects beyond 15 meters, whereas YoloV8 can detect objects much further away. Within a 15-meter range, the proposed work can achieve a notably high recall and detection range, particularly a 94.56% ”Car” class detection rate, and a 91.67% ”Pedestrian” detection rate, with which YoloV8 struggles in frames with bad weather or dark environments. Figure 6.32 shows an example of a YoloV8 missed detection, where the jaywalking pedestrian completely blended into the scene and became extremely difficult to capture with a camera only, and cases like this can be fatal if the vehicle was autonomously controlled. There are many other examples like this, further proves the robustness of sensor fusion during bad weathers.



(a) YolovV8 (b) Camera (c) IR Camera

Figure 6.32: Example of YoloV8 missed detection, the pedestrian to the left was not detected. But purposed vehicle perception caught the jaywalking person.

6.6 Conclusion

This chapter introduces a hybrid neural network designed to process heterogeneous sensor data inputs. By training with augmented data and individually tuning each part of the network, the system achieves greater accuracy than state-of-the-art single sensor detectors such as YoloV8, especially in dark environments or adverse weather conditions. This success has proven the effectiveness of sensor fusion in improving vehicle perception under non-ideal driving conditions. The chapter goes further to detail the implementation of real-valued neural networks, offering an in-depth examination of the layers that comprise both the CNN and autoencoder neural network used in the research. Furthermore, the implementation of CVNN is explained in this chapter, and the performance of a RVNN is compared with CVNN using a standard convolutional classifier setup and image inputs. However, due to application-specific issues, the results are inconclusive, and there is no evidence to show that one network performs better than another.

Chapter 7

Complex-Valued Neural Network Training

7.1 Introduction

The previous chapter introduced CVNN and its layer composition, while this chapter focuses on the training aspect of the CVNN. The first part of the chapter explains common complex-valued activation functions used in CVNN and details three common approaches in dealing with complex-valued functions. It also includes a small example of how to perform back-propagation to compute the gradient in CVNN mathematically.

The second part of the chapter concentrates on exploring different optimizers for the CVNN during training, discussing the findings, and introducing a novel first-order gradient-based optimizer called cm-reSVSF. This optimizer aims to train CVNN both quickly and accurately. Through experimentation, it was demonstrated that the cm-reSVSF optimizer performs exceptionally well in training CVNN in terms of both accuracy and speed.

7.2 Implementation

7.2.1 Complex Activation Function

In the complex domain, an activation function necessitates additional characteristics. A bounded entire function refers to a complex function that is both bounded and holomorphic at all finite points across the entire complex plane. Liouville's theorem asserts that a complex function must be constant throughout the entire complex plane to qualify as a bounded entire function. This constancy contradicts the requirement for the function to be monotonic. Theoretically, there is no bounded entire function suitable for use as a complex activation function. Various research studies are exploring alternative solutions, primarily focusing on four different approaches.

Using a split function is a common approach. This method splits the complex number into real and imaginary counterparts, then performs activation based on the computed results. It avoids the unboundedness of the complex function, but the split function itself could never satisfy the Cauchy-Riemann equations. For example, separated hyperbolic tangent function (Figure 7.1, equation 7.1) mentioned in the book[69], split-Sigmoid function (Figure7.5, Equation 7.3).

$$\mathrm{f}(z) = \tanh(\mathbf{Re}[z]) + i\tanh(\mathbf{Im}[z]) \tag{7.1}$$

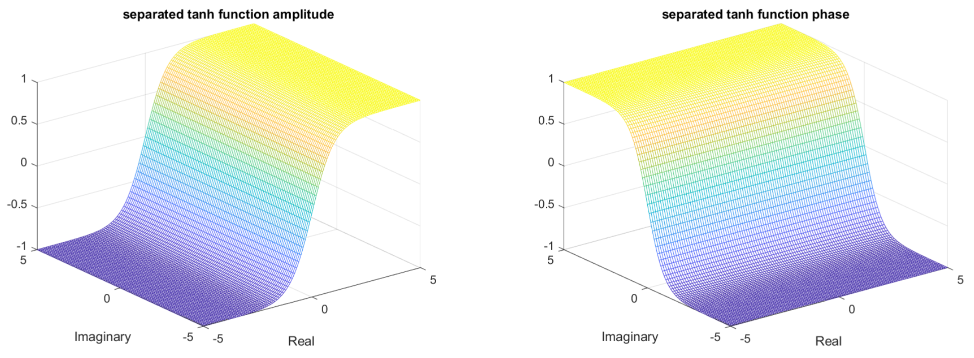


Figure 7.1: Amplitude and phase of separated hyperbolic tan function.

Another approach suggested using elementary transcendental functions[90] as a complex activation function. Figure 7.2 shows the plot of the hyperbolic tan function; notably, the function itself is semi-flat across the imaginary plane, yet retains the desired ”monotonic” shape across the real domain. Figure 7.3 shows the plot of the tan function, similar to the hyperbolic counterpart, with an expected flip in phase and amplitude. Despite the periodic singularities in the function, the research[90] suggests it still achieved better performance than split functions.

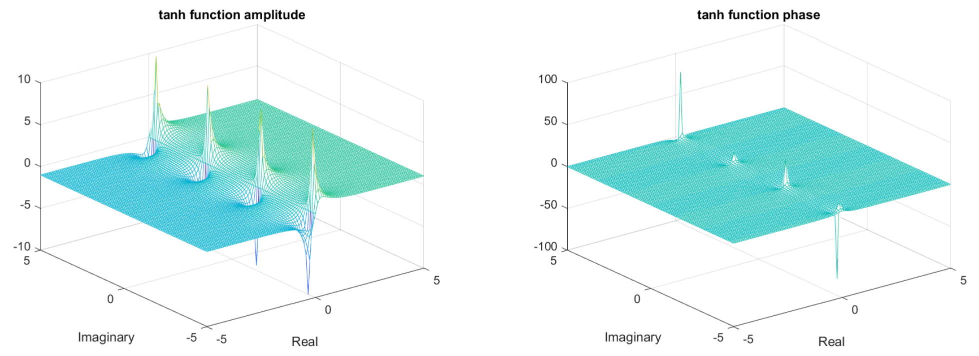


Figure 7.2: Amplitude and phase of hyperbolic tan function.

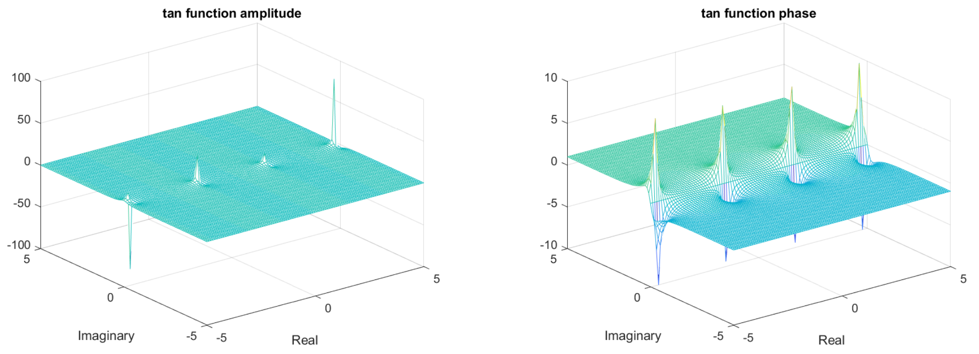


Figure 7.3: Amplitude and phase of tan function.

A third approach is using holomorphic complex functions, such as the sigmoid function (Figure 7.4, equation 7.2) [40]. Regardless, this approach did not solve the issue with singularities.

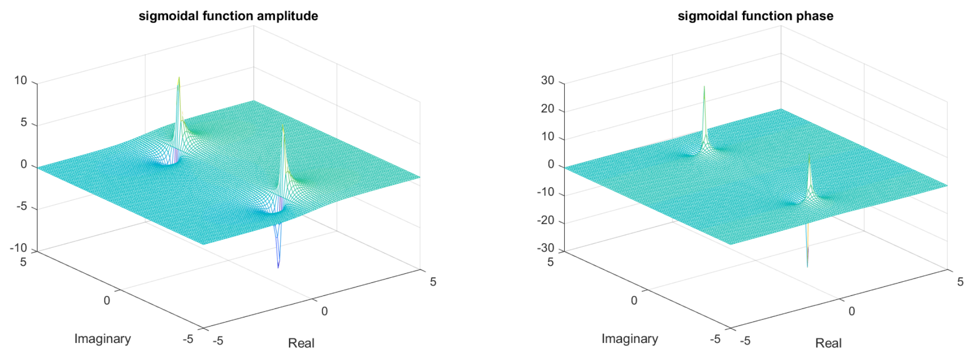


Figure 7.4: Amplitude and phase of sigmoidal function with visible singularities.

$$\mathrm{f}(z) = \frac{1}{1+e^{-z}} \tag{7.2}$$

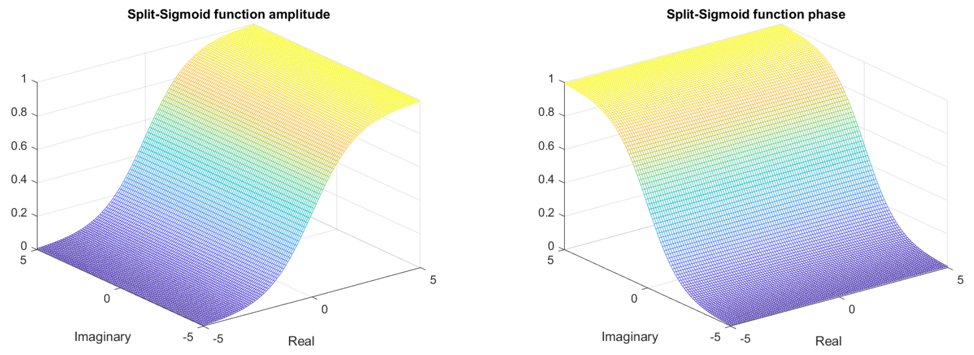


Figure 7.5: Amplitude and phase of split-sigmoid function (Equation 7.3).

$$\text{split-sigmoid}(z_k) = \frac{1}{(1+e^{-\Re(z_k)})} + j\frac{1}{(1+e^{-\Im(z_k)})} \tag{7.3}$$

Lee *et al.*'s work[96] suggests using Wirtinger calculus[171] to bypass the problem. In the paper, leaky ReLU function is modified to holomorphic leaky ReLU function(Equation 7.4)(Figure 7.6). Although the modified function itself retains nice monotonic shape, but as seen in the graph, the function will modify the phase of the input.

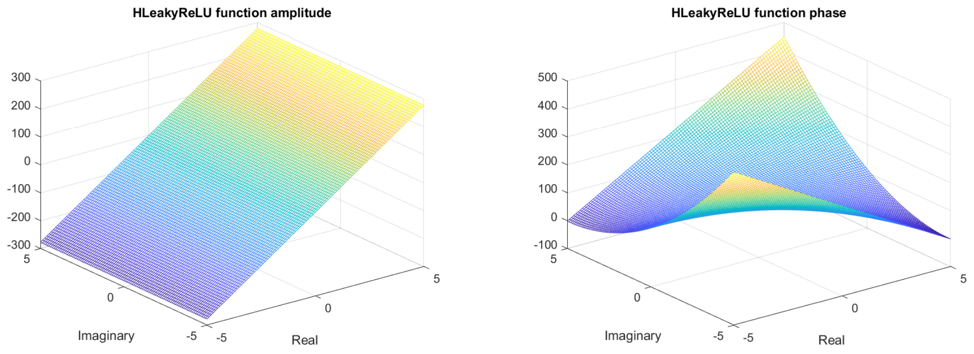


Figure 7.6: Amplitude and phase of Holomorphic modified leaky ReLU function.

$$\text{HLReLU}(z) = \frac{1}{2}\{(1+\alpha)+(1-\alpha)\text{erf}(\frac{z}{\sqrt{2}\sigma})\}z \tag{7.4}$$

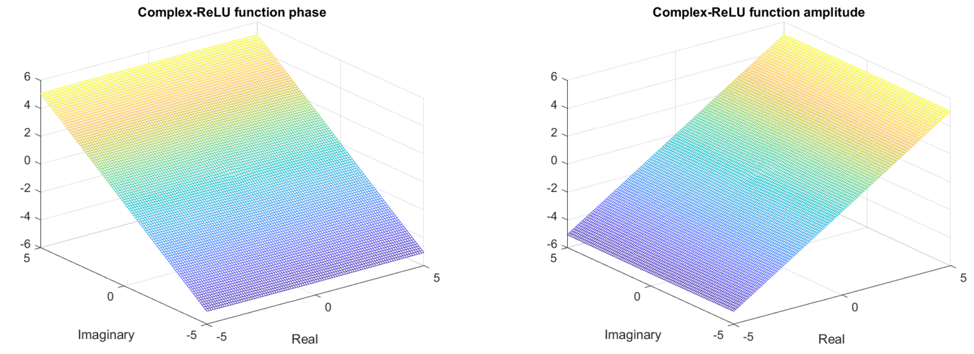


Figure 7.7: Amplitude and phase of Complex-modified ReLU function[7] (Equation 7.5).

$$\text{Complex-ReLu}(z_k) = \frac{\max(0,|z_k|+b)z_k}{|z_k|+c}, b = 0.1, c = 0.001 \tag{7.5}$$

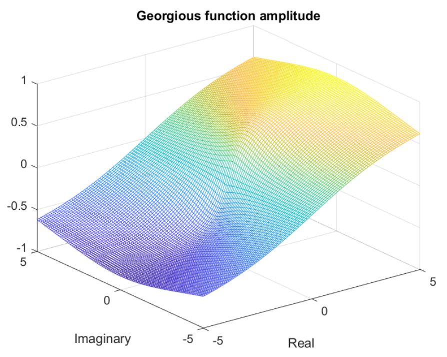


Figure 7.8: Amplitude and phase of Georgious function[57] (Equation 7.6)

$$f(z_k) = \frac{z_k}{c + |z_k| r^{-1}} \tag{7.6}$$

c is a hyper parameter default to 1, and r is a hyper parameter default to 1.

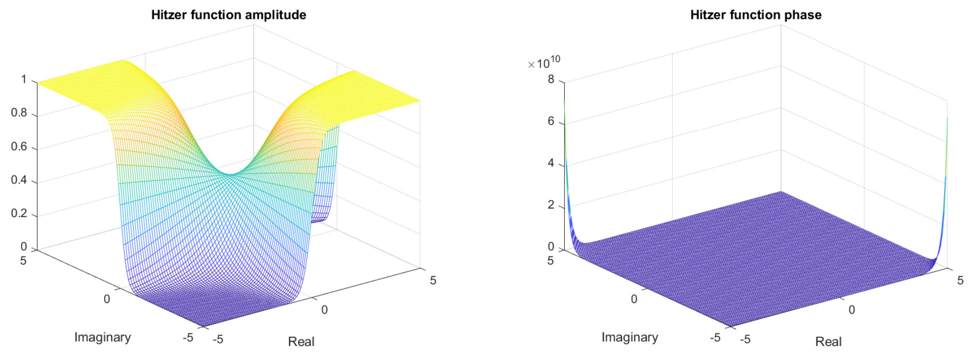


Figure 7.9: Amplitude and phase of Hitzer activation function[70](Equation 7.7).

$$\text{Hitzer}(z_k) = \frac{1}{1 + \exp \mathbf{Re}(z_k)\mathbf{Im}(z_k)} + i\frac{1}{\exp \mathbf{Re}(z_k)\mathbf{Im}(z_k)} \tag{7.7}$$

There are also a few creative attempts at finding the optimal activation functions like CVNN [70][57][7]. Throughout testing, those functions perform very similarly to their closely reassembled counterparts.

7.2.2 Back-propagation

When computing the back-propagation of a complex-valued neural network, the same sets of total chain rules are followed for all the holomorphic functions. First, define the loss function $E(Y,T)$, which is a real-valued function that is differentiable. Here, $Y = f(z)$ is the output from the previously defined CVNN, and T is the training target. With the example set, the back-propagation process can be started. The goal is to find the partial derivatives of all the trainable variables in the network with respect to the loss function $E(Y,T)$. Using Figure 6.19 as a reference, the following items need to be computed:

$\frac{\partial E}{\partial w_{11}}, \frac{\partial E}{\partial w_{12}}, \frac{\partial E}{\partial w_{13}}, \frac{\partial E}{\partial w_{14}}, \frac{\partial E}{\partial w_{21}}, \frac{\partial E}{\partial w_{22}}, \frac{\partial E}{\partial w_{23}}, \frac{\partial E}{\partial w_{24}}, \frac{\partial E}{\partial b_1}$, and $\frac{\partial E}{\partial b_2}$.

For example, to compute $\frac{\partial E}{\partial w_{21}}$:

$$\frac{\partial E}{\partial w_{21}} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \phi_{SM}}\frac{\partial \phi_{SM}}{w_{21}} \tag{7.8}$$

Since ϕ_{SM} is a CR-function, by using Wirtinger Calculus chain rule:

$$\frac{\partial E}{\partial w_{21}} = \frac{\partial E}{\partial y_1}[(\frac{\partial y_1}{\partial w_{21}} \circ \phi_{SM})\frac{\phi_{SM}}{\partial w_{21}} + (\frac{\partial y_1}{\partial \bar{w}_{21}} \circ \phi_{SM})\frac{\bar{\phi}_{SM}}{\partial w_{21}}] \tag{7.9}$$

$\circ$ is the composition function symbol:

$$g \circ f = g(f(z,\bar{z}), \bar{f}(z,\bar{z})) \tag{7.10}$$

Unfortunately, just as shown above, back-propagation for CVNN does not produce sequential-like equations, unlike back-propagation for any sequential RVNN models.

7.2.2.1 Special Layers

Pooling Layer This layer can be simplified with element-wise operation. Figure 7.10 shows an illustration.

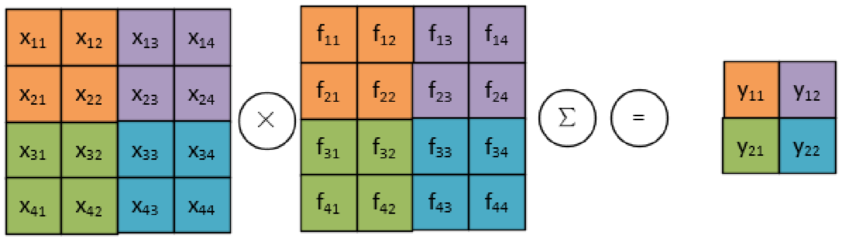


Figure 7.10: Calculation

matrix **X** is the input from the previous layer, matrix **F** is the filter.

$$\begin{bmatrix} y_{11} \\ y_{12} \\ y_{21} \\ y_{22} \end{bmatrix} = \begin{bmatrix} x_{11}f_{11} + x_{12}f_{12} + x_{21}f_{21} + x_{22}f_{22} \\ x_{13}f_{13} + x_{14}f_{14} + x_{23}f_{23} + x_{24}f_{24} \\ x_{21}f_{21} + x_{22}k_{22} + x_{41}k_{41} + x_{42}k_{42} \\ x_{23}f_{23} + x_{24}k_{24} + x_{43}k_{43} + x_{44}k_{44} \end{bmatrix} \tag{7.11}$$

Generalization:

$$y_{k,l} = \sum_{i=km}^{km+m} \sum_{j=ln}^{ln+n} x_{i,j} f_{i,j} \tag{7.12}$$

(m,n) is the pooling window size.

This generalization turns max-pooling layer into an pseudo convolutional layer with only element-wise multiplication and summation. Now it is easy to compute the back-propagation from this generalization.

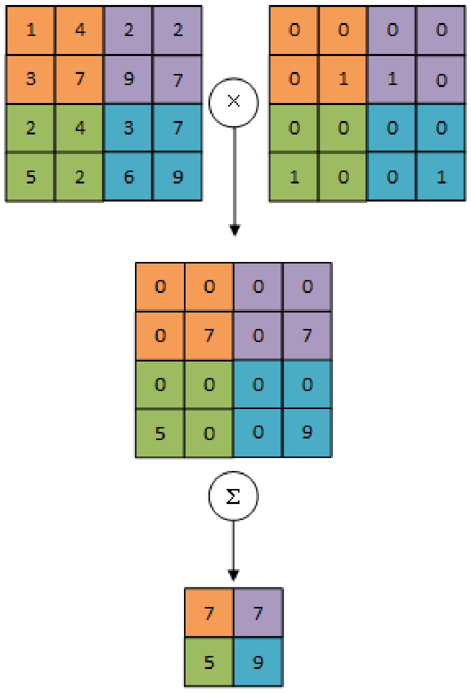


Figure 7.11: A example of max pooling layer.

Full-Connected Layer Full-connected layer is previously defined in equation 6.13 as a simple summation function $\mathbf{F}(x)$.

Softmax Layer Softmax layer can be also treated as a simple summation function $\mathbf{F}(x)$ where previously defined in equation 6.14.

7.2.2.2 Gradient

The gradient of CVNN is similar to RVNN.

$$\frac{\partial E}{\partial w, b} = \begin{bmatrix} \frac{\partial E}{\partial w} & \frac{\partial E}{\partial b} \end{bmatrix} \tag{7.13}$$

In the implementation, Tensorflow computes the gradient with automatic differentiation[170].

$$\frac{\partial E}{\partial w, b} = \begin{bmatrix} \frac{\partial E}{\partial w_1} & \frac{\partial E}{\partial b_1} & \frac{\partial E}{\partial w_2} & \frac{\partial E}{\partial b_2} \end{bmatrix} \tag{7.14}$$

When using *tf.GradientTape()* from Tensorflow, computed gradient is in zipped format and requires flatten to make it a vector with size of m, where m is number of trainable variables.

7.2.2.3 Parameter Updates

The weights updating process follows RVNN, equation 7.15 is the general form.

$$W_{n+1} = W_n + \eta \Delta W_n \tag{7.15}$$

$$b_{n+1} = b_n + \eta \Delta b_n \tag{7.16}$$

η is the learning rate.

7.2.2.4 Complex Modified Gradient Clipping

In the context of neural network training, the phenomenon of gradients either exploding or vanishing is a well-recognized challenge. The vanishing gradient problem occurs when a neural network is excessively deep, comprising numerous layers, resulting in the relative impact of the derivative becoming too minuscule to influence the kernel significantly. Conversely, the exploding gradient phenomenon is characterized by an abnormally large gradient that propels the kernel too swiftly towards a minimum, standing in contrast to the vanishing gradient issue.

Addressing the problem of exploding gradients can be approached through a seemingly simplistic strategy: constraining the gradient's value to a specific range when it becomes excessively large. This approach, referred to as gradient clipping, has been explored and detailed in prior works[177]. Within real-valued neural networks, gradient clipping typically involves suppressing the gradient by dividing it by its norm. Intriguingly, this concept can be extended and applied to complex-valued neural networks as well.

To perform this extension, one can compute the complex norm using equation 7.17. Utilizing the complex norm is advantageous since it yields a scalar quantity without altering the gradient's angle. Upon computation of this norm, it must be ascertained whether the norm surpasses a constant real value c (which is generally preset to 1). Should the norm exceed c, the gradient is scaled down by dividing it by the norm, as delineated in equation 7.18, mirroring the procedure employed in its real-valued analogue.

$$d_k = \sqrt{R(||\mathbf{H}_k||)^2 + C(||\mathbf{H}_k||)^2} \tag{7.17}$$

$$\mathbf{H}_k^{\text{clip}} = \begin{cases} \mathbf{H}_k, d_k < c \\ \frac{\mathbf{H}_k \cdot c}{d_k}, d_k \geq c \end{cases} \tag{7.18}$$

$\mathbf{H}_k$ is the gradient normalizing to the constant value c.

7.3 Optimizer

7.3.1 Complex Extended Kalman Filter

Extended Kalman filter (EKF) is a variant of the Kalman filter method, which approximates the system as a linear system. When modelling the Neural Network as a nonlinear system, EKF can be a powerful tool to train and optimize the neural network [77].

The EKF trainable weights update process follows the figure 7.12.

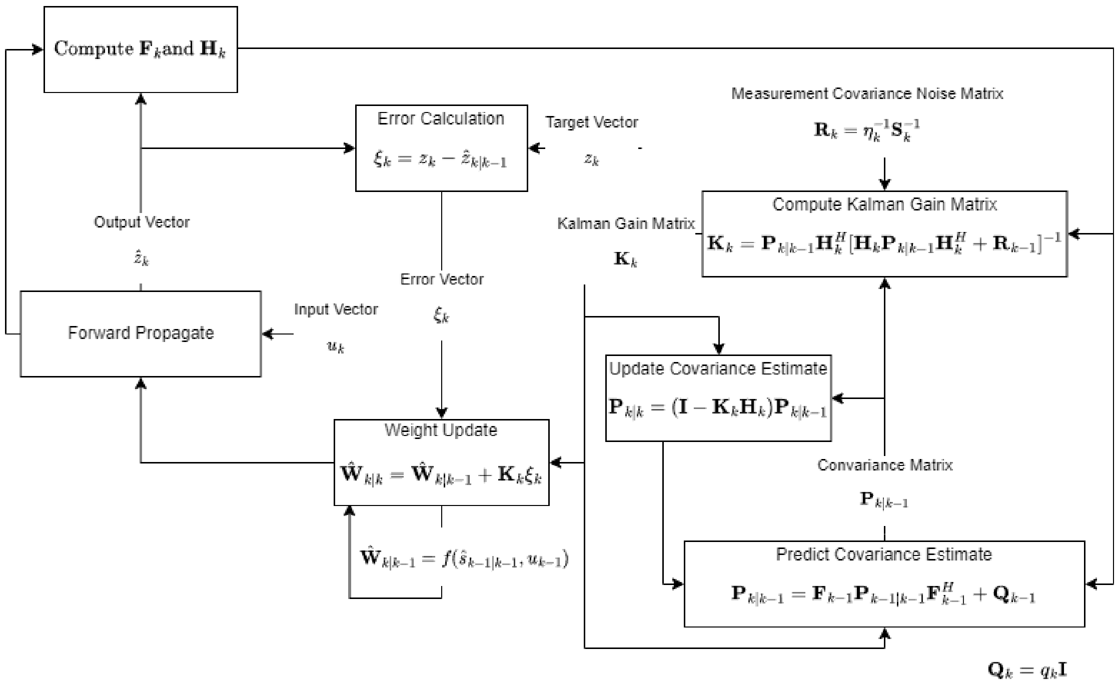

Compute $\mathbf{F}_k$ and $\mathbf{H}_k$
Error Calculation
$\xi_k = z_k - \hat{z}_{k|k-1}$
Target Vector
z_k
Measurement Covariance Noise Matrix
$\mathbf{R}_k = \eta_k^{-1}\mathbf{S}_k^{-1}$
Compute Kalman Gain Matrix
$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}_k^H[\mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^H + \mathbf{R}_{k-1}]^{-1}$
Kalman Gain Matrix
$\mathbf{K}_k$
Output Vector
$\hat{z}_k$
Error Vector
ξ_k
Forward Propagate
Input Vector
u_k
Update Covariance Estimate
$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k|k-1}$
Weight Update
$\hat{\mathbf{W}}_{k|k} = \hat{\mathbf{W}}_{k|k-1} + \mathbf{K}_k\xi_k$
Convariance Matrix
$\mathbf{P}_{k|k-1}$
$\hat{\mathbf{W}}_{k|k-1} = f(\hat{s}_{k-1|k-1}, u_{k-1})$
Predict Covariance Estimate
$\mathbf{P}_{k|k-1} = \mathbf{F}_{k-1}\mathbf{P}_{k-1|k-1}\mathbf{F}_{k-1}^H + \mathbf{Q}_{k-1}$
$\mathbf{Q}_k = q_k\mathbf{I}$

Figure 7.12: EKF Training process

When applying the EKF to train CVNN, the first step involves treating the CVNN as a nonlinear system, as delineated in Equations 7.19 and 7.20. In this framework, $\hat{\mathbf{W}}$ represents the system state vector, which corresponds to the weights of the CVNN (Equation 7.28). The system output, denoted as $\hat{z}$, is the prediction generated by the NN, and u_{k-1} is the system input, referring to the training data. The function $h(\hat{\mathbf{W}})$ is described as a nonlinear output function of the CVNN. $\mathbf{Q}$ is the covariance of the process noise, $\mathbf{F}$ is the jacobian of function f with respect to the weights $\mathbf{W}$(Equation 7.26).

$$\hat{\mathbf{W}}_{k|k-1} = f(\hat{\mathbf{W}}_{k-1|k-1}, u_{k-1}) \tag{7.19}$$

$$\hat{z}_{k|k-1} = h(\hat{\mathbf{W}}_{k|k-1}) \tag{7.20}$$

$$\mathbf{P}_{k|k-1} = \mathbf{F}_{k-1}\mathbf{P}_{k-1|k-1}\mathbf{F}_{k-1}^H + \mathbf{Q}_{k-1} \tag{7.21}$$

Equations 7.22 to 7.25 illustrate the update process. The residual ξ_k is computed from the NN prediction $\hat{z}_{k|k-1}$ and the target output z_k. The Kalman Gain $\mathbf{K}$ is computed using the covariance of the measurement noise $\mathbf{R}$, the covariance estimate $\mathbf{P}$, and the Jacobian matrix $\mathbf{H}$ (Equation 7.27). Equation 7.24 is utilized to compute the update of the weights, while Equation 7.25 is responsible for updating the covariance estimate.

$$\xi_k = z_k - \hat{z}_{k|k-1} \tag{7.22}$$

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}_k^H[\mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^H + \mathbf{R}_{k-1}]^{-1} \tag{7.23}$$

$$\hat{\mathbf{W}}_{k|k} = \hat{\mathbf{W}}_{k|k-1} + \mathbf{K}_k\xi_k \tag{7.24}$$

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k|k-1} \tag{7.25}$$

$$\mathbf{F}_{k-1} = \frac{\partial f}{\partial \mathbf{W}}\Big|_{\hat{\mathbf{W}}_{k-1|k-1}, u_{k-1}} \tag{7.26}$$

$$\mathbf{H}_k = \frac{\partial h}{\partial \mathbf{W}}\Big|_{\hat{\mathbf{W}}_{k|k-1}} \tag{7.27}$$

$$\mathbf{W} = \begin{bmatrix} W_{1,1}^{1} \\ \vdots \\ W_{N_{\text{Layers}}, N_{\text{Layers}-1}}^{\text{Layers}-1} \end{bmatrix} \tag{7.28}$$

$$\mathbf{F} = \mathbf{I} \tag{7.29}$$

Table 7.1: EKF Parameters

Parameter	Memory Usage
$\mathbf{P}$	$O(n^2)$
$\mathbf{Q}$	$O(n^2)$
$\mathbf{R}$	$O(n^2)$
$\mathbf{H}$	$O(n^2)$
$\mathbf{K}$	$O(n)$
$\hat{\mathbf{W}}$	$O(n)$
ξ	$O(n)$

7.3.2 Smooth Variable Structure Filter

Smooth variable structure filter (SVSF)[64] is a variant of a linear estimator that creates a confined boundary on the estimated state trajectory. Compared to EKF, SVSF relaxes the constraint on the uncertainty assumptions made by the Kalman filter. There are studies[64, 78] on applying the SVSF to neural network training and proving its superior performance. Paper[2] proposed the SVSF-training method on perception by feeding the loss value and tracking kernel as its state, and the work showed a variant of SVSF performs superior to Adam or Stochastic Gradient Descent (SGD) in a real-valued neural network.

Algorithm: SVSF follows the same EKF-style update.

1. Initialize all the tuning parameters:γ,ψ,η. $\mathbf{W_k}$ is initialized with complex-modified Glorot initialization.
2. Compute the forward propagation by feeding a training variable z_k into the CVNN, then the output y_k. k is the step number.
3. Loss $\hat{y}_k$ is computed from a selected loss function, in my MINST test this is computed with cross-entropy loss.
4. Perform the backward propagation and obtain the derivative matrix $\mathbf{H}_k$, computed by Jacobian of loss respect to all the trainable variables.
5. Compute the Moore–Penrose inverse of the derivative matrix $\mathbf{H}_k^+ = \frac{\mathbf{H}_k^t}{\sum_{i=1}^n (H_k(i))^2}$.
6. Obtain the SVSF Gain $\mathbf{K}_k = \mathbf{H}_k^+(\eta\|\hat{y}_k\|_{\text{abs}} + \gamma\|\hat{y}_{k-1}\|_{\text{abs}}) \circ \text{sat}(\frac{\hat{y}_k}{\psi})$
7. Update the state vector $\hat{\mathbf{W}}_{k+1} = \hat{\mathbf{W}}_k + \mathbf{K}_k$. This also updates all the trainable in the CVNN.
8. Update the tuning parameter ψ based on the following conditions:
9. Repeat from the Step 2 until the training condition is met: ξ_k is below a threshold or reaching maximum number of epochs.

Saturation function $\text{sat}(\frac{\hat{y}_k}{\psi})$ is defined in equation 7.30

$$\text{sat}(\frac{\hat{y}_k}{\psi}) = \begin{cases} -1, & \text{for } \frac{\hat{y}_k}{\psi} \leq -1 \\ \frac{\hat{y}_k}{\psi}, & \text{for } -1 \leq \frac{\hat{y}_k}{\psi} \leq 1 \\ 1, & \text{for } 1 \leq \frac{\hat{y}_k}{\psi} \end{cases} \tag{7.30}$$

7.3.3 Complex-Valued Stochastic Gradient Descent

Complex-valued SGD follows exactly the same procedure as its real-valued counterpart, as shown in Algorithm 5. The algorithm uses batch training across multiple samples and randomized training sample orders,

Table 7.2: SVSF Parameters

Parameter	Memory Usage
$\mathbf{H}$	$O(n^2)$
$\mathbf{K}$	$O(n)$
$\mathbf{W}$	$O(n)$
l_r	$O(1)$
c_v	$O(1)$

Algorithm 5 CVNN SGD Training

```
Require: $l_r$, $c_v$
  while $k \leq$ number of epochs do
    $k \leftarrow k+1$
          # Compute the gradient
    $\mathbf{H}_k = \nabla_{\mathbf{W}} \text{loss}(\mathbf{W}_k)$
          # Compute the clipped gradient
    $||\mathbf{H}_k|| = \sum_{i=1}^{n} \sqrt{h_i \bar{h}_i}$
    $d_k = \sqrt{\Re(||\mathbf{H}_k||)^2 + \Im(||\mathbf{H}_k||)^2}$
    $\mathbf{H}_k^{\text{clip}} = \begin{cases} \mathbf{H}_k, d_k < c_v \\ \frac{\mathbf{H}_k \cdot c}{d_k}, d_k \geq c_v \end{cases}$
          # Update the weights and bias
    $\mathbf{W}_{k+1} = \mathbf{W}_k - l_r \cdot \mathbf{H}_k^{\text{clip}}$
  end while
  return $\mathbf{W}_k$
```

thus earning the name Stochastic. Just like the counterpart, this randomness is developed to counter overfitting and allows the training to escape local minima.

Table 7.3: SGD Parameters

Parameter	Memory Usage
$\mathbf{H}$	$O(n^2)$
$\mathbf{K}$	$O(n)$
$\mathbf{W}$	$O(n)$
l_r	$O(1)$
c_v	$O(1)$

7.3.4 Complex-Modified Reduced-Smooth Variable Structure Filter

reduced-SVSF SVSF is a second-order optimization algorithm that relies on the Hessian matrix. It faces a similar issue as the EKF method, using a substantial amount of VRAM to track state variables and compute second-order derivatives through auto-differential. Consequently, Ismail *et al.*'s work[78] introduced a streamlined version of the SVSF optimizer to rival other first-order optimizers in terms of performance and speed. In contrast to the original SVSF, the modified reSVSF approach conducts optimization using gradients instead of the Hessian matrix. This version of the algorithm focuses on scalar output for the non-linear system and shares scalar values for tuning parameters. Consequently, the algorithm significantly reduces VRAM usage during the training process. However, the initial reSVSF implementation was not applicable to complex-valued neural networks, displaying instability and resulting in exploding gradients. As a solution, this study introduces modifications to the reSVSF algorithm as follows:

- A complex-modified gradient clipping strategy was adopted to prevent exploding gradients in CVNN.
- An additional learning rate variable was introduced for fine-tuning the training.
- Constants were modified to accommodate the larger magnitude of complex numbers.

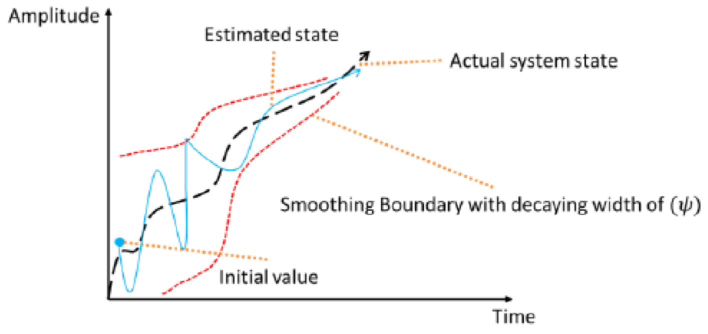
$$\text{sat}(\frac{y_{k|k-1}}{\psi}) = \begin{cases} 1, & y_{k|k-1} \geq 1 \\ y_{k|k-1}, & -1 < y_{k|k-1} < 1 \\ -1, & y_{k|k-1} \leq -1 \end{cases} \tag{7.31}$$

Algorithm 6 complex-modified reduced SVSF Training

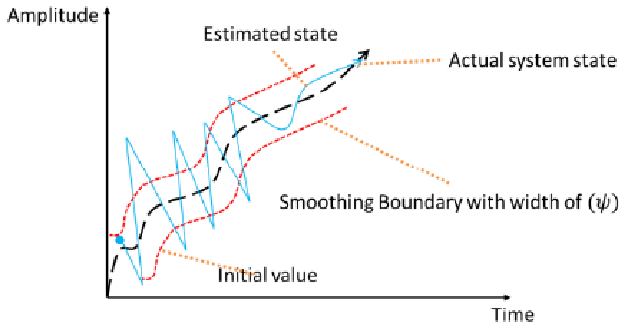
```
Require: $\gamma$, $\psi$, $\eta$, $l_r$:
  while $k \leq$ number of epochs do
    $k \leftarrow k+1$
    $y_{k|k-1} = 1 - \text{loss}(\mathbf{W}_k)$
    $\mathbf{H}_k = \nabla_{\mathbf{W}} \text{loss}(\mathbf{W}_k)$
          # Compute the update parameter
    $||\mathbf{H}_k|| = \sum_{i=1}^{n} \sqrt{h_i \bar{h}_i}$
    $d_k = \sqrt{R(||\mathbf{H}_k||)^2 + C(||\mathbf{H}_k||)^2}$
    $\mathbf{H}_k^{\text{clip}} = \begin{cases} \mathbf{H}_k, d_k < c \\ \frac{\mathbf{H}_k \cdot c}{d_k}, d_k \geq c \end{cases}$
    $\mathbf{K}_k = \mathbf{H}_k^{\text{clip}} (\eta |y_{k|k-1}|_{\text{abs}} + \gamma |y_{k-1|k-1}|_{\text{abs}}) \cdot l_r \cdot \text{sat}(\frac{y_{k|k-1}}{\psi})$
          # Update the weights and bias
    $\mathbf{W}_{k+1} = \mathbf{W}_k + l_r \cdot \mathbf{K}_k$
    if Decay then
      $\psi = \max(\frac{\psi}{1+0.48k} - 1, 10^{-2}\psi)$
    end if
  end while
```

Table 7.4: reSVSF Parameters

Parameter	Memory Usage
$\mathbf{H}$	$O(n)$
$\hat{y}_k$	$O(1)$
$\hat{y}_{k-1}$	$O(1)$
ψ	$O(1)$
η	$O(1)$
γ	$O(1)$
c_v	$O(1)$



(a) Decaying boundary layer width



(b) Fixed boundary layer width

Figure 7.13: SVSF boundary layer width

Boundary Decay The saturation function (Equation 7.31) was used to suppress the chattering of $\mathbf{K}_k$ in the tracking trajectory. The original reSVSF work proposed two approaches for suppression (Figure 7.13a and Figure 7.13b). The first was to have a decaying boundary layer width, where, for each iteration, the boundary width ψ is modified so the boundary shrinks. The other approach is to have a fixed boundary width ψ. The cm-reSVSF follows the same approach with the options of decaying boundary width or fixed boundary width during the training.

7.3.5 Complex Adam

Adam is a very popular optimizer used in NN training. The algorithm is designed to adaptively change the learning rate throughout the training process by utilizing moments computed from gradients and squared gradients. Adam in RVNN has a reputation for performing better than SGD in training, and it is often the first optimizer of choice for training the NN. This optimizer has been implemented and deployed in this study to benchmark and compare the performance of Adam in CVNN.

Algorithm 7 CVNN Adam Training

```
Require: $l_0$, $c_v$, $\beta_1$,$\beta_2$,$\varepsilon$
  while $k \leq$ number of epochs do
    $k \leftarrow k+1$
          # Compute the gradient
    $\mathbf{H}_k = \nabla_{\mathbf{W}} \text{loss}(\mathbf{W}_k)$
          # Compute the clipped gradient
    $||\mathbf{H}_k|| = \sum_{i=1}^{n} \sqrt{h_i \bar{h}_i}$
    $d_k = \sqrt{\Re(||\mathbf{H}_k||)^2 + \Im(||\mathbf{H}_k||)^2}$
    $\mathbf{H}_k^{\text{clip}} = \begin{cases} \mathbf{H}_k, d_k < c_v \\ \frac{\mathbf{H}_k \cdot c}{d_k}, d_k \geq c_v \end{cases}$
          # Compute the momentum
    $\mathbf{m}_k = \beta_1 \cdot \mathbf{m}_{k-1} + (1-\beta_1) \cdot \mathbf{H}_k^{\text{clip}}$
    $\mathbf{v}_k = \beta_2 \cdot \mathbf{v}_{k-1} + (1-\beta_2) \cdot (\mathbf{H}_k^{\text{clip}})^2$
          # Compute the Learning Rate
    $l_k = l_{k-1} \cdot \frac{\sqrt{1-\beta_2^k}}{(1-\beta_1^k)}$
          # Update the weights and bias
    $\mathbf{W}_{k+1} = \mathbf{W}_k - \frac{l_k \cdot \mathbf{m}_k}{\sqrt{\mathbf{v}+\varepsilon}}$
  end while
  return $\mathbf{W}_k$
```

Table 7.5: Adam Parameters

Parameter	Memory Usage
$\mathbf{m}$	$O(n)$
$\mathbf{v}$	$O(n)$
$\mathbf{W}$	$O(n)$
$\mathbf{H}$	$O(n)$
l	$O(1)$
β_1	$O(1)$
β_2	$O(1)$
ε	$O(1)$
c_v	$O(1)$

7.4 Result

7.4.1 Setup

The experiment was performed on three different PCs, and specs are shown in the table 7.6. Since the implementation utilizes CUDA GPU calculations, therefore all testing machines are equipped with Nvidia GPUs. CPU is equally important in the specifications, as there are many operations done in the CPU, which affect the computing time, more discussion on this later on.

	Desktop 1	**Desktop 2**	**Laptop 1**
CPU	Intel i7 9700K	AMD Ryzen 5900X	AMD Ryzen 4900HS
CPU Cores	8	12	8
CPU Threads	8	24	16
RAM	32GB	32G	16G
GPU	Nvidia RTX 2080Ti	Nvidia RTX 3090	Nvidia RTX 2060 MAXQ
VRAM	11GB	24GB	6GB
Cuda Cores	4352	10752	1920
CUDA Ver.	10.1	11.7	11.7
Driver	450.00.00	515.65.01	515.65.01
HDD	Samsung 970Pro	Samsung 980Pro	Samsung OEM SSD
OS	Ubuntu 20.04 LTS	Ubuntu 20.04 LTS	Ubuntu 20.04 LTS

Table 7.6: Training machine specification

Second-order training algorithms were incorporated into the CVNN package. Nevertheless, due to their exceedingly high memory requirements and lengthy computation times, these methods lack feasibility for practical applications. The utilization of second-order differentiation through auto-differentiation entails that even a miniscule complex perception cannot be accommodated on the laptop owing to the constraints of limited VRAM. The sole experiment feasible with second-order training encompasses fewer than two hidden layers of perception or a complex-valued CNN. Given the aforementioned reasons, all second-order training has been omitted from the conducted experiments.

The goal of the experiment is to test the performance on different training methods on CVNN, performance are measured based off the following measurements: validation accuracy, loss, and training time. For the experiment, 3 complex-valued NN were deployed (Figure 7.14) for Modified National Institute of Standards and Technology (CIFAR), Canadian Institute For Advanced Research (CIFAR)-10, and CIFAR-100 dataset.

For the training dataset, the CIFAR dataset[43] was utilized. The dataset is composed of 28×28 greyscale images of 10 classes, representing numbers from 0 to 9, with a total of 10k training samples. The dataset offers a good balance of complexity and relatively small input size, making it an excellent benchmark testing dataset for classification tasks. To prepare the data for CVNN, the spatial representation of handwritings was

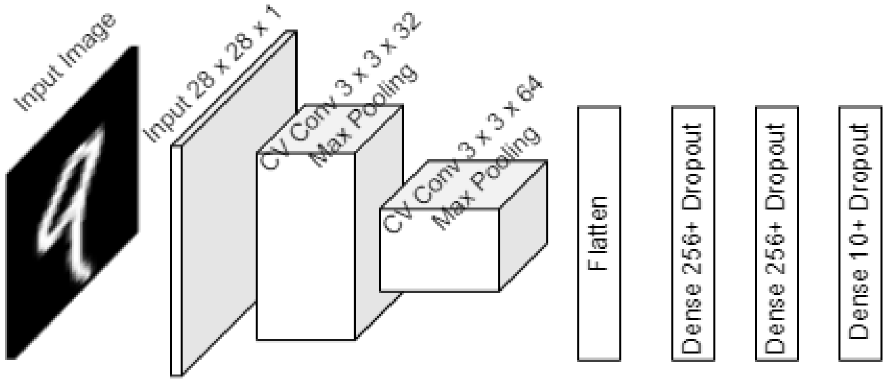


Figure 7.14: Testing complex-valued CNN setup for CIFAR dataset

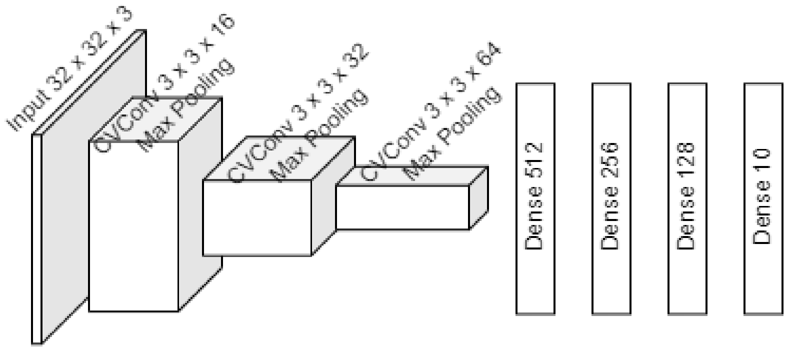


Figure 7.15: Testing complex-valued CNN setup for CIFAR-10 dataset[92]

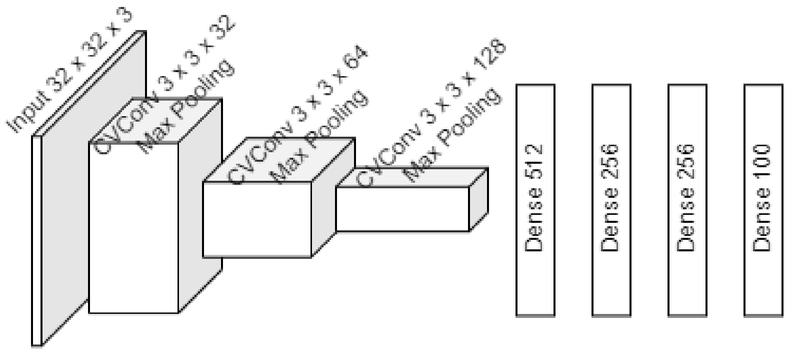


Figure 7.16: Testing complex-valued CNN setup for CIFAR-100 dataset[93]

	MINST	CIFAR10	CIFAR100
Split Tanh	0.947854	0.925235	0.793214
Split ReLU	0.943669	0.925814	0.798926
cm-ReLU	0.952068	0.926417	0.795441
Tanh	0.947195	0.92868	0.791218
Georgious	0.939633	0.926358	0.795249

Table 7.7: Best Validation Accuracy of each activation function

pre-converted into 2D frequency bins through the 2D Fourier transform process. Thus, the input tensor for each sample is a $[28, 28, 1]$ complex float tensor. Since mini-batch training is being used, and the batch size is set to 64, the input tensor for each training instance is $[64, 28, 28, 1]$. CIFAR-10 dataset[92] and CIFAR-100 dataset[93] are other sets of more complex classification benchmark databases. The CIFAR-10 consists of 60,000 $[32 \times 32]$ coloured images that belong to one of the 10 classes, thus giving it the name CIFAR-10. CIFAR-100, on the other hand, has 60,000 $[32 \times 32]$ coloured images but with 100 classes, making it more challenging for a classifier than CIFAR-10 due to having fewer samples per class. Both datasets are popular among neural network researchers. In this research, images were preprocessed and converted to the frequency domain with 2D-FFT performed on each channel, and with a batch size of 32, the input size of the NN is $[32, 32, 32, 3]$ complex tensor.

7.4.2 Complex-Valued Activation Functions

A few selected Complex-Valued activation functions were tested, as shown in the Table 7.7. The first observation is that singularities in the function do not affect the performance of the function; this is mainly due to value clipping and normalization applied between layers. Another observation is that there is no significant performance difference across different functions; it is safe to assume that small variations in results are due to training variations.

7.4.3 Clipping VS Non-Clipping

Through the experiment, it can be concluded that training a complex-valued neural network does require clipping the gradient. The experimental result showed that training with clipped gradients performs better than with non-clipped gradients. This observation further indicates that CVNN is more sensitive compared to RVNN[15]. Figure 7.17 displayed the clipped cm-reSVSF optimizer with clipping versus SGD, Adam, and keras-implemented Adam, where the latter optimizers do not have clipping. The diagram shows significant fluctuation in training if clipping was not applied.

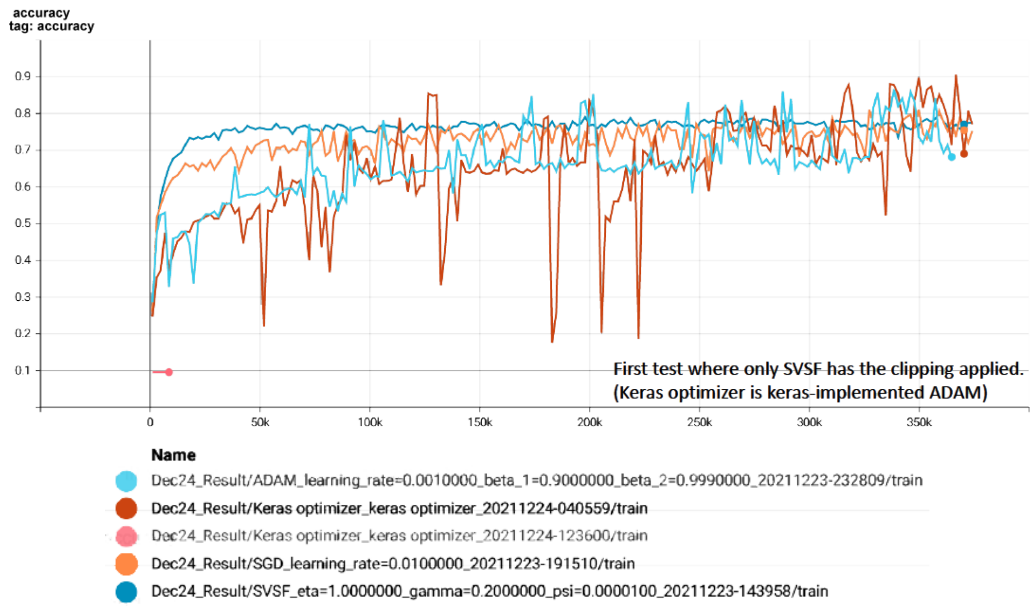
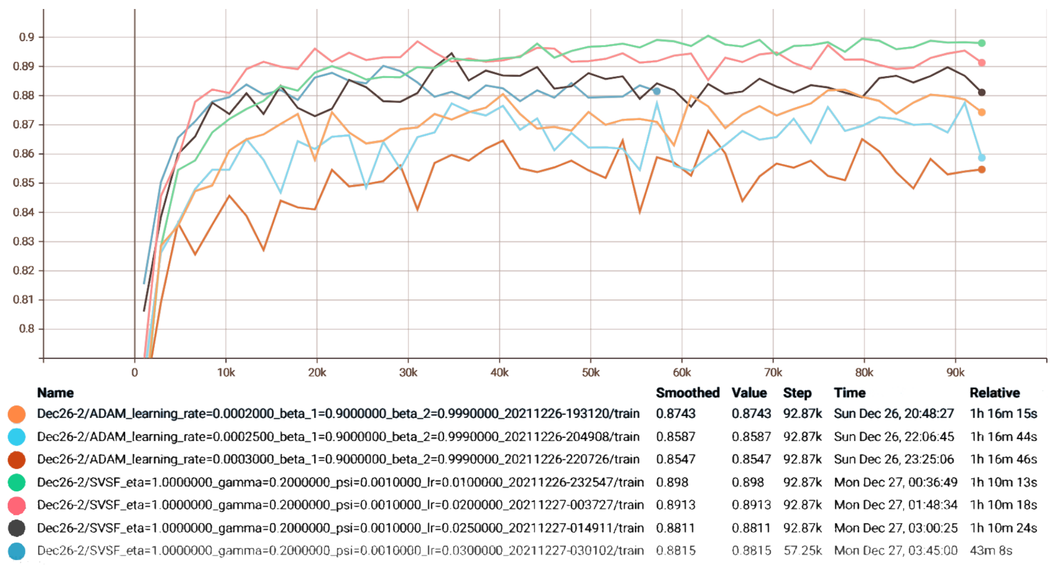


Figure 7.17: Clipping Benchmark

7.4.4 cm-reSVSF vs SGD vs Adam

Cm-reSVSF not only converges significantly faster when compared to Adam and SGD with the same number of epochs, but it also exhibits substantially reduced computing time (around 10% less training time) compared to Adam. All three algorithms were implemented using the same coding style to avoid differences in computing time arising from coding efficiency. Additionally, Adam from the KERAS package with CUDA support was tested against the implemented Adam implementation, and both versions of Adam ran at approximately the same speed. This further confirms that the computational speed advantage of cm-reSVSF is not attributed to programming deficiencies in the implementation. Table 7.8 shows the training result of CiFAR-100 dataset, and clearly cm-reSVSF is faster than other two algorithms in comparison with higher accuracy and less training time.



	Name	Smoothed	Value	Step	Time	Relative
●	Dec26-2/ADAM_learning_rate=0.0002000_beta_1=0.9000000_beta_2=0.9990000_20211226-193120/train	0.8743	0.8743	92.87k	Sun Dec 26, 20:48:27	1h 16m 15s
●	Dec26-2/ADAM_learning_rate=0.0002500_beta_1=0.9000000_beta_2=0.9990000_20211226-204908/train	0.8587	0.8587	92.87k	Sun Dec 26, 22:06:45	1h 16m 44s
●	Dec26-2/ADAM_learning_rate=0.0003000_beta_1=0.9000000_beta_2=0.9990000_20211226-220726/train	0.8547	0.8547	92.87k	Sun Dec 26, 23:25:06	1h 16m 46s
●	Dec26-2/SVSF_eta=1.0000000_gamma=0.2000000_psi=0.0010000_lr=0.0100000_20211226-232547/train	0.898	0.898	92.87k	Mon Dec 27, 00:36:49	1h 10m 13s
●	Dec26-2/SVSF_eta=1.0000000_gamma=0.2000000_psi=0.0010000_lr=0.0200000_20211227-003727/train	0.8913	0.8913	92.87k	Mon Dec 27, 01:48:34	1h 10m 18s
●	Dec26-2/SVSF_eta=1.0000000_gamma=0.2000000_psi=0.0010000_lr=0.0250000_20211227-014911/train	0.8811	0.8811	92.87k	Mon Dec 27, 03:00:25	1h 10m 24s
●	Dec26-2/SVSF_eta=1.0000000_gamma=0.2000000_psi=0.0010000_lr=0.0300000_20211227-030102/train	0.8815	0.8815	57.25k	Mon Dec 27, 03:45:00	43m 8s

Figure 7.18: cm-reSVSF vs Adam in CiFAR 100

	Validation Loss	Validation Accuracy	Time	Learning Rate	Clipping
complex-Adam	0.2132	0.8834	25m 25s	0.001	Yes
cm-reSVSF	0.2105	0.8959	22m 44s	0.002	Yes
SGD	0.2253	0.8405	23m 13s	0.001	Yes

Table 7.8: Training result on CiFAR-100

7.4.5 Repeatability and Stability of cm-reSVSF

Figure 7.19 shows the repeatability of cm-reSVSF training with fixed parameters and 50 times testing on the CIFAR dataset with $\pm 0.52\%$ validation accuracy. The result exhibits similar results on CiFAR-10 $\pm 0.76\%$ and CiFAR-100 $\pm 1.22\%$, showing excellent repeatability of the training algorithm. CiFAR-10 and 100 have larger variance in validation due to deeper network structure and larger training dataset (where training data slicing is done at random, this contributes very little to the validation accuracy variance).

Figure 7.20 shows the stability test with cm-reSVSF, by adjusting training parameters' learning rate starting from 0.010 all the way to 0.060, with an increment of 0.01 each time. The result demonstrates exceptional robustness in accepting the changes in training parameters, as the system remains stable within the testing range (Validation accuracy $\pm 0.58\%$).

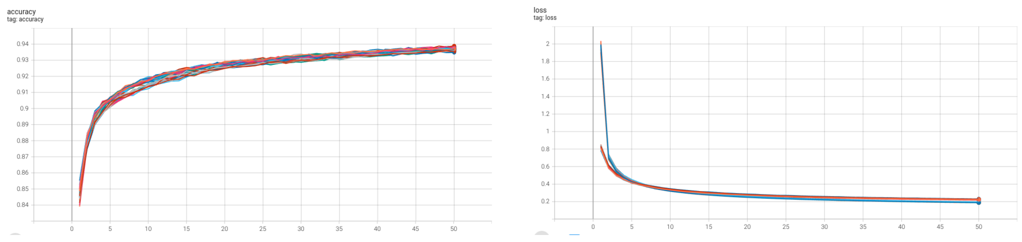


Figure 7.19: Repeatability plot of cm-reSVSF. Left: Validation accuracy; Right: Validation loss.

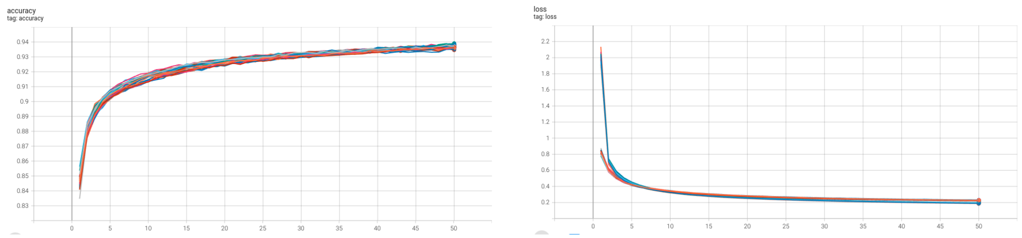


Figure 7.20: Tolerance plot of cm-reSVSF. Left: Validation accuracy; Right: Validation loss.

7.5 Conclusion

First part of this chapter discusses in depth the activation functions and optimizers for CVNN. The chapter begins by explaining three common approaches to activation functions: split functions, transcendental functions, and holomorphic functions. Throughout the experiment, there is no significant performance difference among the activation functions for classification tasks. The second part of the chapter details optimization strategies in training and introduces the novel optimizer cm-reSVSF. Through experimental benchmarking of the strategy with other various training optimizers using the MNIST, CIFAR-10, and CIFAR-100 datasets, the superiority of cm-reSVSF in training the CVNN is demonstrated.

Chapter 8

Conclusion and Future Research

8.1 Conclusion

This chapter covers the primary findings and contributions resulting from the study, followed by suggestions for prospective research for the future that can complement this research.

In the introductory part of the thesis, the technological characteristics of the sensors integrated into the vehicle were detailed. A comprehensive explanation of the implementation of the real-time synchronized sensor capturing system was provided, followed by a proposition of the calibration methodology for the sensors. This involved customizing a tailored sensor automatic rectification process to calibrate the LiDAR, camera, and IR camera affixed to the vehicle. The results were subsequently benchmarked, achieving an accuracy of under 2% pixel error. The ensuing chapter detailed the data amassed with the aforementioned sensor system, furnishing a unique and novel hand-labeled dataset incorporating all three sensors, particularly under adverse weather conditions. This dataset stands as an invaluable resource for autonomous driving studies, especially pertaining to secure driving during nocturnal hours or in rainfall. Additionally, the software development kit (SDK) implemented in conjunction with the dataset was elucidated.

Chapter 5 expounded a novel ground segmentation and peak detection method that leverages the Savitzky-Golay filter for peak detection. When compared with nine other leading ground segmentation techniques, the results were remarkable. The method ranked second in speed, just marginally trailing the fastest method, but outperformed in recall rate, rendering the algorithm an ideal selection for the tasks needed in this research.

Chapter 6 delved into the implementation of preprocessing ROI post ground segmentation on LiDAR data, along with the neural networks designated for classification. Particularly, the autoencoder utilized to reconstruct partially obscured or sparse data missing from LiDAR object clusters was detailed. Further elucidation of the CNN and CVNN neural networks implemented for each sensor input led to terming the system a hybrid neural network. An in-depth explanation of the training of each section of the neural network, the dataset employed during training, and the voting mechanism for testing was provided. The neural network was subsequently subjected to tests with the captured data, affirming the success of the system.

Chapter 7 chronicled the studies conducted on the CVNN, with a primary emphasis on optimization research. This included detailing various activation functions, comparing them, and then shifting focus to the optimization of the NN. In these studies, the SVSF filter, along with a modified version known as reduced-SVSF filter, was harnessed, culminating in the introduction of cm-reSVSF, a complex-valued variant of the reduced SVSF optimizer, designed specifically for CVNN training. The results were extraordinary, with a notable performance enhancement over ADAM, particularly in CVNN.

In summation, the thesis presented the implementation of a real-time vehicle perception system based on heterogeneous neural network sensor fusion. It articulated the strategies and methodologies employed at each stage of the processing pipeline. Through testing of each segment and a final system test, it was demonstrated that the system is functional and constitutes a potential component of autonomous driving vehicles. This research has suggested that heterogeneous sensor fusion is critical to road safety, especially as it mitigates the issues associated with the liability of homogeneous sensor fusion under specific environmental and weather conditions. It should become standardized in future ADAS systems. Furthermore, the datasets produced from this research provide excellent material for future research, serving both research and testing purposes.

8.2 Future Works

8.2.1 SLAM and Localization

By leveraging a dependable vehicle perception system that combines both 3D and 2D environmental data, the implementation of simultaneous localization and mapping (SLAM) for precise vehicle positioning becomes feasible. This capability could potentially be seamlessly integrated into a more comprehensive mapping framework, facilitating the creation of a real-time map updating and navigation system. Additionally, in conjunction with GPS and road information, vehicles could attain a heightened localization performance, nearly pinpointing their exact location. With such an accurate localization system in place, a plethora of applications become possible. For instance, it could enable autonomous vehicle parking or even the automation of vehicle parking within a designated area, essentially allowing vehicles to be parked or retrieved without the need for human intervention.

8.2.2 Centralized Information Exchange

Current research is focused on self-surrounding awareness; thus, each vehicle relies on its own sensor for surrounding detection, without exchanging information with other sensor-equipped vehicles. It is possible to establish a control centre to process and exchange the collected vehicle information. In ongoing smart city projects in numerous countries such as Singapore and China, this sensor information can offer real-time feedback, supplementing existing smart-city infrastructure. These vehicles effectively function as the eyes of a smart city.

Furthermore, when compared to the considerable investments made in smart city IoT infrastructure, these mobile vehicle sensors come with financial advantages (lower costs on the public side), mobility, and high availability. Nonetheless, challenges in constructing such a system can be foreseen, particularly concerning the real-time transmission of a significant volume of data and the creation of optimal algorithms for processing this data. Nevertheless, this research project has the potential to be economical beneficial over the long run.

8.2.3 Sensor Utilization

Due to the length of the studies, this research did not fully utilize the properties of all the sensors, and the system detection range was limited to only 15 meters which is capped by the LiDAR. On the other hand, a camera is very effective at detecting objects from close to far distances under ideal lighting conditions. It is possible to utilize a camera as an incoming object detector focusing on medium to far range by employing a YOLO detector, and using LiDAR as a close-range detector to further enhance object detection. This approach has the potential to reach close to zero detection failure, as multiple sensors operating at different distances and overlapping regions can be used to enhance detection.

A Ground Segmentation Performance Diagram

The figures in this section show the benchmark results of each ground segmentation method.

A.i Processing Time

This section presents a series of diagrams that show the processing time per frame for each benchmark ground segmentation algorithm.

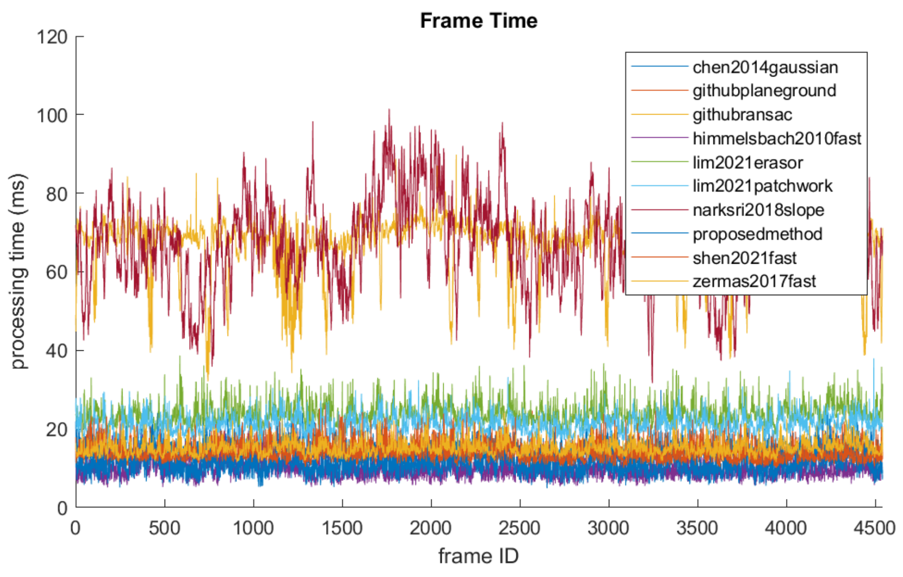


Figure 1: Benchmark time on sequence 00

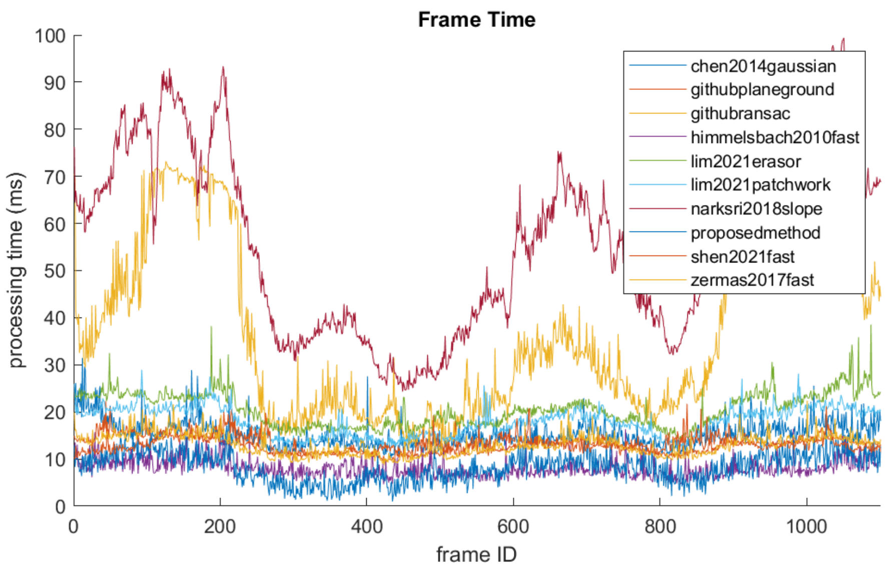


Figure 2: Benchmark time on sequence 01

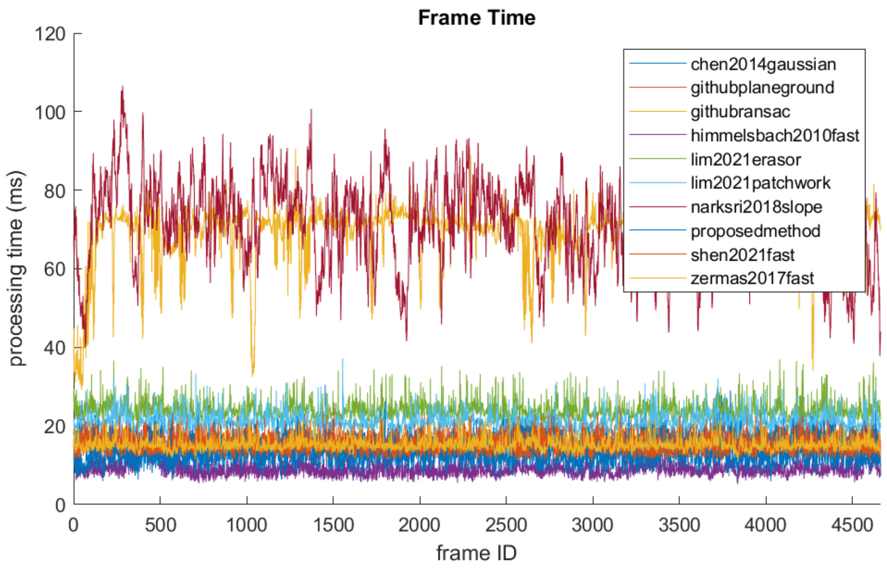


Figure 3: Benchmark time on sequence 02

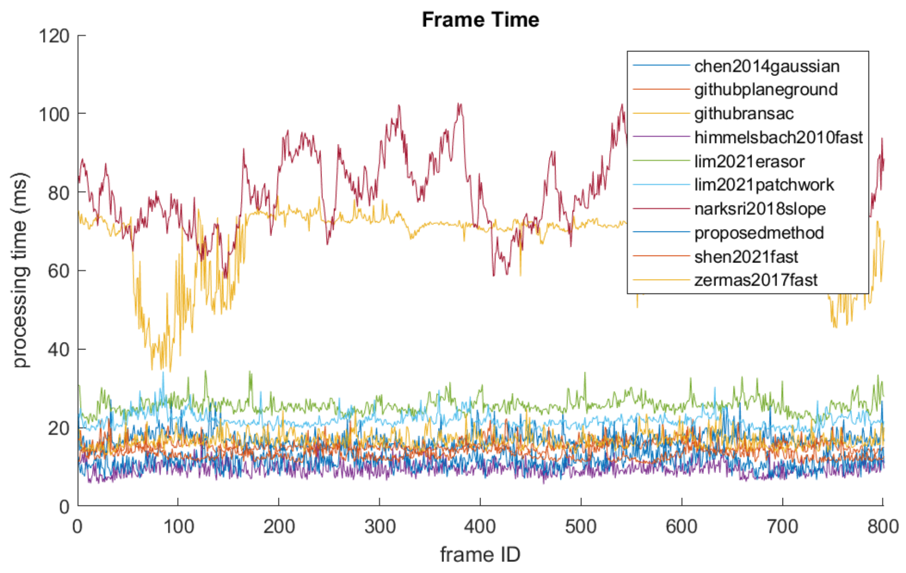


Figure 4: Benchmark time on sequence 03

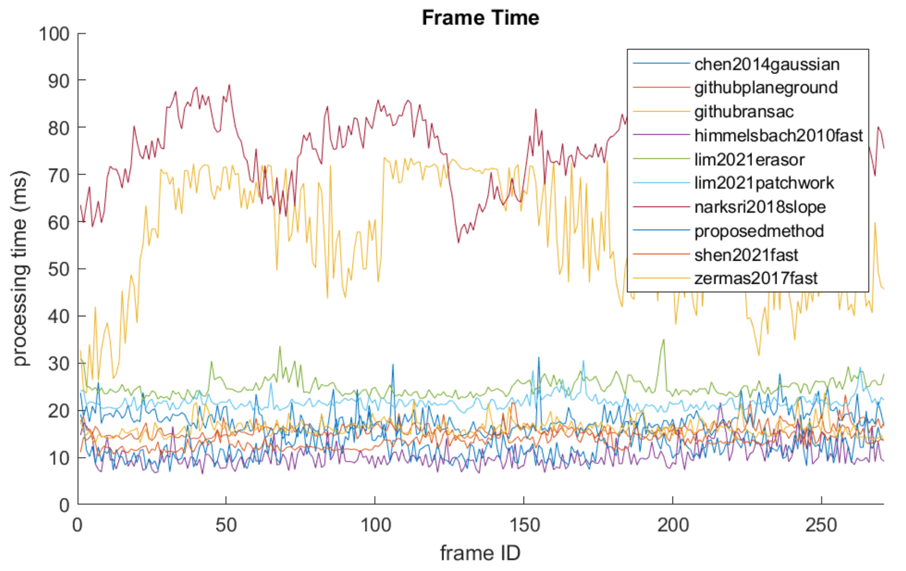


Figure 5: Benchmark time on sequence 04

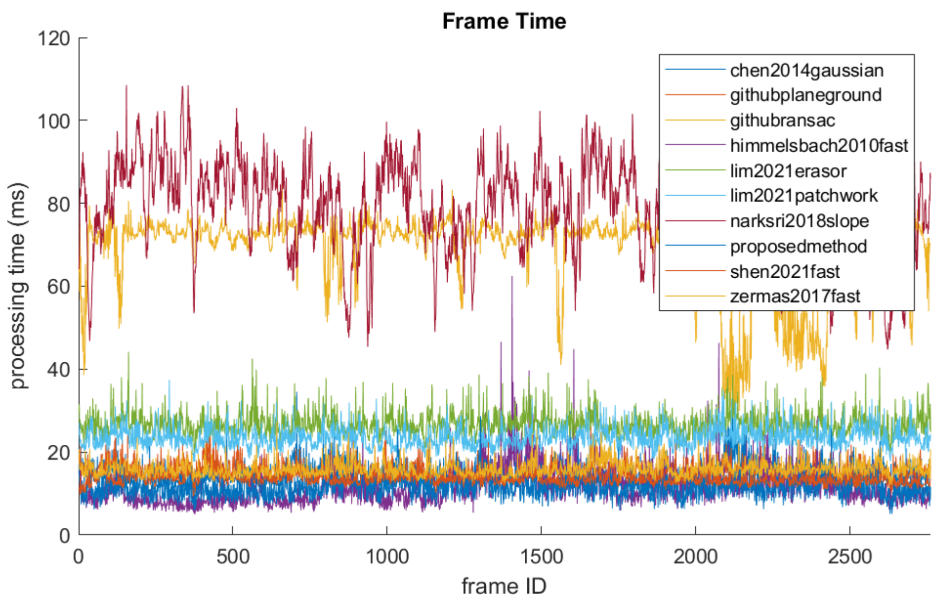


Figure 6: Benchmark time on sequence 05

A.i Precision

This section presents a series of diagrams that show the precision per frame for each benchmark ground segmentation algorithm.

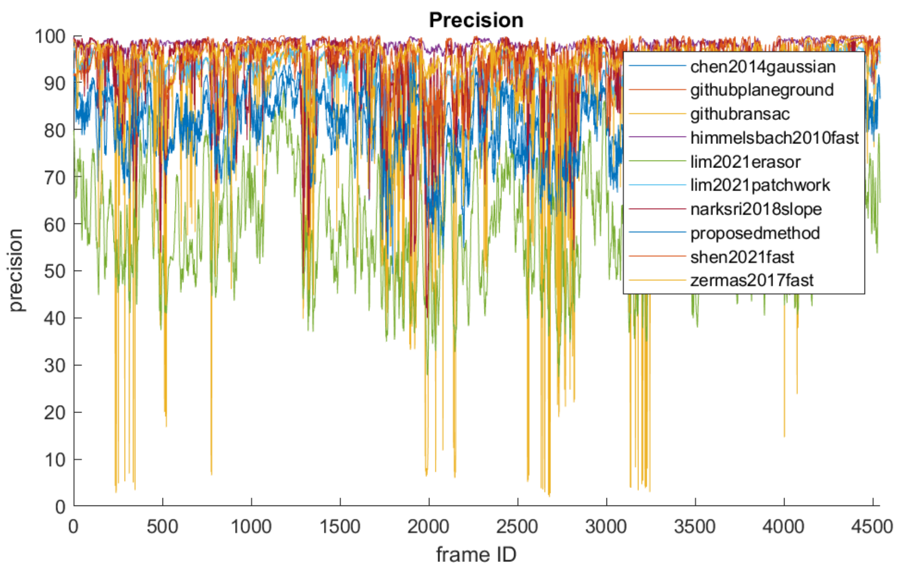


Figure 7: Benchmark precision on sequence 00

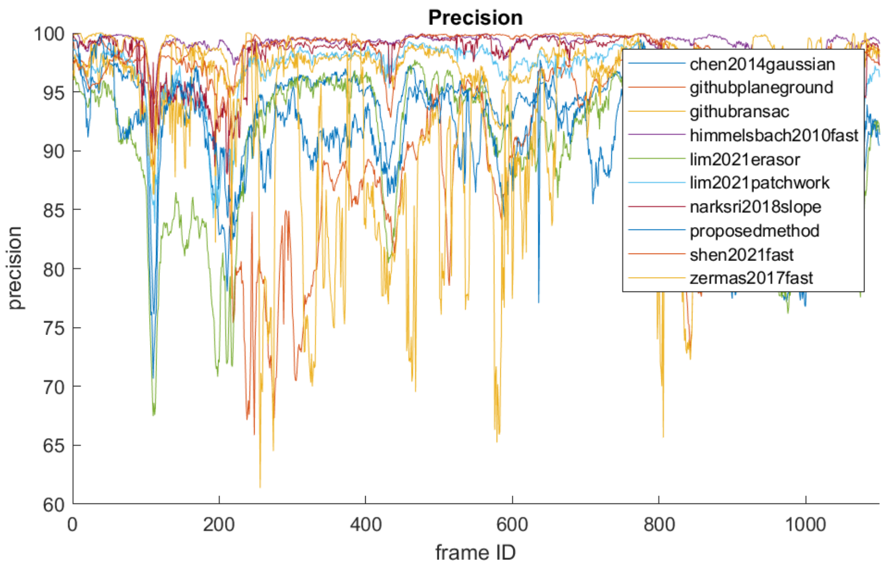


Figure 8: Benchmark precision on sequence 01

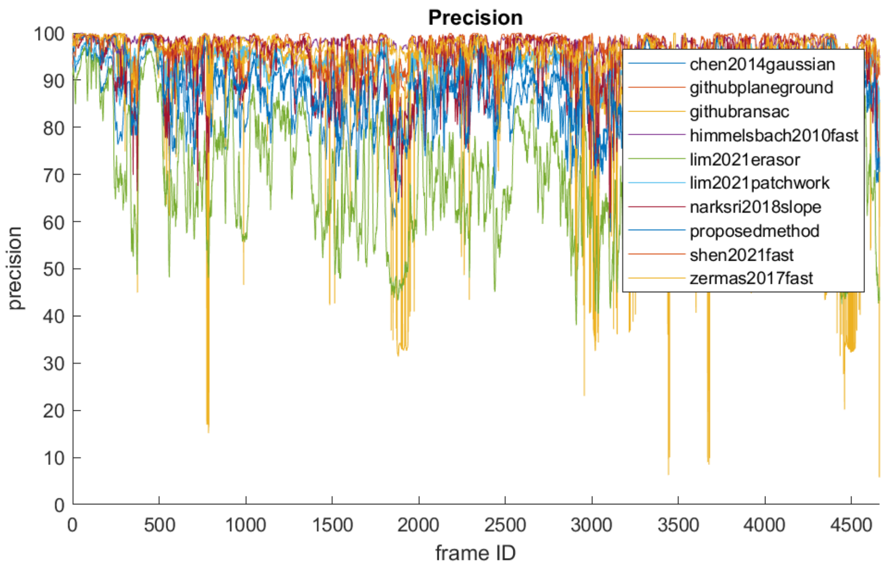


Figure 9: Benchmark precision on sequence 02

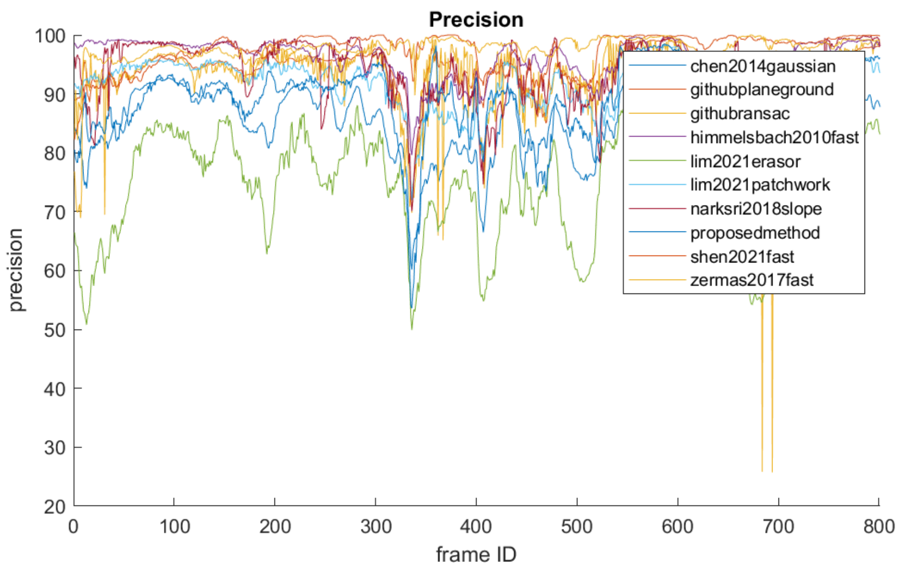


Figure 10: Benchmark precision on sequence 03

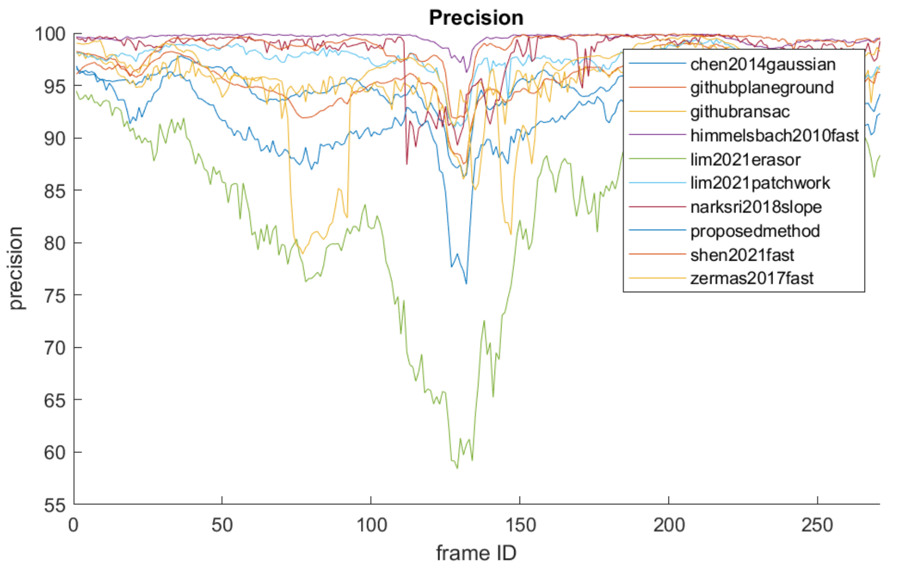


Figure 11: Benchmark precision on sequence 04

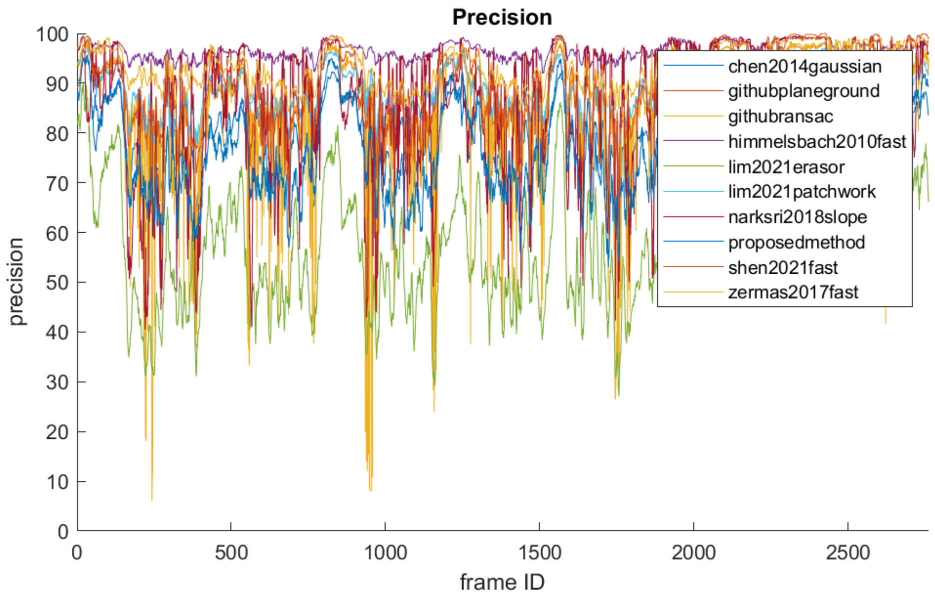


Figure 12: Benchmark precision on sequence 05

A.i Recall

This section presents a series of diagrams that show the recall per frame for each benchmark ground segmentation algorithm.

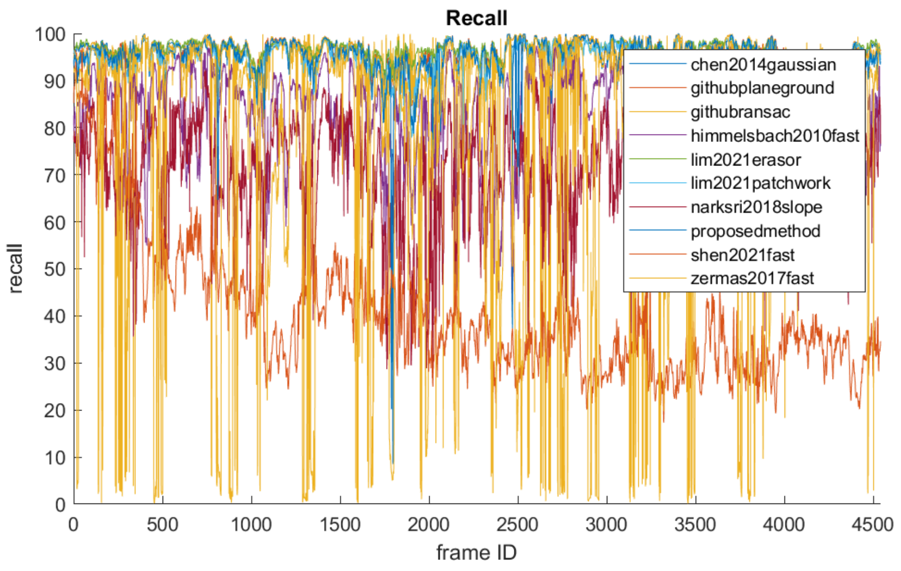


Figure 13: Benchmark recall on sequence 00

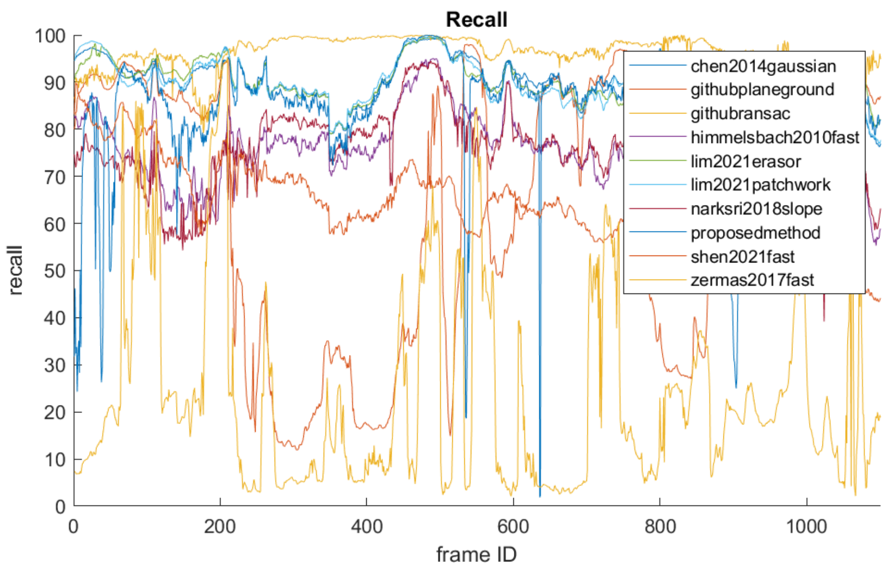


Figure 14: Benchmark recall on sequence 01

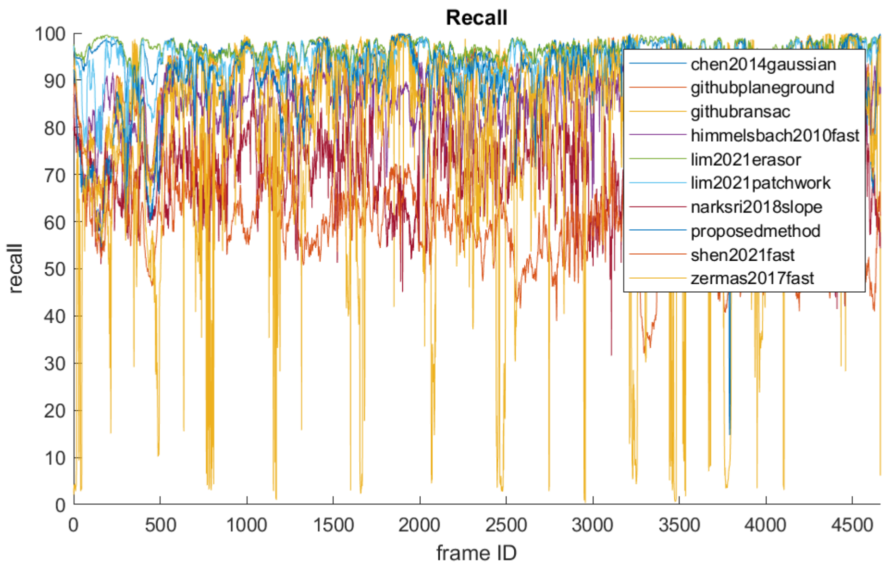


Figure 15: Benchmark recall on sequence 02

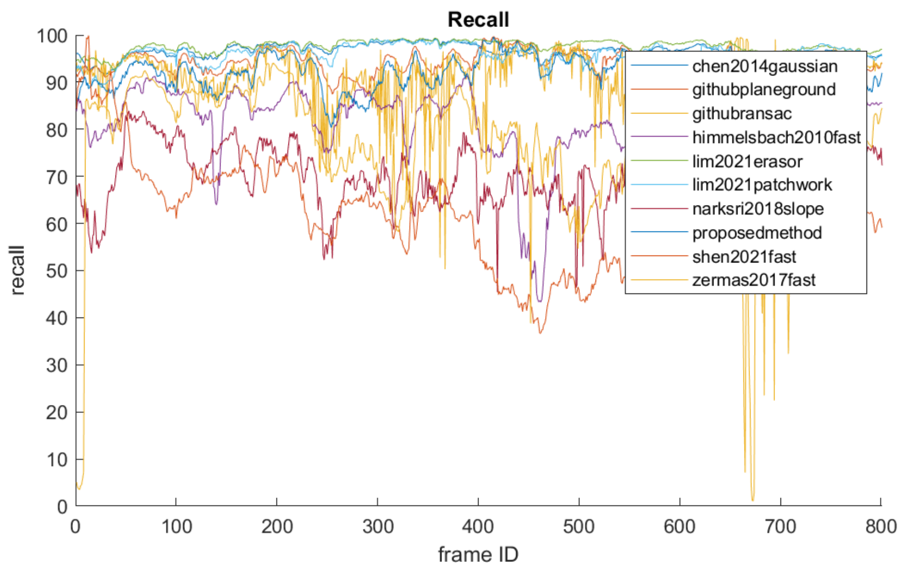


Figure 16: Benchmark recall on sequence 03

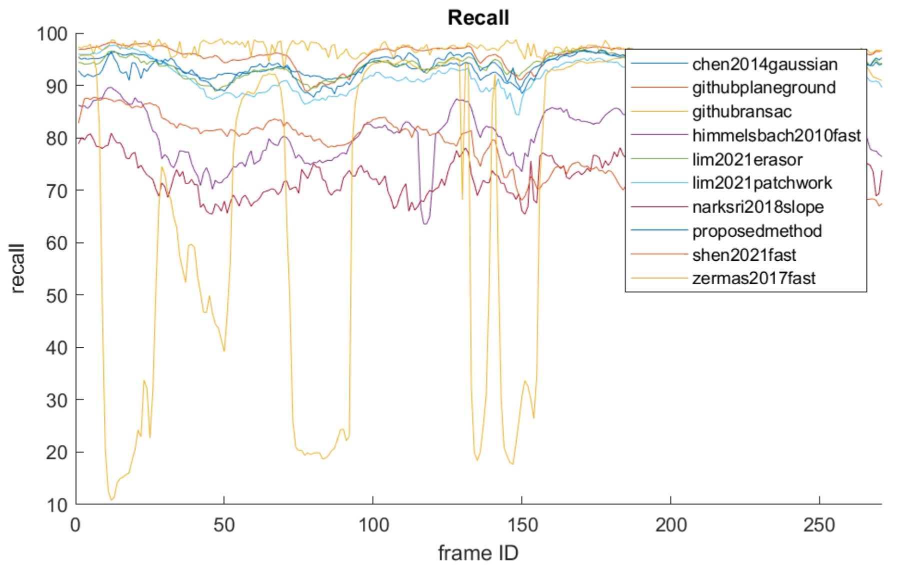


Figure 17: Benchmark recall on sequence 04

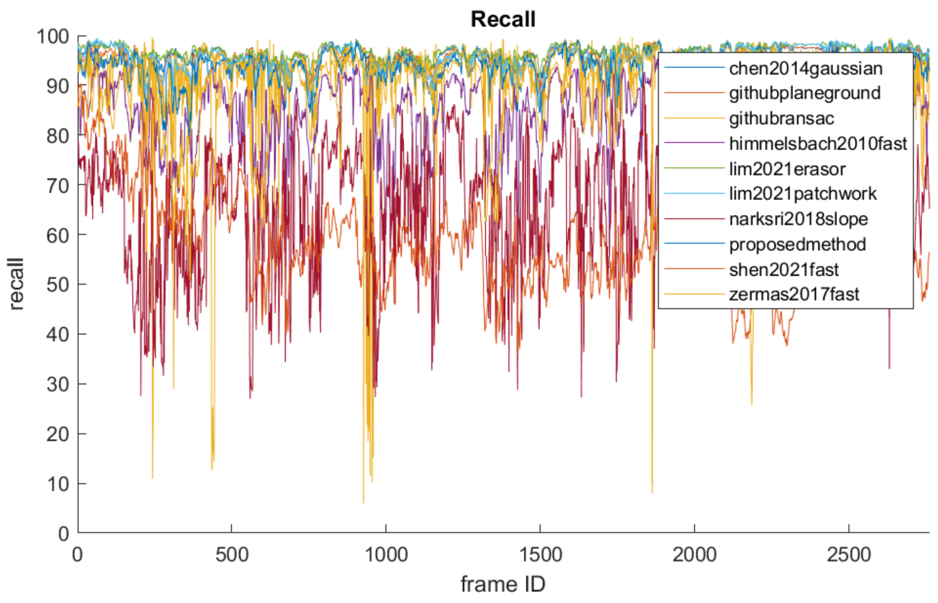


Figure 18: Benchmark recall on sequence 05

B Accuracy and Confusion matrix

For the accuracy measurement, the binary classification method is utilized, starting with the measurement of the following:

True Positives (tp) : These are cases in which the point is correctly predicted as ground.

True Negatives (tn) : These are cases in which the point correctly predicted as non-ground.

False Positives (fp) : These are cases in which the point incorrectly predicted as ground.

False Negatives (fn) : These are cases in which the point incorrectly predicted as non-ground.

Precision Precision is measuring how accurate the result is on identifying one particular object without misidentification.

$$\mathrm{Precision} = \frac{\mathrm{tp}}{\mathrm{tp} + \mathrm{fp}} \tag{1}$$

Recall Recall on the other hand is measuring how accurate the result can identify one particular objects out of all objects.

$$\mathrm{Recall} = \frac{\mathrm{tp}}{\mathrm{tp} + \mathrm{fn}} \tag{2}$$

B.ii Performance Measurement

Performance benchmarking for the classification task is mainly measured in the following scales: precision, recall, f1 and f2 scores.

Fbeta-measure Fbeta-measure score is another popular measurement for analysing the effectiveness of classification neural network.

$$\mathrm{F}_1 = \frac{2 \times \mathrm{Precision} \times \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}} \tag{3}$$

Computed result will indicate the performance of the network scoring from the worst 0.0 to the perfect 1.0.

$$\mathrm{F}_2 = \frac{5 \times \mathrm{Precision} \times \mathrm{Recall}}{4 \times \mathrm{Precision} + \mathrm{Recall}} \tag{4}$$

F2 score is developed to give a better view on the effect of false negative over the false positive.

C Derivative of Complex-Valued Layers

This section presents some examples of computing the derivative of complex-valued neural network layers and performing an analytic check on holomorphic functions.

C.iii Convolutional Layer Back Propagation

$$\mathrm{Conv}_{\mathbf{K},\mathbf{b}}^{t_r,t_c,t_l}(\mathbf{Z}) = \sum_{w=t_l}^{f}\sum_{v=0}^{q-1}\sum_{v=0}^{p-1}\mathbf{K}_{u,v,w}z_{(t_r s_r+u),(t_c s_r+v),(t_l+w)} + \mathbf{b}_{t_r,t_c,w} \tag{5}$$

$\mathbf{Z}$ is a $m \times n \times l$ complex tensor, $\mathbf{K}$ is a $p \times q \times f$ complex tensor, $\mathbf{b}$ is a $r \times s \times f$ complex tensor. t_r, t_c, t_l is the target output position.

$$r = \frac{m - p + 2 \times \text{Padding}}{s_r} + 1 \tag{6}$$

$$s = \frac{n - q + 2 \times \text{Padding}}{s_c} + 1 \tag{7}$$

$$d = l - f + 1 \tag{8}$$

Derivative of the Function:

Analytic Check

$$Conv(x+iy) = \sum\sum\sum \mathbf{K}(x+iy) + \mathbf{b} \tag{9}$$

$$Conv(x+iy) = \sum\sum\sum \mathbf{K}x + \mathbf{b} + i\sum\sum\sum \mathbf{K}y \tag{10}$$

$$u(x+iy) = \sum\sum\sum Kx + b \tag{11}$$

$$v(x+iy) = i\sum Ky \tag{12}$$

$$\frac{\partial u_m}{\partial x_n} = \begin{cases} K_n & \text{if condition is met.} \\ 0 & \text{else} \end{cases} \tag{13}$$

$$\frac{\partial v_m}{\partial y_n} = \begin{cases} K_n & \text{if condition is met.} \\ 0 & \text{else} \end{cases} \tag{14}$$

$$\frac{\partial v}{\partial x} = 0 \tag{15}$$

$$\frac{\partial u}{\partial y} = 0 \tag{16}$$

$$\frac{\partial u_m}{\partial x_n} = \frac{\partial v_m}{\partial y_n}, \frac{\partial v_m}{\partial x_n} = -\frac{\partial u_m}{\partial y_n} \tag{17}$$

$\therefore$ Conv is an analytic function

To understand the derivative of the specific element in the gradient, given a toy example below:

$$x = \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix}, k = \begin{bmatrix} k_{11} & k_{12} \\ k_{21} & k_{22} \end{bmatrix} \tag{18}$$

$$o = \begin{bmatrix} x_{11}k_{11} + x_{12}k_{12} + x_{21}k_{21} + x_{22}k_{22} & x_{12}k_{11} + x_{13}k_{12} + x_{22}k_{21} + x_{23}k_{22} \\ x_{21}k_{11} + x_{22}k_{12} + x_{31}k_{21} + x_{32}k_{22} & x_{22}k_{11} + x_{23}k_{12} + x_{32}k_{21} + x_{33}k_{22} \end{bmatrix} \tag{19}$$

So, $o_{t_r,t_c} = \sum_{u=0}^{q-1}\sum_{v=0}^{p-1} x_{t_r s_r + u, t_c s_c + v} k_{u,v}$, when given a target output o_{t_r,t_c} and the input $x_{\alpha,\beta}$, the respected kernel is related to its position in the filtering window.

So the corresponding kernel is $k_{t_r - \lfloor \frac{\alpha}{s_r} \rfloor + 1, t_c - \lfloor \frac{\beta}{s_c} \rfloor + 1}$.

For example: $o_{2,2}$, and $x_{2,3}$. The respect k should be $k_{(2-\lfloor \frac{2}{1} \rfloor + 1)(3 - \lfloor \frac{2}{1} + 1 \rfloor)}$, which is $k_{1,2}$.

$$\frac{d\text{Conv}}{dz} = \frac{\partial u}{\partial z} + i\frac{\partial v}{\partial z} \tag{20}$$

$$\frac{\partial \text{Conv}_{\mathbf{K},b}^{t_r,t_c,t_l}}{\partial z_{\alpha,\beta,\gamma}} = \begin{cases} K_{t_r - \lfloor \frac{\alpha}{s_r} \rfloor + 1, t_c - \lfloor \frac{\beta}{s_c} \rfloor + 1, t_l - f} & \text{if } 0 < t_r - \lfloor \frac{\alpha}{s_r} \rfloor + 1 \leq p \text{ , } 0 < t_c - \lfloor \frac{\beta}{s_c} \rfloor + 1 \leq q \text{ , } 0 < t_l - f \leq f \\ 0 & \text{else} \end{cases} \tag{21}$$

Compute the derivative respect to **K**

Proof is similar to z, therefore it is skipped.

Similarly, following the example provided earlier, it is necessary to find out $\frac{\partial \text{Conv}_{\mathbf{K},b}^{t_r,t_c,t_l}}{\partial K_{\alpha,\beta,\gamma}}$

$$\frac{\partial \text{Conv}}{\partial K} = \begin{cases} Z_n & \text{if condition is met.} \\ 0 & \text{else} \end{cases} \tag{22}$$

Given output o_{t_r,t_c}, and the kernel $k_{\alpha,\beta}$, the respected input z is $z_{t_r+\alpha s_r-1,t_c+\beta s_c-1}$

$$\frac{\partial \text{Conv}_{\mathbf{K},b}^{t_r,t_c,t_l}}{\partial K_{\alpha,\beta,\gamma}} = \begin{cases} z_{t_r+\alpha s_r-1,t_c+\beta s_c-1,t_l+\gamma} & \text{if } 0 < t_r+\alpha s_r-1 \leq m \text{ , } 0 < t_c+\beta s_c-1 \leq n \text{ , } 0 < t_l+\gamma \leq l \\ 0 & \text{else} \end{cases} \tag{23}$$

Compute the derivative respect to **b**

$$\frac{\partial \text{Conv}_{\mathbf{K},b}^{t_r,t_c,t_l}}{\partial b_{\alpha,\beta,\gamma}} = \begin{cases} 1 & \text{if } \alpha = t_r, \beta = t_c \\ 0 & \text{else} \end{cases} \tag{24}$$

C.iii Dense(Full-Connected) Layer and Derivative

$$\text{FC}_{\mathbf{K}^l,\mathbf{b}^l}(z) = \sum_{i=0}^{m\times n\times d-1} (\mathbf{K}_i^l z_i) + \mathbf{b}^l \tag{25}$$

Back propagation:

Analytic check:

$$\text{FC}(x+iy) = \sum K(x+iy) + b \tag{26}$$

$$\text{FC}(x+iy) = \sum Kx + b + i\sum Ky \tag{27}$$

$$u(x+iy) = \sum Kx + b \tag{28}$$

$$v(x+iy) = i\sum Ky \tag{29}$$

$$\frac{\partial u_m}{\partial x_n} = \begin{cases} K_n & \text{if } n = m \\ 0 & \text{else} \end{cases} \tag{30}$$

$$\frac{\partial v_m}{\partial y_n} = \begin{cases} K_n & \text{if } n = m \\ 0 & \text{else} \end{cases} \tag{31}$$

$$\frac{\partial v_m}{\partial x_n} = 0 \tag{32}$$

$$\frac{\partial u_m}{\partial y_n} = 0 \tag{33}$$

$$\frac{\partial u_m}{\partial x_n} = \frac{\partial v_m}{\partial y_n}, \frac{\partial v_m}{\partial x_n} = -\frac{\partial u_m}{\partial y_n} \tag{34}$$

$\therefore$ FC is an analytic function

$$\frac{d\text{FC}}{dz} = \frac{\partial u}{\partial z} + i\frac{\partial v}{\partial z} \tag{35}$$

$$\frac{\partial \text{FC}_{\mathbf{K}^l,b^l}(z_m)}{\partial z_n} = \begin{cases} K_n^l & \text{if } n = m \\ 0 & \text{else} \end{cases} \tag{36}$$

$$\nabla FC_{\mathbf{K}^l}(z) = \begin{bmatrix} K_1^l & 0 & \dots & 0 \\ 0 & K_2^l & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & K_n^l \end{bmatrix} \tag{37}$$

Compute dFC w.r.t. K

Analytic proof follows the same procedure as $\frac{dFC}{dz}$.

$$\nabla FC_z(K) = \begin{bmatrix} z_1^l & 0 & \dots & 0 \\ 0 & z_2^l & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & z_n^l \end{bmatrix} \tag{38}$$

Compute dFC w.r.t b

Analytic proof skipped.

$$\nabla FC_z(b) = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix} \tag{39}$$

C.iii Pooling Layer and Derivative

Pooling layer acts differently when perform back propagation.

Given a function $f(z)$ for the input z_{ij}, the function writes $f_{st} = a_{mn}z_{ij}$, where

$$a = \begin{cases} 1 & \text{if conditions are met} \\ 0 & \text{else} \end{cases} \tag{40}$$

Based on the given example, during the max pooling computing process, another tuple-matrix is saved to record the position where the input is taken. For this example, the tuple-matrix appears as follows:

So given above, the condition for a to be 1 is when tuple (m,n) matches the A_{st}, else will be 0.

$f_{st} = z_{ij}$ is analytic, proof has been done, the derivative is $\frac{\partial f_{st}}{\partial z} = 1$

Given a simple example below:

$$z = \begin{bmatrix} 1+i & 2+i & 3+i & 4+i \\ 5+i & 6+i & 7+i & 8+i \\ 9+i & 10+i & 11+i & 12+i \\ 13+i & 14+i & 15+i & 16+i \end{bmatrix} \tag{41}$$

Say there is a max pooling with stride of 1, and pooling window 2×2.

$$o = \begin{bmatrix} 6+i & 7+i & 8+i \\ 10+i & 11+i & 12+i \\ 14+i & 15+i & 16+i \end{bmatrix} \tag{42}$$

$$A = \begin{bmatrix} 2,2 & 2,3 & 2,4 \\ 3,2 & 3,3 & 3,4 \\ 4,2 & 4,3 & 4,4 \end{bmatrix} \tag{43}$$

C.iii Complex-valued Modified Softmax Function and Derivative

$$\text{Softmax}_N(z_i) = \frac{e^{\text{Re}(z_i)^2+\text{Im}(z_i)^2-s}}{\sum_{j=1}^{N} e^{\text{Re}(z_j)^2+\text{Im}(z_j)^2-s}} \tag{44}$$

Since the function is defined as $f : \mathbb{C} \to \mathbb{R}$, therefore, it can be treated as a real number function with 2-tuples inputs using Wirtinger Calculus. To simplify the calculation, the partial derivative is applied to the real part and imaginary part of the complex number separately, and then they are combined together. s is $\text{Max}(\|z\|^2)$.

$$\nabla \text{Softmax}(z_\text{r}, z_\text{i}) = \begin{bmatrix} \frac{\partial \text{Softmax}(z_\text{r}, z_\text{i})}{\partial z_\text{r}} \\ \frac{\partial \text{Softmax}(z_\text{r}, z_\text{i})}{\partial z_\text{i}} \end{bmatrix} \tag{45}$$

$$\frac{\partial \text{Softmax}(z_{\text{r}m}, z_{\text{i}m})}{\partial z_{\text{r}n}} = \frac{\partial \frac{e^{z_{\text{r}m}^2+z_{\text{i}m}^2-s}}{\sum_{j=1}^{N} e^{z_{\text{r}j}^2+z_{\text{i}j}^2-s}}}{\partial z_{\text{r}n}} \tag{46}$$

Using Quotient rule:

$$f(x) = \frac{g(x)}{h(x)} \tag{47}$$

$$f'(x) = \frac{g'(x)h(x) - h'(x)g(x)}{[h(x)]^2} \tag{48}$$

$$g_m = e^{z_{\text{r}m}^2+z_{\text{i}m}^2-m} \tag{49}$$

$$h_m = \sum_{j=1}^{N} e^{z_{\text{r}j}^2+z_{\text{i}j}^2-m} \tag{50}$$

$$\frac{\partial \text{Softmax}(z_{\text{r}m}, z_{\text{i}m})}{\partial z_{\text{r}n}} = \frac{g' \sum_{j=1}^{N} e^{z_{\text{r}j}^2+z_{\text{i}j}^2-s} - e^{z_{\text{r}n}^2+z_{\text{i}n}^2-s} e^{z_{\text{r}m}^2+z_{\text{i}m}^2-s}}{[\sum_{j=1}^{N} e^{z_{\text{r}j}^2+z_{\text{i}j}^2-s}]^2} \tag{51}$$

$$\frac{\partial g(z_{\text{r}m}^2 + z_{\text{i}m}^2)}{\partial z_{\text{r}n}} = \begin{cases} 2z_{\text{r}n} e^{z_{\text{r}n}^2+z_{\text{i}n}^2-s} & \text{if } n = m \\ 0 & \text{else} \end{cases} \tag{52}$$

Now consider 2 cases:

When $(n = m)$

$$\frac{\partial \text{Softmax}(z_{rm}, z_{im})}{\partial z_{rn}} = \frac{2z_{rn}e^{z_{rn}^2+z_{in}^2-s}\sum_{j=1}^{N} e^{z_{rj}^2+z_{ij}^2-s} - e^{z_{rm}^2+z_{im}^2-s}e^{z_{rm}^2+z_{im}^2-s}}{[\sum_{j=1}^{N} e^{z_{rj}^2+z_{ij}^2} - s]^2} \tag{53}$$

$$\frac{\partial \text{Softmax}(z_{rm}, z_{im})}{\partial z_{rn}} = \frac{e^{z_{rn}^2+z_{in}^2-s}(2z_{rn}\sum_{j=1}^{N} e^{z_{rj}^2+z_{ij}^2-s} - e^{z_{rm}^2+z_{im}^2-s})}{[\sum_{j=1}^{N} e^{z_{rj}^2+z_{ij}^2-s}]^2} \tag{54}$$

$$\frac{\partial \text{Softmax}(z_{rm}, z_{im})}{\partial z_{rn}} = \frac{e^{z_{rn}^2+z_{in}^2-s}}{\sum_{j=1}^{N} e^{z_{rj}^2+z_{ij}^2-s}} \frac{(2z_{rn}\sum_{j=1}^{N} e^{z_{rj}^2+z_{ij}^2-s} - e^{z_{rm}^2+z_{im}^2-s})}{\sum_{j=1}^{N} e^{z_{rj}^2+z_{ij}^2-s}} \tag{55}$$

$$\frac{\partial \text{Softmax}(z_{rm}, z_{im})}{\partial z_{rn}} = \frac{e^{z_{rn}^2+z_{in}^2-s}}{\sum_{j=1}^{N} e^{z_{rj}^2+z_{ij}^2-s}}[\frac{(2z_{rn}\sum_{j=1}^{N} e^{z_{rj}^2+z_{ij}^2-s}}{\sum_{j=1}^{N} e^{z_{rj}^2+z_{ij}^2-s}} - \frac{e^{z_{rm}^2+z_{im}^2-s})}{\sum_{j=1}^{N} e^{z_{rj}^2+z_{ij}^2-s}}] \tag{56}$$

$$\frac{\partial \text{Softmax}(z_{rm}, z_{im})}{\partial z_{rn}} = \text{Softmax}(z_{rn}, z_{in})[2z_{rn} - \text{Softmax}(z_{rm}, z_{im})] \tag{57}$$

When $(n \neq m)$

$$\frac{\partial \text{Softmax}(z_{rm}, z_{im})}{\partial z_{rn}} = \frac{-e^{z_{rn}^2+z_{in}^2-s}e^{z_{rm}^2+z_{im}^2-s}}{[\sum_{j=1}^{N} e^{z_{rj}^2+z_{ij}^2-s}]^2} \tag{58}$$

$$\frac{\partial \text{Softmax}(z_{rm}, z_{im})}{\partial z_{rn}} = \frac{-e^{z_{rn}^2+z_{in}^2-s}}{\sum_{j=1}^{N} e^{z_{rj}^2+z_{ij}^2-s}} \frac{e^{z_{rm}^2+z_{im}^2-s}}{\sum_{j=1}^{N} e^{z_{rj}^2+z_{ij}^2-s}} \tag{59}$$

$$\frac{\partial \text{Softmax}(z_{rm}, z_{im})}{\partial z_{rn}} = -\text{Softmax}(z_{rn}, z_{in})\text{Softmax}(z_{rm}, z_{im}) \tag{60}$$

So using kronecker delta δ_{mn}

$$\delta_{mn} = \begin{cases} 1 & \text{if } n = m \\ 0 & \text{else} \end{cases} \tag{61}$$

$$\frac{\partial \text{Softmax}(z_{rm}, z_{im})}{\partial z_{rn}} = \text{Softmax}(z_{rn}, z_{in})[\delta_{mn}2z_{rn} - \text{Softmax}(z_{rm}, z_{im})] \tag{62}$$

As for the imaginary part, the derivative is similar.

$$\frac{\partial \text{Softmax}(z_{rm}, z_{im})}{\partial z_{in}} = \text{Softmax}(z_{rn}, z_{in})[\delta_{mn}2z_{in} - \text{Softmax}(z_{rm}, z_{im})] \tag{63}$$

Plug everything back

$$\frac{\partial \mathrm{SM}(z_{\mathrm{r}}, z_{\mathrm{i}})}{\partial (z_{\mathrm{r}})} = \begin{bmatrix} \mathrm{SM}(z_{\mathrm{r}1}, z_{\mathrm{i}1})[2z_{\mathrm{r}1} - \mathrm{SM}(z_{\mathrm{r}1}, z_{\mathrm{i}1})] & \cdots & \mathrm{SM}(z_{\mathrm{r}1}, z_{\mathrm{i}1})[-\mathrm{SM}(z_{\mathrm{r}N}, z_{\mathrm{i}N})] \\ \vdots & \ddots & \vdots \\ \mathrm{SM}(z_{\mathrm{r}N}, z_{\mathrm{i}N})[-\mathrm{SM}(z_{\mathrm{r}1}, z_{\mathrm{i}1})] & \cdots & \mathrm{SM}(z_{\mathrm{r}N}, z_{\mathrm{i}N})[2z_{\mathrm{r}N} - \mathrm{SM}(z_{\mathrm{r}N}, z_{\mathrm{i}N})] \end{bmatrix} \tag{64}$$

$$\frac{\partial \mathrm{SM}(z_{\mathrm{r}}, z_{\mathrm{i}})}{\partial (z_{\mathrm{i}})} = \begin{bmatrix} \mathrm{SM}(z_{\mathrm{r}1}, z_{\mathrm{i}1})[2z_{\mathrm{i}1} - \mathrm{SM}(z_{\mathrm{r}1}, z_{\mathrm{i}1})] & \cdots & \mathrm{SM}(z_{\mathrm{r}1}, z_{\mathrm{i}1})[-\mathrm{SM}(z_{\mathrm{r}N}, z_{\mathrm{i}N})] \\ \vdots & \ddots & \vdots \\ \mathrm{SM}(z_{\mathrm{r}N}, z_{\mathrm{i}N})[-\mathrm{SM}(z_{\mathrm{r}1}, z_{\mathrm{i}1})] & \cdots & \mathrm{SM}(z_{\mathrm{r}N}, z_{\mathrm{i}N})[2z_{\mathrm{i}N} - \mathrm{SM}(z_{\mathrm{r}N}, z_{\mathrm{i}N})] \end{bmatrix} \tag{65}$$

Put in complex form

$$\frac{\partial \mathrm{Softmax}(z_m)}{\partial z_n} = \mathrm{Softmax}(z_n)[\delta_{\mathrm{mn}} 2\Re(z_n) - \mathrm{Softmax}(z_m)] + i\mathrm{Softmax}(z_n)[\delta_{\mathrm{mn}} 2\Im(z_n) - \mathrm{Softmax}(z_m)] \tag{66}$$

Bibliography

[1] Siddharth Agarwal, Ankit Vora, Gaurav Pandey, Wayne Williams, Helen Kourous, and James McBride. Ford multi-av seasonal dataset, 2020.

[2] Ryan Ahmed, Mohammed El Sayed, S. Andrew Gadsden, Jimi Tjong, and Saeid Habibi. Artificial neural network training utilizing the smooth variable structure filter estimation strategy. *Neural Computing and Applications*, 27(3):537–548, April 2016.

[3] Eren Erdal Aksoy, Saimir Baci, and Selcuk Cavdar. Salsanet: Fast road and vehicle segmentation in lidar point clouds for autonomous driving. In *2020 IEEE intelligent vehicles symposium (IV)*, pages 926–932. IEEE, 2020.

[4] AllaboutCircuits. Solid-state lidar is coming to an autonomous vehicle near you - news, https://www.allaboutcircuits.com/news/solid-state-lidar-is-coming-to-an-autonomous-vehicle-near-you/.

[5] Md. Faijul Amin, Md. Monirul Islam, and Kazuyuki Murase. Ensemble of single-layered complex-valued neural networks for classification tasks. *Neurocomputing*, 72(10–12):2227–2234, June 2009.

[6] Md. Faijul Amin and Kazuyuki Murase. Single-layered complex-valued neural network for real-valued classification problems. *Neurocomputing*, 72(4–6):945–955, January 2009.

[7] Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. In *International conference on machine learning*, pages 1120–1128. PMLR, 2016.

[8] Alireza Asvadi, Luis Garrote, Cristiano Premebida, Paulo Peixoto, and Urbano J Nunes. Multimodal vehicle detection: fusing 3d-lidar and color camera data. *Pattern Recognition Letters*, 115:20–29, 2018.

[9] Gerardo Atanacio-Jiménez, José-Joel González-Barbosa, Juan B. Hurtado-Ramos, Francisco J. Ornelas-Rodríguez, Hugo Jiménez-Hernández, Teresa García-Ramirez, and Ricardo González-Barbosa. Lidar velodyne hdl-64e calibration using pattern planes. *International Journal of Advanced Robotic Systems*, 8(5):59, November 2011.

[10] Bob Atkins. *Fish eye example*. 2017.

[11] Pierre Baldi and Zhiqin Lu. Complex-valued autoencoders. *Neural Networks*, 33:136–147, September 2012.

[12] Haris Baltzakis, Antonis Argyros, and Panos Trahanias. Fusion of laser and visual data for robot motion planning and collision avoidance. *Machine Vision and Applications*, 15:92–100, 2003.

[13] C Bradford Barber, David P Dobkin, and Hannu Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software (TOMS)*, 22(4):469–483, 1996.

[14] Dan Barnes, Matthew Gadd, Paul Murcutt, Paul Newman, and Ingmar Posner. The oxford radar robotcar dataset: A radar extension to the oxford robotcar dataset. *arXiv preprint arXiv: 1909.01300*, 2019.

[15] Jose Agustin Barrachina, Chenfang Ren, Christele Morisseau, Gilles Vieillard, and J-P Ovarlez. Complex-valued vs. real-valued neural networks for classification perspectives: An example on non-circular data. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2990–2994. IEEE, 2021.

[16] Jan C Becker. Fusion of heterogeneous sensors for the guidance of an autonomous vehicle. In *Proceedings of the Third International Conference on Information Fusion*, volume 2, pages WED5–11. IEEE, 2000.

[17] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, C. Stachniss, and J. Gall. SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences. In *Proc. of the IEEE/CVF International Conf. on Computer Vision (ICCV)*, 2019.

[18] Siavash Arjomand Bigdeli and Matthias Zwicker. Image restoration using autoencoding priors. *Proceedings of the 13th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, 2018.

[19] Josep Miquel Biosca and José Luis Lerma. Unsupervised robust planar segmentation of terrestrial laser scanner point clouds based on fuzzy clustering methods. *ISPRS Journal of Photogrammetry and Remote Sensing*, 63(1):84–98, 2008.

[20] Sebastian Blanco. Report: Tesla autopilot involved in 736 crashes since 2019, June 2023.

[21] Erik Blasch, Ivan Kadar, John Salerno, Mieczyslaw M Kokar, Subrata Das, Gerald M Powell, Daniel D Corkill, and Enrique H Ruspini. Issues and challenges of knowledge representation and reasoning methods in situation assessment (level 2 fusion). *Signal Processing, Sensor Fusion, and Target Recognition XV*, 6235:355–368, 2006.

[22] Erik Blasch, Alan Steinberg, Subrata Das, James Llinas, Chee Chong, Otto Kessler, Ed Waltz, and Frank White. Revisiting the jdl model for information exploitation. In *Proceedings of the 16th International Conference on Information Fusion*, pages 129–136. IEEE, 2013.

[23] Erik P Blasch and Susan Plano. Jdl level 5 fusion model: user refinement issues and applications in group tracking. In *Signal processing, sensor fusion, and target recognition XI*, volume 4729, pages 270–279. SPIE, 2002.

[24] Erik P Blasch and Susan Plano. Level 5: user refinement to aid the fusion process. In *Multisensor, Multisource Information Fusion: Architectures, Algorithms, and Applications 2003*, volume 5099, pages 288–297. SPIE, 2003.

[25] Igor Bogoslavskyi and Cyrill Stachniss. Fast range image-based segmentation of sparse 3d laser scans for online operation. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 163–169. IEEE, 2016.

[26] Ryan G. Brazeal, Benjamin E. Wilkinson, and Hartwig H. Hochmair. A rigorous observation model for the risley prism-based livox mid-40 lidar sensor. *Sensors*, 21(14):4722, July 2021.

[27] Duane C Brown. Close-range camera calibration. page 12.

[28] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. *arXiv preprint arXiv:1903.11027*, 2019.

[29] Transport Canada. Canadian motor vehicle traffic collision statistics: 2018, December 2020.

[30] Hubert Cecotti and Axel Graeser. Convolutional neural network with embedded fourier transform for eeg classification. In *2008 19th International Conference on Pattern Recognition*, page 1–4. IEEE, December 2008.

[31] CEIC. *United States Number of Registered Vehicles*. 2023.

[32] Tongtong Chen, Bin Dai, Ruili Wang, and Daxue Liu. Gaussian-process-based real-time ground segmentation for autonomous land vehicles. *Journal of Intelligent & Robotic Systems*, 76:563–582, 2014.

[33] Xieyuanli Chen, Ignacio Vizzo, Thomas Läbe, Jens Behley, and Cyrill Stachniss. Range image-based lidar localization for autonomous vehicles. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5802–5808. IEEE, 2021.

[34] Hyun-Woong Cho, Sungdo Choi, Young-Rae Cho, and Jongseok Kim. Complex-valued channel attention and application in ego-velocity estimation with automotive radar. *IEEE Access*, 9:17717–17727, 2021.

[35] Hyunggi Cho, Young-Woo Seo, BVK Vijaya Kumar, and Ragunathan Raj Rajkumar. A multi-sensor fusion system for moving object detection and tracking in urban driving environments. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1836–1843. IEEE, 2014.

[36] Yukyung Choi, Namil Kim, Soonmin Hwang, Kibaek Park, Jae Shin Yoon, Kyounghwan An, and In So Kweon. Kaist multi-spectral day/night data set for autonomous and assisted driving. *IEEE Transactions on Intelligent Transportation Systems*, 19(3):934–948, 2018.

[37] Phuong Chu, Seoungjae Cho, Sungdae Sim, Kiho Kwak, and Kyungeun Cho. A fast ground segmentation method for 3d point cloud. *J. Inf. Process. Syst.*, 13(3):491–499, 2017.

[38] Phuong Minh Chu, Seoungjae Cho, Jisun Park, Simon Fong, and Kyungeun Cho. Enhanced ground segmentation method for lidar point clouds in human-centric autonomous robot systems. *Human-centric Computing and Information Sciences*, 9(1):1–14, 2019.

[39] A. E. Conrady. Decentred lens-systems. *Monthly Notices of the Royal Astronomical Society*, 79(5):384–390, March 1919.

[40] Diana Thomson La Corte and Yi Ming Zou. Newton's method backpropagation for complex-valued holomorphic multilayer perceptrons. *arXiv:1406.5254 [math]*, June 2014. arXiv: 1406.5254.

[41] National Safety Council. *Injury Facts*. April 2023.

[42] T. Dang, C. Hoffmann, and C. Stiller. Fusing optical flow and stereo disparity for object tracking. In *Proceedings. The IEEE 5th International Conference on Intelligent Transportation Systems*, pages 112–117, 2002.

[43] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[44] James F. Dougherty. Ransac gpf, 2019.

[45] Bertrand Douillard, J Underwood, Narek Melkumyan, S Singh, Shrihari Vasudevan, C Brunner, and A Quadros. Hybrid elevation maps: 3d surface models for segmentation. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1532–1538. IEEE, 2010.

[46] Bertrand Douillard, James Underwood, Noah Kuntz, Vsevolod Vlaskine, Alastair Quadros, Peter Morton, and Alon Frenkel. On the segmentation of 3d lidar point clouds. In *2011 IEEE International Conference on Robotics and Automation*, pages 2798–2805. IEEE, 2011.

[47] Infiniti Electro-Optic Infiniti Electro-Optic. *electromagnetic spectrum*. Infiniti Electro-Optics, November 2021.

[48] Wilfried Elmenreich. An introduction to sensor fusion. *Vienna University of Technology, Austria*, 502:1–28, 2002.

[49] Hossam El-Rewaidy, Ahmed S. Fahmy, Farhad Pashakhanloo, Xiaoying Cai, Selcuk Kucukseymen, Ibolya Csecs, Ulf Neisius, Hassan Haji-Valizadeh, Bjoern Menze, and Reza Nezafat. Multi-domain convolutional neural network (md-cnn) for radial reconstruction of dynamic cardiac mri. *Magnetic Resonance in Medicine*, 85(3):1195–1208, March 2021.

[50] Martin Ester, Hans-Peter Kriegel, and Xiaowei Xu. Knowledge discovery in large spatial databases: Focusing techniques for efficient class identification. *Advances in Spatial Databases Lecture Notes in Computer Science*, page 67–82, 1995.

[51] Sagi Filin. Surface clustering from airborne laser scanning data. In *ISPRS Commission III, Symposium 2002 September 9-13, 2002, Graz, Austria*, 2002.

[52] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[53] Eric R Fossum and Donald B Hondongwa. A review of the pinned photodiode for ccd and cmos image sensors. *IEEE Journal of the electron devices society*, 2014.

[54] fvisin. *Transposed convolution animations*. 2019.

[55] Jingkun Gao, Bin Deng, Yuliang Qin, Hongqiang Wang, and Xiang Li. Enhanced radar imaging using a complex-valued convolutional neural network. *IEEE Geoscience and Remote Sensing Letters*, 16(1):35–39, January 2019.

[56] A Geiger, P Lenz, C Stiller, and R Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, September 2013.

[57] George M Georgiou and Cris Koutsougeras. Complex domain backpropagation. *IEEE transactions on Circuits and systems II: analog and digital signal processing*, 39(5):330–334, 1992.

[58] Jakob Geyer, Yohannes Kassahun, Mentar Mahmudi, Xavier Ricou, Rupesh Durgesh, Andrew S Chung, Lorenz Hauswald, Viet Hoang Pham, Maximilian Mühlegg, Sebastian Dorn, et al. A2d2: Audi autonomous driving dataset. *arXiv preprint arXiv:2004.06320*, 2020.

[59] Michael Giering, Vivek Venugopalan, and Kishore Reddy. Multi-modal sensor registration for vehicle perception via deep neural networks. *2015 IEEE High Performance Extreme Computing Conference (HPEC)*, 2015.

[60] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. page 8.

[61] Raffaele Gravina, Parastoo Alinia, Hassan Ghasemzadeh, and Giancarlo Fortino. Multi-sensor fusion in body sensor networks: State-of-the-art and research challenges. *Information Fusion*, 35:68–80, 2017.

[62] Pamela M Greenwood, John K Lenneman, and Carryl L Baldwin. Advanced driver assistance systems (adas): Demographics, preferred sources of information, and accuracy of adas knowledge. *Transportation research part F: traffic psychology and behaviour*, 86:131–150, 2022.

[63] Ayman Habib, Ki In Bang, Ana Paula Kersting, and Jacky Chow. Alternative methodologies for lidar system calibration. *Remote Sensing*, 2(3):874–907, March 2010.

[64] Saeid Habibi. The smooth variable structure filter. *Proceedings of the IEEE*, 95(5):1026–1059, 2007.

[65] Chris Harris, Mike Stephens, et al. A combined corner and edge detector. In *Alvey vision conference*, volume 15, pages 10–5244. Citeseer, 1988.

[66] 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*, page 1106–1112, San Juan, Puerto Rico, 1997. IEEE Comput. Soc.

[67] Michael Himmelsbach, Felix V Hundelshausen, and H-J Wuensche. Fast segmentation of 3d point clouds for ground vehicles. In *2010 IEEE Intelligent Vehicles Symposium*, pages 560–565. IEEE, 2010.

[68] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv:1207.0580 [cs]*, July 2012. arXiv: 1207.0580.

[69] Akira Hirose. *Complex-Valued Neural Networks: Advances and Applications*. 2013.

[70] Eckhard Hitzer. Non-constant bounded holomorphic functions of hyperbolic numbers-candidates for hyperbolic activation functions. *arXiv preprint arXiv:1306.1653*, 2013.

[71] J. Houston, G. Zuidhof, L. Bergamini, Y. Ye, A. Jain, S. Omari, V. Iglovikov, and P. Ondruska. One thousand and one hours: Self-driving motion prediction dataset. `https://level-5.global/level5/data/`, 2020.

[72] J.-W. Hsieh, S.-H. Yu, Y.-S. Chen, and W.-F. Hu. Automatic traffic surveillance system for vehicle tracking and classification. *IEEE Transactions on Intelligent Transportation Systems*, 7(2):175–187, 2006.

[73] Xiao Hu, F Sergio A Rodriguez, and Alexander Gepperth. A multi-modal system for road detection and segmentation. In *2014 IEEE Intelligent Vehicles Symposium Proceedings*, pages 1365–1370. IEEE, 2014.

[74] Xinyu Huang, Peng Wang, Xinjing Cheng, Dingfu Zhou, Qichuan Geng, and Ruigang Yang. The apolloscape open dataset for autonomous driving and its application. *IEEE transactions on pattern analysis and machine intelligence*, 42(10):2702–2719, 2019.

[75] Thu Nguyen Huu and Sejin Lee. Development of volumetric image descriptor for urban object classification using 3d lidar based on convolutional neural network. In *2022 22nd International Conference on Control, Automation and Systems (ICCAS)*, pages 984–989. IEEE, 2022.

[76] Soonmin Hwang, Jaesik Park, Namil Kim, Yukyung Choi, and In So Kweon. Multispectral pedestrian detection: Benchmark dataset and baseline. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[77] Y. Iiguni, H. Sakai, and H. Tokumaru. A real-time learning algorithm for a multilayered neural network based on the extended kalman filter. *IEEE Transactions on Signal Processing*, 40(4):959–966, 1992.

[78] Mahmoud Ismail, Mina Attari, Saeid Habibi, and Samir Ziada. Estimation theory and neural networks revisited: Rekf and rsvsf as optimization techniques for deep-learning. *Neural Networks*, 108:509–526, 2018.

[79] James R Janesick, Tom Elliott, Stewart Collins, Morley M Blouke, and Jack Freeman. Scientific charge-coupled devices. *Optical Engineering*, 26(8):692–714, 1987.

[80] Zhen Jia, A. Balasuriya, and S. Challa. Sensor fusion based 3d target visual tracking for autonomous vehicles with imm. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 1829–1834, 2005.

[81] Víctor Jiménez, Jorge Godoy, Antonio Artuñedo, and Jorge Villagra. Ground segmentation algorithm for sloped terrain and sparse lidar point cloud. *IEEE Access*, 9:132914–132927, 2021.

[82] Xianjian Jin, Hang Yang, Xin Liao, Zeyuan Yan, Qikang Wang, Zhiwei Li, and Zhaoran Wang. A robust gaussian process-based lidar ground segmentation algorithm for autonomous driving. *Machines*, 10(7):507, 2022.

[83] Kichun Jo, Keounyup Chu, and Myoungho Sunwoo. Interacting multiple model filter-based sensor fusion of gps with in-vehicle sensors for real-time vehicle positioning. *IEEE Transactions on Intelligent Transportation Systems*, 13(1):329–343, 2011.

[84] Vijay John and Seiichi Mita. Rvnet: Deep sensor fusion of monocular camera and radar for image-based obstacle detection in challenging environments. In *Pacific-Rim Symposium on Image and Video Technology*, pages 351–364. Springer, 2019.

[85] Kelsey M Judd, Michael P Thornton, and Austin A Richards. Automotive sensing: Assessing the impact of fog on lwir, mwir, swir, visible, and lidar performance. In *Infrared Technology and Applications XLV*, volume 11002, pages 322–334. SPIE, 2019.

[86] Moshe Kam, Xiaoxun Zhu, and Paul Kalata. Sensor fusion for mobile robot navigation. *Proceedings of the IEEE*, 85(1):108–119, 1997.

[87] Gunzung Kim, Jeongsook Eom, and Yongwan Park. An experiment of mutual interference between automotive lidar scanners. In *2015 12th International Conference on Information Technology-New Generations*, pages 680–685. IEEE, 2015.

[88] Je Seok Kim and Jahng Hyon Park. Weighted-graph-based supervoxel segmentation of 3d point clouds in complex urban environment. *Electronics Letters*, 51(22):1789–1791, 2015.

[89] Jihun Kim, Dong Seog Han, and Benaoumeur Senouci. Radar and vision sensor fusion for object detection in autonomous vehicle surroundings. *2018 Tenth International Conference on Ubiquitous and Future Networks (ICUFN)*, 2018.

[90] T. Kim and T. Adali. Complex backpropagation neural network using elementary transcendental activation functions. In *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.01CH37221)*, volume 2, page 1281–1284. IEEE, 2001.

[91] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980 [cs]*, January 2017. arXiv: 1412.6980.

[92] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research).

[93] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-100 (canadian institute for advanced research).

[94] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, May 2017.

[95] Tobias Lang, Christian Plagemann, and Wolfram Burgard. Adaptive non-stationary kernel regression for terrain modeling. In *Robotics: Science and Systems*, volume 6. Citeseer, 2007.

[96] HyeonSeok Lee and Hyo Seon Park. A generalization method of partitioned activation function for complex number. *arXiv:1802.02987 [cs, math]*, February 2018. arXiv: 1802.02987.

[97] Seungjae Lee, Hyungtae Lim, and Hyun Myung. Patchwork++: Fast and robust ground segmentation solving partial under-segmentation using 3D point cloud. In *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, pages 13276–13283, 2022.

[98] Vladimir Lekic and Zdenka Babic. Automotive radar and camera fusion using generative adversarial networks. *Computer Vision and Image Understanding*, 184:1–8, 2019.

[99] Zhixin Leng, Shu Li, Xin Li, and Bingzhao Gao. An improved fast ground segmentation algorithm for 3d point cloud. In *2020 Chinese Control And Decision Conference (CCDC)*, pages 5016–5020. IEEE, 2020.

[100] R Lenz. Lens distortion corrected ccd-camera calibration with co-planar calibration points for real-time 3d measurements. *Prodding of ISPRS, Fast Processing of Photogrammetric Data, 1987*, pages 60–67, 1987.

[101] E Li, Shuaijun Wang, Chengyang Li, Dachuan Li, Xiangbin Wu, and Qi Hao. Sustech points: A portable 3d point cloud interactive annotation platform system. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pages 1108–1115, 2020.

[102] Qingquan Li, Long Chen, Ming Li, Shih-Lung Shaw, and Andreas Nüchter. A sensor-fusion drivable-region and lane-detection system for autonomous vehicle navigation in challenging road scenarios. *IEEE Transactions on Vehicular Technology*, 63(2):540–555, 2013.

[103] Hyungtae Lim, Sungwon Hwang, and Hyun Myung. Erasor: Egocentric ratio of pseudo occupancy-based dynamic object removal for static 3d point cloud map building. *IEEE Robotics and Automation Letters*, 6(2):2272–2279, 2021.

[104] Hyungtae Lim, Oh Minho, and Hyun Myung. Patchwork: Concentric zone-based region-wise ground segmentation with ground likelihood estimation using a 3d lidar sensor. *IEEE Robotics and Automation Letters*, 2021.

[105] Dave Litwiller. Ccd vs. cmos. *Photonics spectra*, 35(1):154–158, 2001.

[106] Jingjing Liu, Shaoting Zhang, Shu Wang, and Dimitris N Metaxas. Multispectral deep neural networks for pedestrian detection. *arXiv preprint arXiv:1611.02644*, 2016.

[107] Shuang Liu. Fourier neural network for machine learning. *2013 International Conference on Machine Learning and Cybernetics*, 2013.

[108] Zhongzhen Luo. *LiDAR Based Perception System: Pioneer Technology for Safety Driving*. PhD thesis, 2017.

[109] Zhongzhen Luo, Martin V Mohrenschildt, and Saeid Habibi. A probability occupancy grid based approach for real-time lidar ground segmentation. *IEEE Transactions on Intelligent Transportation Systems*, 21(3):998–1010, 2019.

[110] Michael Manz, Michael Himmelsbach, Thorsten Luettel, and Hans-Joachim Wuensche. Fusing lidar and vision for autonomous dirt road following: Incorporating a visual feature into the tentacles approach. In *Autonome Mobile Systeme 2009: 21. Fachgespräch Karlsruhe, 3./4. Dezember 2009*, pages 17–24. Springer, 2009.

[111] Akram Marseet and Ferat Sahin. Application of complex-valued convolutional neural network for next generation wireless networks. In *2017 IEEE Western New York Image and Signal Processing Workshop (WNYISPW)*, page 1–5. IEEE, November 2017.

[112] Leandro Masello, German Castignani, Barry Sheehan, Finbarr Murphy, and Kevin McDonnell. On the road safety benefits of advanced driver assistance systems in different driving contexts. *Transportation research interdisciplinary perspectives*, 15:100670, 2022.

[113] Fredrik Matsson. Sensor fusion for positioning of an autonomous vehicle, 2018.

[114] media@nsc.org. My car does what, https://mycardoeswhat.org/, February 2023.

[115] Gledson Melotti, Cristiano Premebida, Nuno MM da S Gonçalves, Urbano JC Nunes, and Diego R Faria. Multimodal cnn pedestrian classification: a study on combining lidar and camera data. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 3138–3143. IEEE, 2018.

[116] Claudio Michaelis, Benjamin Mitzkus, Robert Geirhos, Evgenia Rusak, Oliver Bringmann, Alexander S Ecker, Matthias Bethge, and Wieland Brendel. Benchmarking robustness in object detection: Autonomous driving when winter is coming. *arXiv preprint arXiv:1907.07484*, 2019.

[117] Ben Miethig, Ash Liu, Saeid Habibi, and Martin v Mohrenschildt. Leveraging thermal imaging for autonomous driving. In *2019 IEEE Transportation Electrification Conference and Expo (ITEC)*, pages 1–5. IEEE, 2019.

[118] Frank Moosmann, Oliver Pink, and Christoph Stiller. Segmentation of 3d lidar data in non-flat urban environments using a local convexity criterion. In *2009 IEEE Intelligent Vehicles Symposium*, pages 215–220. IEEE, 2009.

[119] Hans Moravec and Alberto Elfes. High resolution maps from wide angle sonar. In *Proceedings. 1985 IEEE international conference on robotics and automation*, volume 2, pages 116–121. IEEE, 1985.

[120] N Muhammad and S Lacroix. Calibration of a rotating multi-beam lidar. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, page 5648–5653, Taipei, October 2010. IEEE.

[121] Kiin Na, Jaemin Byun, Myongchan Roh, and Bumsu Seo. The ground segmentation of 3d lidar point cloud with the optimized region merging. In *2013 International Conference on Connected Vehicles and Expo (ICCVE)*, pages 445–450. IEEE, 2013.

[122] Patiphon Narksri, Eijiro Takeuchi, Yoshiki Ninomiya, Yoichi Morales, Naoki Akai, and Nobuo Kawaguchi. A slope-robust cascaded ground segmentation in 3d point cloud for autonomous vehicles. In *2018 21st International Conference on intelligent transportation systems (ITSC)*, pages 497–504. IEEE, 2018.

[123] NHTSA. Critical reasons for crashes investigated in the national motor vehicle crash causation survey, 2015.

[124] Felix Nobis, Maximilian Geisslinger, Markus Weber, Johannes Betz, and Markus Lienkamp. A deep learning-based radar and camera sensor fusion architecture for object detection. In *2019 Sensor Data Fusion: Trends, Solutions, Applications (SDF)*, pages 1–7. IEEE, 2019.

[125] Statistics Canada. Government of Canada and Statistique Canada. Gouvernement du Canada. Statistics canada: Canada's national statistical agency / statistique canada : Organisme statistique national du canada, December 2012.

[126] Minho Oh, Euigon Jung, Hyungtae Lim, Wonho Song, Sumin Hu, Eungchang Mason Lee, Junghee Park, Jaekyung Kim, Jangwoo Lee, and Hyun Myung. TRAVEL: Traversable ground and above-

ground object segmentation using graph representation of 3D LiDAR scans. In *IEEE Robotics and Automation Letters*, 2022. Submitted.

[127] Sang-Il Oh and Hang-Bong Kang. Object detection and classification by decision-level fusion for intelligent vehicle systems. *Sensors*, 17(1):207, 2017.

[128] World Health Organization. *Save lives: a road safety technical package*. 2023.

[129] Stanislaw Osowski and Do Dinh Nghia. Fourier and wavelet descriptors for shape recognition using neural networks—a comparative study. *Pattern Recognition*, 35(9):1949–1957, September 2002.

[130] Anshul Paigwar, Özgür Erkent, David Sierra-Gonzalez, and Christian Laugier. Gndnet: Fast ground plane estimation and point cloud segmentation for autonomous vehicles. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2150–2156. IEEE, 2020.

[131] Yoonsu Park, Seokmin Yun, Chee Sun Won, Kyungeun Cho, Kyhyun Um, and Sungdae Sim. Calibration between color camera and 3d lidar instruments with a polygonal planar board. *Sensors*, 14(3):5333–5353, 2014.

[132] Richard G. Pearson. Human factors aspects of lightplane safety. *PsycEXTRA Dataset*, 1963.

[133] Matthew Pitropov, Danson Evan Garcia, Jason Rebello, Michael Smart, Carlos Wang, Krzysztof Czarnecki, and Steven Waslander. Canadian adverse driving conditions dataset. *The International Journal of Robotics Research*, 40(4-5):681–690, 2021.

[134] Calin-Adrian Popa. Complex-valued convolutional neural networks for real-valued image classification. *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017.

[135] Calin-Adrian Popa and Cosmin Cernazanu-Glavan. Fourier transform-based image classification using complex-valued convolutional neural networks. *Advances in Neural Networks ? ISNN 2018 Lecture Notes in Computer Science*, page 300?309, 2018.

[136] Gerald B Popko, Thomas K Gaylord, and Christopher R Valenta. Geometric approximation model of inter-lidar interference. *Optical Engineering*, 59(3):033104, 2020.

[137] Harry Pratt, Bryan Williams, Frans Coenen, and Yalin Zheng. Fcnn: Fourier convolutional neural networks. *Machine Learning and Knowledge Discovery in Databases Lecture Notes in Computer Science*, page 786?798, 2017.

[138] Danil V. Prokhorov. Object recognition in 3d lidar data with recurrent neural network. *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, 2009.

[139] Charles R. Qi, Hao Su, Matthias Niebner, Angela Dai, Mengyuan Yan, and Leonidas J. Guibas. Volumetric and multi-view cnns for object classification on 3d data. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[140] Renishaw. *Image of a rotating LiDAR unit. Image courtesy of Renishaw.* February 2018.

[141] SAE. Sae international releases updated visual chart for its "levels of driving automation" standard for self-driving vehicles, December 2018.

[142] Abraham. Savitzky and M. J. E. Golay. Smoothing and differentiation of data by simplified least squares procedures. *Analytical Chemistry*, 36(8):1627–1639, 1964.

[143] Davide Scaramuzza, Ahad Harati, and Roland Siegwart. Extrinsic self calibration of a camera and a 3d laser range finder from natural scenes. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4164–4169. IEEE, 2007.

[144] Simone Scardapane, Steven Van Vaerenbergh, Amir Hussain, and Aurelio Uncini. Complex-valued neural networks with non-parametric activation functions. *arXiv:1802.08026 [cs]*, February 2018. arXiv: 1802.08026.

[145] Joel Schlosser, Christopher K Chow, and Zsolt Kira. Fusing lidar and images for pedestrian detection using convolutional neural networks. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2198–2205. IEEE, 2016.

[146] Steve Shafer, Anthony Stentz, and Charles Thorpe. An architecture for sensor fusion in a mobile robot. In *Proceedings. 1986 IEEE International Conference on Robotics and Automation*, volume 3, pages 2002–2011. IEEE, 1986.

[147] Adnan Shaout, Dominic Colella, and S. Awad. Advanced driver assistance systems - past, present and future. In *2011 seventh International Computer Engineering Conference (ICENCO'2011)*, page 72–82. IEEE, December 2011.

[148] Zhihao Shen, Huawei Liang, Linglong Lin, Zhiling Wang, Weixin Huang, and Jie Yu. Fast ground segmentation for 3d lidar point cloud based on jump-convolution-process. *Remote Sensing*, 13(16):3239, 2021.

[149] Tomoyoshi Shimobaba, Yutaka Endo, Ryuji Hirayama, Yuki Nagahama, Takayuki Takahashi, Takashi Nishitsuji, Takashi Kakue, Atsushi Shiraki, Naoki Takada, Nobuyuki Masuda, and et al. Autoencoder-based holographic image restoration. *Applied Optics*, 56(13), 2017.

[150] Wei Song, Lingfeng Zhang, Yifei Tian, Simon Fong, Jinming Liu, and Amanda Gozho. Cnn-based 3d object classification using hough space of lidar point clouds. *Human-centric Computing and Information Sciences*, 10:1–14, 2020.

[151] Ioannis Stamos, Olympia Hadjiliadis, Hongzhong Zhang, and Thomas Flynn. Online algorithms for classification of urban objects in 3d point clouds. In *2012 Second International Conference on 3D Imaging, Modeling, Processing, Visualization & Transmission*, pages 332–339. IEEE, 2012.

[152] Alan N Steinberg and Christopher L Bowman. Revisions to the jdl data fusion model. In *Handbook of multisensor data fusion*, pages 65–88. CRC press, 2017.

[153] Ashley W Stroupe, Martin C Martin, and Tucker Balch. Distributed sensor fusion for object position estimation by multi-robot systems. In *Proceedings 2001 ICRA. IEEE international conference on robotics and automation (Cat. No. 01CH37164)*, volume 2, pages 1092–1098. IEEE, 2001.

[154] Jae Kyu Suhr, Jeungin Jang, Daehong Min, and Ho Gi Jung. Sensor fusion-based low-cost vehicle localization system for complex urban environments. *IEEE Transactions on Intelligent Transportation Systems*, 18(5):1078–1086, 2016.

[155] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, Vijay Vasudevan, Wei Han, Jiquan Ngiam, Hang Zhao, Aleksei Timofeev, Scott Ettinger, Maxim Krivokon, Amy Gao, Aditya Joshi, Yu Zhang, Jonathon Shlens, Zhifeng Chen, and Dragomir Anguelov. Scalability in perception for autonomous driving: Waymo open dataset. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[156] Zehang Sun, G. Bebis, and R. Miller. On-road vehicle detection: a review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(5):694–711, May 2006.

[157] Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, Kenny Lau, Celia Oakley, Mark Palatucci, Vaughan Pratt, Pascal Stang, Sven Strohband, Cedric Dupont, Lars-Erik Jendrossek, Christian Koelen, Charles Markey, Carlo Rummel, Joe van Niekerk, Eric Jensen, Philippe Alessandrini,

Gary Bradski, Bob Davies, Scott Ettinger, Adrian Kaehler, Ara Nefian, and Pamela Mahoney. Stanley: The robot that won the darpa grand challenge. *Journal of Field Robotics*, 23(9):661–692, September 2006.

[158] Chen Tongtong, Dai Bin, Liu Daxue, Zhang Bo, and Liu Qixu. 3d lidar-based ground segmentation. In *The First Asian Conference on Pattern Recognition*, pages 446–450. IEEE, 2011.

[159] Chiheb Trabelsi, Olexa Bilaniuk, Ying Zhang, Dmitriy Serdyuk, Sandeep Subramanian, João Felipe Santos, Soroush Mehri, Negar Rostamzadeh, Yoshua Bengio, and Christopher J. Pal. Deep complex networks. *arXiv:1705.09792 [cs]*, May 2017. arXiv: 1705.09792.

[160] R. Tsai. A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses. *IEEE Journal on Robotics and Automation*, 3(4):323–344, August 1987.

[161] Mark Tygert, Joan Bruna, Soumith Chintala, Yann Lecun, Serkan Piantino, and Arthur Szlam. A mathematical motivation for complex-valued convolutional networks. *Neural Computation*, 28(5):815?825, 2016.

[162] Martin Velas, Michal Spanel, Michal Hradis, and Adam Herout. Cnn for very fast ground segmentation in velodyne lidar data. In *2018 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 97–103. IEEE, 2018.

[163] Gustavo Velasco-Hernandez, De Jong Yeong, John Barry, and Joseph Walsh. Autonomous driving architectures, perception and data fusion: A review. In *2020 IEEE 16th International Conference on Intelligent Computer Communication and Processing (ICCP)*, page 315–321, Cluj-Napoca, Romania, September 2020. IEEE.

[164] Anh-Vu Vo, Linh Truong-Hong, Debra F Laefer, and Michela Bertolotto. Octree-based region growing for point cloud segmentation. *ISPRS Journal of Photogrammetry and Remote Sensing*, 104:88–100, 2015.

[165] Walter. Plane fit ground filter, 2019.

[166] Nick Waltham. Ccd and cmos sensors. In *Observing photons in space*, pages 423–442. Springer, 2013.

[167] Georg Waltner, Michael Maurer, Thomas Holzmann, Patrick Ruprecht, Michael Opitz, Horst Possegger, Friedrich Fraundorfer, and Horst Bischof. Deep 2.5d vehicle classification with sparse sfm depth

prior for automated toll systems. *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, 2018.

[168] Dingkang Wang, Connor Watkins, and Huikai Xie. Mems mirrors for lidar: a review. *Micromachines*, 11(5):456, 2020.

[169] Juyang Weng, Paul Cohen, Marc Herniou, et al. Camera calibration with distortion models and accuracy evaluation. *IEEE Transactions on pattern analysis and machine intelligence*, 14(10):965–980, 1992.

[170] R. E. Wengert. A simple automatic derivative evaluation program. *Communications of the ACM*, 7(8):463–464, 1964.

[171] Wilhelm Wirtinger. Zur formalen theorie der funktionen von mehr complexen ver anderlichen. *Mathematische Annalen*, pages 357–75, 1927.

[172] Bichen Wu, Alvin Wan, Xiangyu Yue, and Kurt Keutzer. Squeezeseg: Convolutional neural nets with recurrent crf for real-time road-object segmentation from 3d lidar point cloud. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 1887–1893. IEEE, 2018.

[173] Bisheng Yang and Zhen Dong. A shape-based segmentation method for mobile laser scanning point clouds. *ISPRS journal of photogrammetry and remote sensing*, 81:19–30, 2013.

[174] Kun Zeng, Jun Yu, Ruxin Wang, Cuihua Li, and Dacheng Tao. Coupled deep autoencoder for single image super-resolution. *IEEE Transactions on Cybernetics*, 47(1):27–37, 2017.

[175] Dimitris Zermas, Izzat Izzat, and Nikolaos Papanikolopoulos. Fast segmentation of 3d point clouds: A paradigm on lidar data for autonomous vehicle applications. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5067–5073. IEEE, 2017.

[176] Guohui Zhang, Ryan P. Avery, and Yinhai Wang. Video-based vehicle detection and classification system for real-time traffic data collection using uncalibrated video cameras. *Transportation Research Record: Journal of the Transportation Research Board*, 1993(1):138–147, January 2007.

[177] Jingzhao Zhang, Tianxing He, Suvrit Sra, and Ali Jadbabaie. Why gradient clipping accelerates training: A theoretical justification for adaptivity. *arXiv preprint arXiv:1905.11881*, 2019.

[178] Qilong Zhang and R. Pless. Extrinsic calibration of a camera and laser range finder (improves camera calibration). In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, page 2301–2306. IEEE, 2004.

[179] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, November 2000.

[180] 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)*, pages 5562–5569. IEEE, 2018.

[181] Hassan Zoghi, Kianoush Siamardi, and Morteza Tolouei. Ada systems application for traffic safety improvement on roadways. In *2010 The 2nd International Conference on Computer and Automation Engineering (ICCAE)*, volume 2, pages 629–634. IEEE, 2010.